Rapport TIPE :

Reconnaissance du code postal sur une enveloppe

Introduction

Plus petit, le fait que la Nintendo DS puisse reconnaître ce que l'on écrivait avec le stylet m'avait beaucoup impressionné. J'ai alors vu le TIPE comme une occasion de comprendre comment cela était possible et j'en ai profité pour appliquer la reconnaissance de chiffres manuscrits à la lecture des codes postaux sur les enveloppes. La problématique suivante s'est alors naturellement présentée : Comment peut-on développer un programme permettant d'obtenir le code postal inscrit sur une enveloppe à partir de sa photo ?

La démarche que j'ai suivie est alors composée de deux parties assez indépendantes. Une première partie plutôt orientée sur du traitement d'image qui me permet de récupérer les images des chiffres composant le code postal sur les photos d'enveloppes. Puis une seconde partie centrée sur la reconnaissance des chiffres via des réseaux de neurones principalement.

Table des Matières :

I. Obtention des chiffres composant le code postal

La première étape est donc l'extraction des chiffres de l'enveloppe. Après avoir pris connaissance de quelques filtres d'images, j'ai pu avancer pas à pas pour ne garder finalement que les cases du code postal. Après avoir constaté que la couleur orange des cases variait beaucoup d'une photo à l'autre j'ai décidé de commencer par appliquer un filtre de détection de contour appelé filtre de Sobel (figure 1). On applique deux matrices de convolutions à l'image

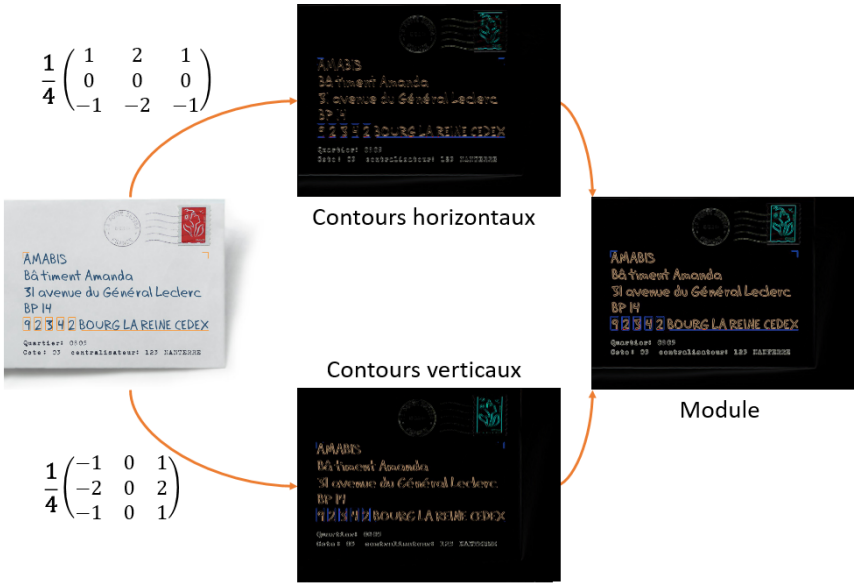

Figure 1 : Filtre de Sobel

pour obtenir les contours horizontaux et verticaux. Puis on prend le module de ces deux images pour obtenir les contours [6]. En récupérant les contours de plusieurs images j'ai remarqué que la couleur bleue obtenue pour les contours du code postal variait peu d'une photo à l'autre. J'ai alors appliqué un filtre ne conservant que cette couleur. Puis j'ai utilisé un filtre gaussien afin d'épaissir les traits pour se faire rejoindre les contours des cases du code postal et aussi pour assombrir et presque effacer les restes d'écriture qui ne sont pas parti avec le filtrage des couleurs. Je termine enfin par rendre noir les pixels trop sombres et si

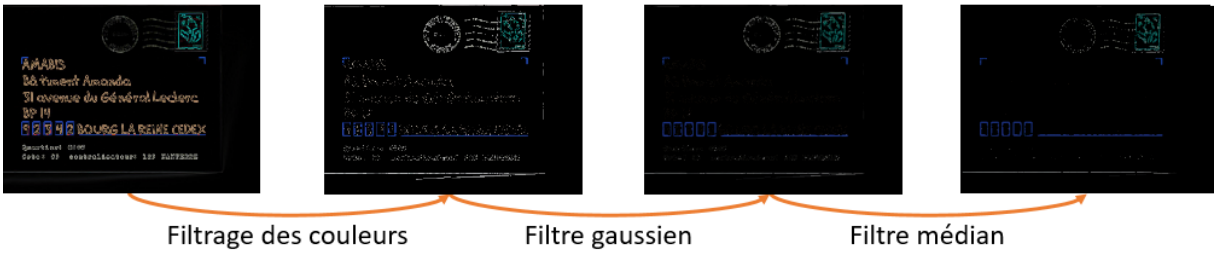

Figure 2 : Traitement des contours

l'image est assez grande j'applique un filtre médian me permettant de « casser » des contours qui seraient trop fins et surtout d'enlever des pixels isolés généralement dus à un léger bruit. Cette dernière étape sert alors à en quelque sorte nettoyer l'image pour l'étape suivante (figure 2) [7]. Les paramètres des filtres ont été ajusté sur une dizaine d'images, si les filtres ne semblent pas complétement utiles ici, ils peuvent l'être pour d'autres images. Grâce à la dernière image je crée une liste de toutes les composantes connexes qui ne sont pas noires et via une formule trouvée empiriquement portant sur la hauteur et la largeur des composantes, je détermine quelle composante semble le plus probablement être le code postal. En récupérant cette dernière (première image de la figure 3) j'applique des filtres sélectionnant les couleurs à garder sur la composante et si tout se passe bien, on récupère une image composée de cinq nouvelles zones connexes qui sont les chiffres que l'on peut alors séparer en cinq images (figure 3).

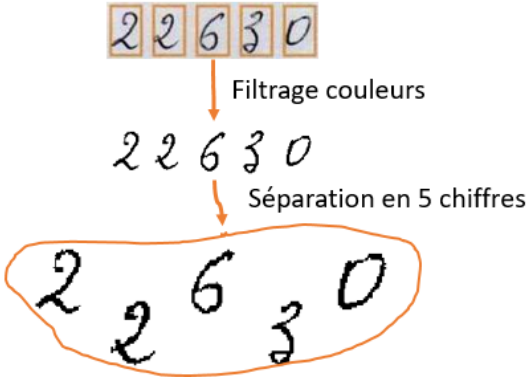

Figure 3 : Séparation

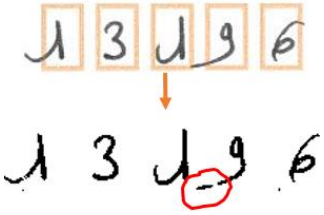

Figure 4 : Première erreur

En testant sur une autre dizaine d'images, les résultats sont toujours bon sauf pour une image où le contraste entre les rectangles orange et l'écriture n'est pas assez marqué (figure 4). Le programme est donc plutôt correct mais le temps de calcul reste un peu long (4.9s en moyenne avec un maximum de 14.8s pour les photos de dimensions maximales) et il n'y a pour l'instant pas moyen d'être sûr d'avoir récupérer les chiffres du code postal.

II. Reconnaissance des chiffres

Maintenant que les images des chiffres peuvent être récupérés séparément il faut pouvoir obtenir la valeur des chiffres sur les images. Pour cela j'ai implémenté deux programmes qui prennent en entrée 100 « variables caractéristiques » récupérées à partir de l'image d'un chiffre et un troisième programme dont nous parlerons en dernier prenant, lui, tous les pixels de l'image en entrée. Les variables caractéristiques que j'ai choisies ont été imaginées par Gilles Burel dans sa thèse [1]. Elles sont composées de 20 variables indiquant le pourcentage de pixels noir dans certaines zones, et de 80 variables indiquant la distance devant être parcouru pour rencontrer un premier pixel noir en partant d'un pixel du bord de l'image et en se déplaçant en ligne droite (suivant trois directions différentes formant ainsi trois profils, voir figure 5). Les programmes demandent aussi des données pour s'entraîner. J'ai donc écrit et demandé à des amis d'écrire des chiffres grâce à une application que j'ai écrite. Aujourd'hui la base de données contient un peu plus d'une centaine d'images par chiffres. Je me suis gardé 200 exemples pour tester les programmes et tout le reste des images sert à l'apprentissage.

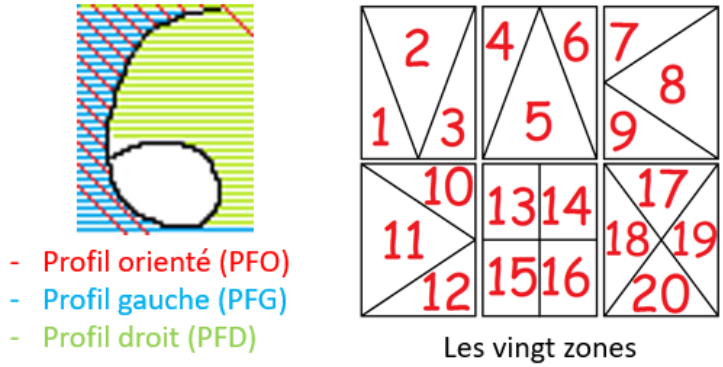

Figure 5 : Variables caractéristiques

a) Classification bayésienne

Le premier programme est une classification bayésienne. Son fonctionnement est purement mathématique. J'ai calculé la moyenne et la variance des variables caractéristiques sur la base d'apprentissage pour chaque chiffre. Grâce à ces valeurs j'ai modélisé les densités de probabilité des variables caractéristiques sachant le chiffre de l'image par des lois normales. Si l'on présente alors une image à l'algorithme on peut obtenir la probabilité que l'image soit tel ou tel chiffre sachant les variables caractéristiques (supposées indépendantes) via les formules de Bayes (figure 6), où le premier facteur est supposé indépendant de k car un chiffre n'est pas plus probable qu'un autre dans le code postal.

C variable de classe dans $[\![0, 9]\!]$.
(X_i) variables caractéristiques mutuellement indépendantes.

Formule de Bayes : $\mathbb{P}(C = k \mid X_1 \ \dots \ X_{100}) = \frac{\mathbb{P}(C=k)}{\mathbb{P}(X_1 \dots X_{100})} \prod_{i=1}^{100} \mathbb{P}(X_i \mid C = k)$

Indépendant de k — Modélisé par loi normale

Figure 6 : Classification bayésienne

b) Perceptron multicouche

Le second programme est un perceptron multicouche, un certain type de réseau de neurones à apprentissage supervisé. J'ai étudié ce réseau durant l'année de MPSI avec un autre élève de ma classe avant qu'il finisse par partir sur un autre sujet. Ensemble nous avons observé l'influence de différents paramètres qui interviennent dans un tel réseau (coefficient d'apprentissage, d'inertie, nombre de couches …) dans un premier temps pour l'apprentissage de la fonction XOR puis sur la reconnaissance de chiffres. Sur ce dernier nous avons par exemple essayer différentes manières de présenter au réseau les images de la base d'apprentissage (de façon aléatoire, de manière cyclique, par « batch », c'est-à-dire que l'on présente plusieurs images au réseau avant de modifier les poids). On aura finalement retenu l'apprentissage en présentant les images aléatoirement et sans batchs. Nous avons aussi pu observer le phénomène de surapprentissage. Cela m'a permis de savoir quand arrêter l'apprentissage des réseaux. Il suffisait d'arrêter l'apprentissage lorsque l'erreur sur la base d'évaluation cessait de décroître. Finalement la structure du perceptron utilisé est assez similaire à celle décrite par Gilles Burel dans le chapitre 8 de sa thèse [1] et prend en entrée les 100 variables caractéristiques. L'apprentissage se fait par rétropropagation du gradient [3]. J'ai créé une interface avec notamment une reconnaissance en temp réel (figure 7). Les performances sont plutôt satisfaisantes puisque pour un million d'apprentissages il me faut moins de dix minutes sachant que le réseau comporte plus de 2200 axones (liaisons entre neurones). Finalement après deux millions d'apprentissages le réseau que j'ai retenu reconnaît 199 images des 200 images de ma base d'évaluation. De plus la somme des erreurs quadratiques commises est de 0.258 sur l'ensemble de la base. On pourra comparer cette erreur avec le réseau qui suit.

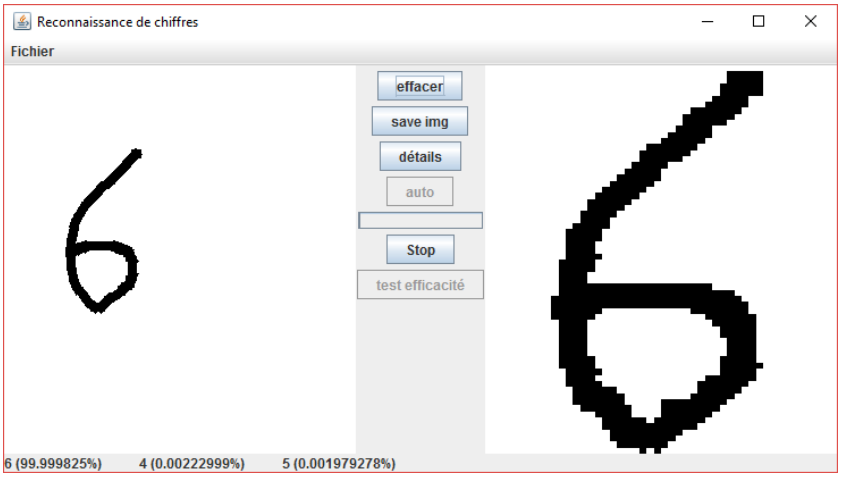

Figure 7 : Logiciel d'entrainement pour perceptron multicouche

c) Deep Belief Network

Le troisième programme est encore un réseau de neurones utilisant le 'deep learning' en prenant tous les pixels en entrée. Il est constitué de deux machines de Boltzmann fonctionnant par apprentissage non supervisé suivi d'un perceptron [2, 4, 5]. Comme pour le perceptron, j'ai commencé par étudier les différents paramètres et j'ai notamment choisi pour ce type de réseau un apprentissage avec batchs et un coefficient d'apprentissage beaucoup plus faible. J'obtiens avec ce type de réseau, des résultats moins satisfaisant, puisque six images ne sont pas correctement reconnues sur les 200 présentes dans la base d'évaluation

et la somme des erreurs quadratiques est de 7.65 soit près de 30 fois plus élevé que pour le perceptron. Ce réseau est donc beaucoup moins sûr de lui lorsqu'il donne une réponse que le perceptron. J'ai aussi pu réaliser grâce à ce dernier plusieurs expériences de mon côté comme l'inversion du réseau pour générer des chiffres par l'ajout d'une autre machine de Boltzmann restreinte (la figure 8 montre la génération d'un chiffre 2 à partir d'un 7). C'est d'ailleurs une des raisons qui m'a motivé à essayer cette architecture.

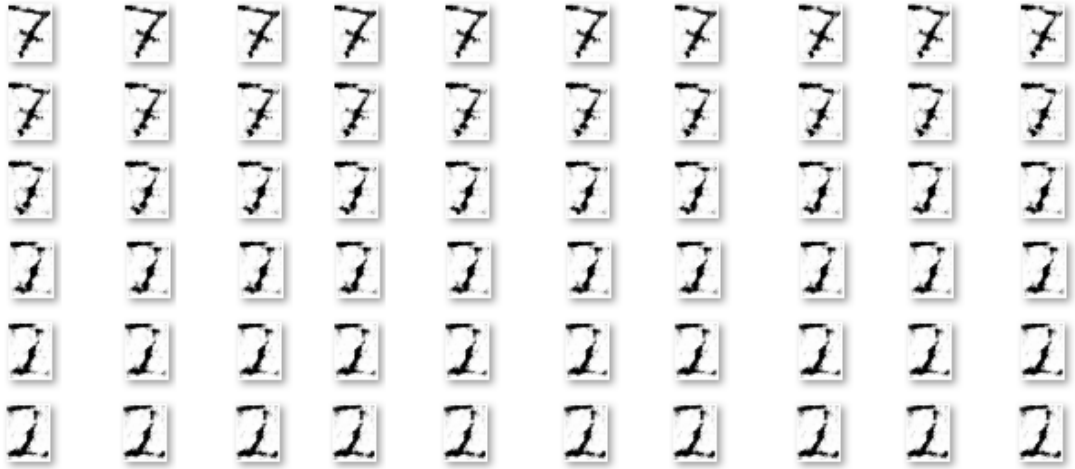

Figure 8 : Génération du chiffre 2 à partir d'un 7

Conclusion

J'obtiens alors bien le logiciel souhaité (figure 10) pouvant même utiliser trois méthodes différentes pour reconnaître les chiffres. Les erreurs dues à la première partie, c'est-à-dire à la récupération des cinq petites images de chiffres sont plutôt rares (1 cas sur 18) et pourrait être corrigé avec un peu plus de temps. Pour ce qui est de la reconnaissance des chiffres, la figure 9 résume les pourcentages de réussite des différents programmes sur la base d'évaluation et sur les chiffres écrits sur les 17 enveloppes. On remarque que le perceptron est le plus performant pour ce problème, le réseau bayésien étant trop « simpliste » et le deep learning plus utile pour de plus gros problèmes. De plus les résultats sont moins bons sur les enveloppes à causes de certaines photos de faible qualité. Avec de meilleurs photos les résultats pourraient tendre vers ceux de la base d'évaluation. Enfin je n'ai pu faire que peu d'apprentissages avec le deep belief network à cause du temps de calcul assez élevé. Lorsque j'ai arrêté (à un peu plus d'un million d'apprentissages par machines de Boltzmann) le réseau semblait encore pouvoir s'améliorer. L'idéal aurait été d'utiliser une carte graphique pour faire les calculs.

PROGRAMME	BASE D'EVALUATION (200 IMAGES)	ENVELOPPES (17×5 IMAGES)
RESEAU BAYESIEN	65.%	44%
PERCEPTRON	99.5%	94%
DEEP LEARNING	97%	78%

Figure 9 : Résultats des réseaux

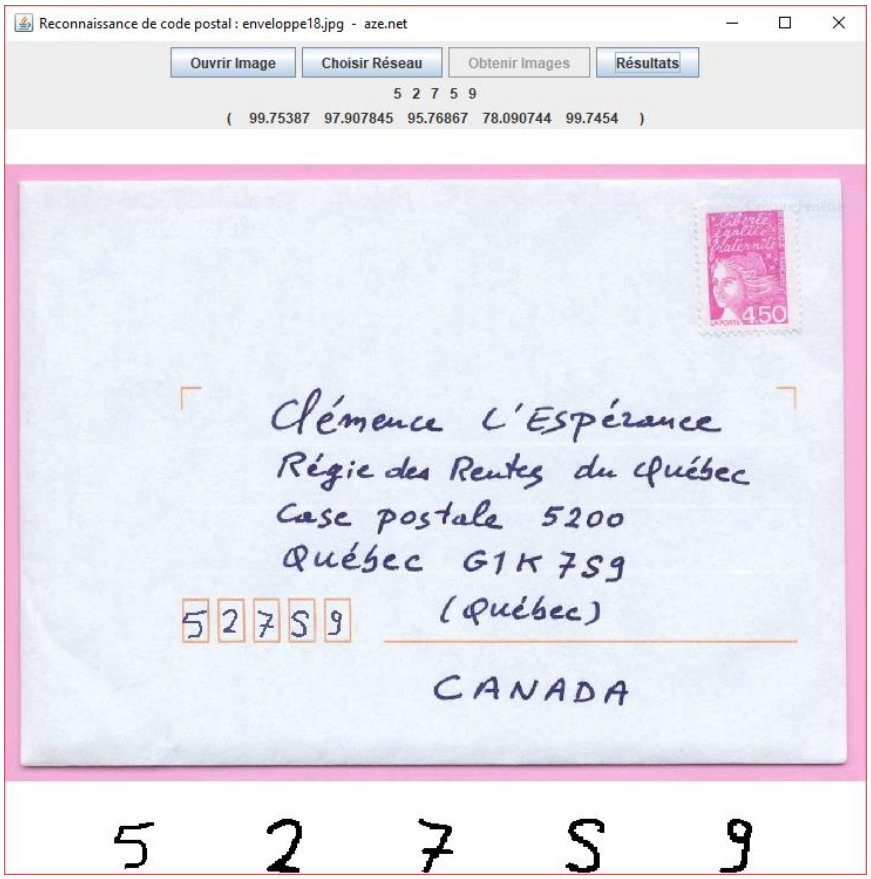

Figure 10 : Logiciel final

Bibliographie

[1] BUREL GILLES, *Réseaux de neurones en traitement d'images - Des modèles théoriques aux applications industrielles*, Thèse, Université de Bretagne Occidentale-Brest, 1991

[2] LAROCHELLE HUGO, YouTube, *Hugo Larochelle – YouTube*, https://www.youtube.com/user/hugolarochelle , (notamment le chapitre 12-Apprentissage automatique de sa playlist Intelligence artificielle ainsi que sa playlist Neural networks class - Université de Sherbrooke), dernière consultation le 2 Mars 2017, dernière modification le 15 Novembre 2013

[3] AUTEUR INCONNU, Wikepedia, *Rétropropagation du gradient*, https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient , dernière consultation 20 Mars 2016, dernière modification le 8 janvier 2017

[4] GEOFFREY HINTON, SIMON OSINDERO, YEE-WHYE TEH, *Neural Computation*, juillet 2006 (Vol. 18), pages 1527-1554, A fast learning algorithm for deep belief nets

[5] QUENTIN FRESNEL : *Apprentissage de descripteurs audio par Deep learning, application pour la classification en genre musical*, Rapport de stage, Telecom Paritech, 2015

[6] JEAN-HUGH THOMAS, Optique pour l'ingénieur, *Détection de contours - Quelques opérateurs gradient*,

http://www.optique-ingenieur.org/fr/cours/OPI_fr_M04_C05/co/Contenu_02.html , dernière consultation le 12 Décembre 2016, dernière modification en Juillet 2007

[7] BERGOUNIOUX MAITINE, *Introduction au traitement mathématique des images - méthodes déterministes*, Springer, 20 février 2015, http://www.springer.com/la/book/9783662465387

Contacts

[1] HUGO LAROCHELLE, Professeur / Chercheur, *Université de Sherbooke / Google Brain (Montréal)*, premier contact : 13 Mai 2016

[2] MAITINE BERGOUNIOUX, Professeur de Mathématiques, *Université d'Orléans*, premier contact : 4 Décembre 2016

Annexes

Dans les annexes se trouvent tout le code écrit en java. On trouvera dans l'ordre les classes suivantes :

```java
import java.awt.Color;

/**
 * Classe qui contient tout ce qui est utile au traitement de l'image.
 * @author Yoann
 */
public class ImageUtil {

    /**
     * Permet de récupérer les informations sur les proportions de pixels dans les 20 zones.
     * @param image L'image à traiter.
     * @return Le tableau contenant les 20 pourentages (un par zone).
     */
    public static float[] stats(BufferedImage image) {
        float[] res = new float[20];
        float w = image.getWidth(), h = image.getHeight();
        float mx = w-1;
        float s = w*h;
        for(float x = 0; x < w; x++)
            for(float y = 0; y < h; y++) {
                int c = image.getRGB((int) x, (int) y);
                if(c < -10000000) {
                    if(y >= 2*x*h/w) res[0] += 4/s;
                    else if(y <= 2*(mx-x)*h/w) res[1] += 2/s;
                    else res[2] += 4/s;
                    if(y <= (mx-2*x)*h/w) res[3] += 4/s;
                    else if(y >= (2*x-mx)*h/w) res[4] += 2/s;
                    else res[5] += 4/s;
                    if(y <= (mx-x)*h/2/w) res[6] += 4/s;
                    else if(y <= (mx+x)*h/2/w) res[7] += 2/s;
                    else res[8] += 4/s;
                    if(y <= x*h/2/w) res[9] += 4/s;
                    else if(y <= (2*mx-x)*h/2/w) res[10] += 2/s;
                    else res[11] += 4/s;
                    if(y < h/2) {
                        if(x < w/2) res[12] += 4/s;
                        else res[13] += 4/s;
                    } else {
                        if(x < w/2) res[14] += 4/s;
                        else res[15] += 4/s;
                    }
                    if(y <= x*h/w) {
                        if(y <= (mx-x)*h/w) res[16] += 4/s;
                        else res[18] += 4/s;
                    } else {
                        if(y <= (mx-x)*h/w) res[17] += 4/s;
                        else res[19] += 4/s;
                    }
                }
            }
        return res;
    }

    /**
     * Permet de récupérer le profile gauche (PFG) de l'image.
     * @param image L'image à traiter.
     * @return Le profle gauche sous forme d'un tableau de taille 32.
     */
    public static float[] pfg(BufferedImage image) {
        float[] res = new float[32];
        for(float y = 0; y < 32; y++) {
            float x = 0;
            while(x < 48 && image.getRGB((int) x, 2*((int) y)) > -10000000)
```

```
71                  x ++;
72              res[(int) y] = (x-24)/24;
73          }
74          return res;
75      }
76
77      /**
78       * Permet de récupérer le profile droit (PFD) de l'image.
79       * @param image L'image à traiter.
80       * @return Le profle droit sous forme d'un tableau de taille 32.
81       */
82      public static float[] pfd(BufferedImage image) {
83          float[] res = new float[32];
84          for(float y = 0; y < 32; y++) {
85              float x = 47;
86              while(x >= 0 && image.getRGB((int) x, 2*((int) y)) > -10000000)
87                  x --;
88              res[(int) y] = (x-24)/24;
89          }
90          return res;
91      }
92
93      /**
94       * Permet de récupérer le profile orienté (PFO) de l'image.
95       * @param image L'image à traiter.
96       * @return Le profle orienté sous forme d'un tableau de taille 16.
97       */
98      public static float[] pfo(BufferedImage image) {
99          float[] res = new float[16];
100         for(int i = 0; i < 16; i ++) {
101             float x = -33 + 3*i, y = 17 - 3*i;
102             while(x < 0 || y < 0) {
103                 x ++;
104                 y ++;
105             }
106             float d = 0, len = Math.min(48-x, 64-y);
107             while(x <= 47 && y <= 63 && image.getRGB((int) x, (int) y) > -10000000)
    {
108                 x ++;
109                 y ++;
110                 d ++;
111             }
112             res[i] = (d-len/2)*2/len;
113         }
114         return res;
115     }
116
117     /**
118      * Permet de récupéré toutes les infos nécessaire au réseau.
119      * @param im L'image à traiter.
120      * @return Le tableu des 100 infos nécessaires.
121      */
122     public static float[] allinfo(BufferedImage im) {
123         float[] res = new float[100];
124         float[] pfo = pfo(im), pfg = pfg(im), pfd = pfd(im), stats = stats(im);
125         for(int i = 0; i < 16; i++)
126             res[i] = pfo[i];
127         for(int i = 0; i < 32; i++)
128             res[i+16] = pfg[i];
129         for(int i = 0; i < 32; i++)
130             res[i+48] = pfd[i];
131         for(int i = 0; i < 20; i++)
132             res[i+80] = stats[i];
133         return res;
```

```
134     }
135
136     /**
137      * Retourne un tableau de pixels correspondant à une image.
138      * @param im L'image dont on veut récupérer le tableau.
139      * @return Un tableau contenant dans chaque case 1 si le pixel est noir, 0
    sinon. Les pixles sont disposés lignes par lignes dans
140      * le tableau.
141      */
142     public static float[] pixels(BufferedImage im) {
143         int w = im.getWidth(), h = im.getHeight();
144         float[] res = new float[w*h];
145         for(int x = 0; x < w; x++)
146             for(int y = 0; y < h; y++)
147                 res[x+y*w] = (im.getRGB(x, y) < -10000000) ?  1 : 0;
148         return res;
149     }
150
151     /**
152      * Crée une image aléatoire de pixels noir ou blanc.
153      * @param len La nombre de pixels à générés (w*h).
154      * @return La liste de pixels aléatoires.
155      */
156     public static float[] randomImage(int len) {
157         float[] x = new float[len];
158         for(int a = 0; a < len; a++)
159             x[a] = (float) Math.random();
160         return x;
161     }
162
163     /**
164      * Retourne une image en noir et blanc à partir d'un tableau de pixels.
165      * @param pix Le tableau des pixels.
166      * @param w La largeur de la nouvelle image.
167      * @param h La hauteur de la nouvelle image.
168      * @return L'image voulue.
169      */
170     public static BufferedImage ImageFromPixels(float[] pix, int w, int h) {
171         int[] pix2 = new int[w*h];
172         for(int i = 0; i < w*h; i++) {
173             int c = (int) ((1-pix[i])*255);
174             pix2[i] = c + (c << 8) + (c << 16);
175         }
176         BufferedImage res = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
177         res.setRGB(0, 0, w, h, pix2, 0, w);
178         return res;
179     }
180
181     /**
182      * Retourne une image en noir et blanc adapté aux représsentations des DBN et
    RBM à partir d'un tableau de pixels.
183      * @param pix Le tableau des pixels.
184      * @param w La largeur de la nouvelle image.
185      * @param h La hauteur de la nouvelle image.
186      * @return L'image voulue.
187      */
188     public static BufferedImage ImageFromPixelsB(float[] pix, int w, int h) {
189         int[] pix2 = new int[w*h];
190         for(int i = 0; i < w*h; i++) {
191             int c = (int) (Math.max(Math.min(0.5+Math.pow(pix[i], 3)*0.9,
    0.5),-0.5)*255);
192             pix2[i] = c + (c << 8) + (c << 16);
193         }
194         BufferedImage res = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
```

```
195         res.setRGB(0, 0, w, h, pix2, 0, w);
196         return res;
197     }
198 
199     /**
200      * Pour obtenir la matrice de Gauss de taille (2<b>n</b>+1) et de paramètre <b>s</b>.
201      * @param n Le paramètre de taille.
202      * @param s Le paramètre de la gaussiène.
203      * @return La matrice voulue.
204      */
205     public static float[][] matGauss(int n, float s) {
206         float[][] res = new float[2*n+1][2*n+1];
207         float tot = res[n][n] = gauss(0, 0, s);
208         for(byte i = 1; i <= n; i++) {
209             for(byte j = 0; j <= n; j++) {
210                 res[n-j][n+i] = res[n-i][n-j] = res[n+j][n-i] = res[n+i][n+j] = gauss(i, j, s);
211                 tot += 4*res[i+n][j+n];
212             }
213         }
214         for(int i = -n; i <= n; i++)
215             for(int j = -n; j <= n; j++)
216                 res[i+n][j+n] /= tot;
217         return res;
218     }
219 
220     private static float gauss(float x,float y, float s) { return (float) Math.exp((-x*x-y*y)/2/(s*s)); } //La gaussienne
221 
222     /**
223      * Effectue un lissage gaussien sur une image.
224      * @param image L'image à traiter.
225      * @param size La taille du filtre gaussien.
226      * @param s Le paramètre de la gaussienne.
227      * @param name Le nom du fichier dans lequel est enregistré l'image finale.
228      */
229     public static void lissageG(BufferedImage image, int size, float s, String name) {
230         int w = image.getWidth();
231         int h = image.getHeight();
232         int[][][] rgb = getRGB(image, w, h);
233         int[] pix = new int[w*h];
234         float[][] m = matGauss(size, s);
235         float[] sum = new float[]{0, 0, 0};
236         for(int x = size; x < w-size; x++) {
237             for(int y =size; y < h-size; y++) {
238                 sum[0] = sum[1] = sum[2] = 0;
239                 for(int xx = -size; xx <= size; xx++) {
240                     for(int yy = -size; yy <= size; yy++) {
241                         for(int i = 0; i < 3; i++)
242                             sum[i] += (float) rgb[i][x+xx][y+yy]*m[size+xx][size+yy];
243                     }
244                 }
245                 for(int i = 0; i < 3; i++)
246                     pix[x + y*w] += ((int) sum[i]) << ((2-i)*8);
247             }
248         }
249         image.setRGB(0, 0, w, h, pix, 0, w);
250         save(image, name);
251     }
252 
253     /**
```

```
254      * Ajoute si possible, un élément dans un tableau trié.
255      * @param ls Le tableau.
256      * @param b L'élément à ajouter.
257      */
258     public static void addSort(int[] ls, int b) {
259         int i = 0;
260         while(i < ls.length && b <=  ls[i])
261             i ++;
262         while(i < ls.length) {
263             int c = ls[i];
264             ls[i] = b;
265             b = c;
266             if(b == 0)
267                 return;
268             i ++;
269         }
270     }
271 
272     /**
273      * Effectue un lissage médian sur une image.
274      * @param image L'image à traiter.
275      * @param size La taille du filtre médian.
276      * @param name Le nom du fichier dans lequel sera enregistrée l'image.
277      */
278     public static void lissageM(BufferedImage image, int size, String name) {
279         int w = image.getWidth();
280         int h = image.getHeight();
281         int[][][] rgb = getRGB(image, w, h);
282         int[] pix = new int[w*h];
283         int d2 = ((2*size+1) * (2*size+1) + 1) / 2;
284         int[][] ls = new int[3][0];
285         for(int x = size; x < w-size; x++) {
286             for(int y = size; y < h-size; y++) {
287                 for(byte i = 0; i < 3; i++)
288                     ls[i] = new int[d2];
289                 for(int xx = -size; xx <= size; xx++)
290                     for(int yy = -size; yy <= size; yy++)
291                         for(byte i = 0; i < 3; i++)
292                             addSort(ls[i], rgb[i][x+xx][y+yy]);
293                 for(byte i = 0; i < 3; i++)
294                     pix[x + y*w] += ls[i][d2-1] << ((2-i)*8);
295             }
296         }
297         image.setRGB(0, 0, w, h, pix, 0, w);
298         save(image, name);
299     }
300 
301     /**
302      * Permet de séparer en plusieurs matrices (une pour chaques couleurs) une
    image.
303      * @param image L'image dont on veut récupérer les matrices.
304      * @param w La largeur de l'image.
305      * @param h La heuteur de l'image.
306      * @return Un tableau des trois matrices, rouge, vert puis bleu.
307      */
308     public static int[][][] getRGB(BufferedImage image, int w, int h) {
309         int[] pixels = new int[w*h];
310         image.getRGB(0, 0, w, h, pixels, 0, w);
311         int[][][] res = new int[3][w][h];
312         for(int x = 0; x < w; x++)
313             for(int y = 0; y < h; y++) {
314                 int i = pixels[x+y*w];
315                 res[0][x][y] = (i >> 16) & 0xFF;
316                 res[1][x][y] = (i >> 8) & 0xFF;
```

```
317                 res[2][x][y] = i & 0xFF;
318             }
319         return res;
320     }
321 
322     /**
323      * Sauvegarde dans un fichier une image en format png.
324      * @param image L'image à enregistrer.
325      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
326      */
327     public static void save(BufferedImage image, String name) {
328         try {
329             ImageIO.write(image, "png", new File(name));
330         } catch (IOException e) {
331             e.printStackTrace();
332         }
333     }
334 
335     /**
336      * Effectue un filtrage de Sobel pour récupérer les contours de l'image.
337      * @param image L'image à traiter.
338      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
339      */
340     public static void contour(BufferedImage image, String name) {
341         int w = image.getWidth();
342         int h = image.getHeight();
343         int[][][] rgb = getRGB(image, w, h);
344         int[] pix = new int[w*h];
345         float[][] m = new float[][]{new float[]{-0.25f, -0.5f, -0.25f}, new
    float[]{0, 0, 0}, new float[]{0.25f, 0.5f, 0.25f}};
346         float[][] m2 = new float[][]{new float[]{-0.25f, 0, 0.25f}, new
    float[]{-0.5f, 0, 0.5f}, new float[]{-0.25f, 0, 0.25f}};
347         float[] sum = new float[]{0, 0, 0}, sum2 = new float[]{0, 0, 0};
348         for(int x = 1; x < w-1; x++) {
349             for(int y = 1; y < h-1; y++) {
350                 sum[0] = sum[1] = sum[2] = sum2[0] = sum2[1] = sum2[2] = 0;
351                 for(int xx = -1; xx <= 1; xx++)
352                     for(int yy = -1; yy <= 1; yy++) {
353                         for(byte i = 0; i < 3; i++) {
354                             sum[i] += (float) rgb[i][x+xx][y+yy]*m[1+xx][1+yy];
355                             sum2[i] += (float) rgb[i][x+xx][y+yy]*m2[1+xx][1+yy];
356                         }
357                     }
358                 for(byte i = 0; i < 3; i++)
359                     pix[x + y*w] += (int) Math.min(255, Math.sqrt(sum[i]*sum[i] +
    sum2[i]*sum2[i])) << ((2-i)*8);
360             }
361         }
362         image.setRGB(0, 0, w, h, pix, 0, w);
363         save(image, name);
364     }
365 
366     /**
367      * Transforme une image en couleur en une image en noir et blanc.
368      * @param image L'image à traiter.
369      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
370      */
371     public static void blackWhite(BufferedImage image, String name) {
372         int w = image.getWidth();
373         int h = image.getHeight();
374         for(int x = 0; x < w; x++)
375             for(int y = 0; y < h; y++) {
376                 int i = image.getRGB(x, y);
377                 int j = (((i >> 16) & 0xFF) + ((i >> 8) & 0xFF) + (i & 0xFF)) / 3;
```

```
378                 image.setRGB(x, y, j + (j << 8) + (j << 16));
379             }
380         save(image, name);
381     }
382 
383     /**
384      * Permet de changer la couleur de tous les pixels qui sont ou ne sont pas dans une certaine bande de couleurs choisie en une nouvelle
385      * couleur.
386      * @param image L'image à traiter.
387      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
388      * @param r1 Limite inférieur de la composante rouge de la bande de couleur.
389      * @param r2 Limite supérieur de la composante rouge de la bande de couleur.
390      * @param g1 Limite inférieur de la composante verte de la bande de couleur.
391      * @param g2 Limite supérieur de la composante verte de la bande de couleur.
392      * @param b1 Limite inférieur de la composante bleue de la bande de couleur.
393      * @param b2 Limite supérieur de la composante bleue de la bande de couleur.
394      * @param bool Vrai si ce sont les pixels dans la bande de couleurs qui sont remplacés, faux si se sont ceux à l'extérieur de la
395      * bande qui sont remplacés.
396      * @param color La nouvelle couleur à appliquer.
397      */
398     public static void passeBande(BufferedImage image, String name, int r1, int r2, int g1, int g2, int b1, int b2, boolean bool, int color) {
399         int w = image.getWidth();
400         int h = image.getHeight();
401         for(int x = 0; x < w; x++)
402             for(int y = 0; y < h; y++) {
403                 int i = image.getRGB(x, y);
404                 int r = (i >> 16) & 0xFF, g = (i >> 8) & 0xFF, b = i & 0xFF;
405                 if((r>r2 || r<r1 || g>g2 || g<g1 || b>b2 || b<b1) != bool)
406                     image.setRGB(x, y, color);
407             }
408         save(image, name);
409     }
410 
411     /**
412      * Permet de ne récupérer que la partie où se trouve le code postal sur une image au préalablement traité.
413      * @param image L'image de l'enveloppe déjà traité.
414      * @param image2 L'image de l'enveloppe sans traitement.
415      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
416      */
417     public static void separation(BufferedImage image, BufferedImage image2, String name) {
418         int w = image.getWidth();
419         int h = image.getHeight();
420         int dist = (int)(w*0.012);
421         float wish = (float) Math.sqrt(0.00359f*w*h);
422         ArrayList<Integer[]> comp = new ArrayList<Integer[]>();
423         int[][] app = new int[w][h];
424         int itt = 0, wma = (int) (w*0.9f);
425         for(int x = w/14; x < wma; x++) {
426             for(int y = (int) (h*0.35f); y < h; y++) {
427                 int i = image.getRGB(x, y);
428                 int r = (i >> 16) & 0xFF, g = (i >> 8) & 0xFF, b = i & 0xFF;
429                 if(r != 0 || g != 0 || b != 0) {
430                     int mj = Math.min(w-1, x+dist);
431                     for(int j = Math.max(0, x-dist); j <= mj; j++) {
432                         for(int k = Math.max(0, y-dist); k <= y; k++) {
433                             if(app[j][k] != 0) {
434                                 app[x][y] = app[j][k];
435                                 Integer[] c = comp.get(app[x][y]-1);
436                                 c[0] = Math.min(c[0], x);
```

```
437                                 c[1] = Math.max(c[1], x);
438                                 c[2] = Math.min(c[2], y);
439                                 c[3] = Math.max(c[3], y);
440                                 j = w;
441                                 k = h;
442                             }
443                         }
444                     }
445                     if(app[x][y] == 0) {
446                         app[x][y] = itt + 1;
447                         comp.add(itt, new Integer[]{x, x, y, y});
448                         itt ++;
449                     }
450                 }
451             }
452         }
453         for(int i = 0; i < itt-1; i++) {
454             Integer[] c1 = comp.get(i);
455             for(int j = i+1; j < itt; j++) {
456                 Integer[] c2 = comp.get(j);
457                 if(!(c1[0] > c2[1] || c1[1] < c2[0] || c1[2] > c2[3] || c1[3] <
    c2[2])) {
458                     c2[0] = Math.min(c2[0], c1[0]);
459                     c2[1] = Math.max(c2[1], c1[1]);
460                     c2[2] = Math.min(c2[2], c1[2]);
461                     c2[3] = Math.max(c2[3], c1[3]);
462                     c1[0] = -1;
463                     break;
464                 }
465             }
466         }
467         int res = 0;
468         double max = w*h;
469         for(int i = 0; i < itt; i++) {
470             Integer[] c = comp.get(i);
471             if(c[0] == -1)
472                 continue;
473             int w1 = c[1]-c[0] + 1, h1 = c[3]-c[2] + 1;
474             double diff;
475             if(w1 < 4*wish) {
476                 if(h1 < wish)
477                     diff = 4*wish*(wish - h1) + h1*(4*wish-w1);
478                 else
479                     diff = w1*(h1-wish) + wish*(4*wish-w1);
480             } else {
481                 if(h1 < wish)
482                     diff = 4*wish*(wish - h1) + h1*(w1-4*wish);
483                 else
484                     diff = w1*(h1-wish) + wish*(w1-4*wish);
485             }
486             diff += 8.6 * (Math.abs(w1 - 4*wish) + Math.abs(h1 - wish));
487             if(diff < max) {
488                 res = i;
489                 max = diff;
490             }
491         }
492         Integer[] c = comp.get(res);
493         int w1 = c[1]-c[0] + 1, h1 = c[3]-c[2] + 1;
494         BufferedImage im = new BufferedImage(w1, h1, BufferedImage.TYPE_INT_RGB);
495         Graphics2D g2Image = im.createGraphics();
496         g2Image.drawImage(image2, 0, 0, w1, h1, c[0], c[2], c[1]+1, c[3]+1, null);
497         save(im, name);
498     }
499
```

```
500     /**
501      * Applique un seuillage à une image.
502      * @param image L'image à traiter.
503      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
504      * @param seuil Le seuil pour les 3 composantes rouge, vert et bleu.
505      * @param bool Vrai si se sont les pixels au dessus du seuil qui deviennet
   blanc, faux si se sont les pixels en dessous du seuil qui
506      * deviennent noir.
507      */
508     public static void seuil(BufferedImage image, String name, int seuil, boolean
   bool) {
509         int w = image.getWidth();
510         int h = image.getHeight();
511         for(int x = 0; x < w; x++)
512             for(int y = 0; y < h; y++) {
513                 int i = image.getRGB(x, y);
514                 int r = (i >> 16) & 0xFF, g = (i >> 8) & 0xFF, b = i & 0xFF;
515                 if(bool && r >= seuil && g >= seuil && b >= seuil)
516                     image.setRGB(x, y, 0xFFFFFF);
517                 else if(!bool && r <= seuil && g <= seuil && b <= seuil)
518                     image.setRGB(x, y, 0);
519             }
520         save(image, name);
521     }
522 
523     /**
524      * Sépare les chiffres dans le code postal après avoir effectué le traitement
   nécessaire à celui ci.
525      * @param image L'image à traiter.
526      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
527      */
528     public static void sep2(BufferedImage image, String name) {
529         int w = image.getWidth();
530         int h = image.getHeight();
531         int[] df = new int[10];
532         int[] dg = new int[]{h, 0, h, 0, h, 0, h, 0, h, 0};
533         int i = 0;
534         boolean col = false;
535         for(int x = 0; x < w; x++) {
536             boolean c = false;
537             for(int y = 0; y < h; y++) {
538                 if((image.getRGB(x, y) & 0xFF) == 0) {
539                     dg[2*(int)(i/2)] = Math.min(dg[2*(int)(i/2)], y);
540                     dg[2*(int)(i/2)+1] = Math.max(dg[2*(int)(i/2)+1], y);
541                     c = true;
542                 }
543             }
544             if(c != col) {
545                 df[i] = x;
546                 i ++;
547             }
548             col = c;
549         }
550         if(df[9] == 0)
551             df[9] = w;
552         for(int j = 0; j < 5; j++) {
553             BufferedImage im = new BufferedImage(df[2*j+1] - df[2*j] + 1, dg[2*j+1]
   - dg[2*j] + 1, BufferedImage.TYPE_INT_RGB);
554             Graphics2D g2Image = im.createGraphics();
555             g2Image.drawImage(image, 0, 0, df[2*j+1] - df[2*j] + 1, dg[2*j+1] -
   dg[2*j] + 1, df[2*j], dg[2*j], df[2*j+1], dg[2*j+1] + 1, null);
556             save(im, name+j+".png");
557         }
558     }
```

```
559
560     /**
561      * Change la dimension d'une image de chiffre en gardant les proportions
    adéquates vers les dimensions nécessaire pour le réseau de
562      * neurones.
563      * @param image L'image à traiter.
564      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
565      */
566     public static void conversion(BufferedImage image, String name, int nw, int nh,
    int d) {
567         float w = image.getWidth();
568         float h = image.getHeight();
569         float h2, w2;
570         if(w > 3*h/4) {
571             w2 = nw-d;
572             h2 = w2 / w * h;
573         } else {
574             h2 = nh-d;
575             w2 = h2 / h * w;
576         }
577         int dw = (int) ((nw - w2)/2), dh = (int) ((nh - h2)/2);
578         BufferedImage im = new BufferedImage(nw, nh, BufferedImage.TYPE_INT_RGB);
579         Graphics2D g2Image = im.createGraphics();
580         g2Image.setColor(Color.white);
581         g2Image.fillRect(0, 0, nw, nh);
582         g2Image.drawImage(image, dw, dh, nw - dw, nh - dh, 0, 0, (int)w, (int)h,
    null);
583         save(im, name);
584     }
585
586     /**
587      * Change les pixels qui sont dans une certaine zone de gris en des pixels
    noirs.
588      * @param image L'image à traiter.
589      * @param name Le nom du fichier dans lequel sera sauvegardée l'image.
590      */
591     public static void GtoB(BufferedImage image, String name) {
592         float w = image.getWidth();
593         float h = image.getHeight();
594         for(int x = 0; x < w; x++)
595             for(int y = 0; y < h; y++) {
596                 int i = image.getRGB(x, y);
597                 int r = (i >> 16) & 0xFF, g = (i >> 8) & 0xFF, b = i & 0xFF;
598                 if(g > 100 && g < 148 && Math.abs(r-g) + Math.abs(g-b) < 27 )
599                     image.setRGB(x, y, 0);
600             }
601         save(image, name);
602     }
603
604     /**
605      * Converti toutes les images du dossier Images2 au format 24x32 et les
    enregistre dans le dossier Images3.
606      */
607     public static void moveI2ToI3() {
608         BufferedReader reader;
609         try {
610             reader = new BufferedReader(new InputStreamReader(new
    FileInputStream(new File("Images2/nums.txt"))), 32768);
611             for(int n = 0; n < 10; n++) {
612                 int num = Integer.parseInt(reader.readLine());
613                 for(int i = 0; i <= num; i++)
614                     conversion(ImageIO.read(new File("Images2/"+n+""+i+".png")),
    "Images3/"+n+""+i+".png", 24, 32, 0);
615             }
```

```
616             reader.close();
617         } catch (NumberFormatException | IOException e) {
618             e.printStackTrace();
619         }
620     }
621 
622     public static void moveI2ToV() {
623         BufferedReader reader;
624         PrintWriter writer;
625         try {
626             reader = new BufferedReader(new InputStreamReader(new
    FileInputStream(new File("Images2/nums.txt"))), 32768);
627             for(int n = 0; n < 10; n++) {
628                 int num = Integer.parseInt(reader.readLine());
629                 for(int i = 0; i <= num; i++) {
630                     float[] v = allinfo(ImageIO.read(new
    File("Images2/"+n+""+i+".png")));
631                     writer = new PrintWriter(new BufferedWriter(new
    FileWriter("Var/"+n+""+i+".txt")));
632                     for(int j = 0; j < 100; j++)
633                         writer.println(v[j]);
634                     writer.close();
635                 }
636             }
637             reader.close();
638         } catch (NumberFormatException | IOException e) {
639             e.printStackTrace();
640         }
641     }
642 
643     public static float[] getVars(int n, int i) throws IOException {
644         float[] res = new float[100];
645         BufferedReader reader = new BufferedReader(new InputStreamReader(new
    FileInputStream(new File("Var/"+n+""+i+".txt"))), 32768);
646         for(int j = 0; j < 100; j++)
647             res[j] = Float.valueOf(reader.readLine());
648         reader.close();
649         return res;
650     }
651 
652     /**
653      * Permet d'obtenir les cinq petites images des chiffres de code postal sur une
    photo d'enveloppe dans le format 48x64 et 24x32.
654      * @param image L'image de l'enveloppe dont on veut récupérer le code postal.
655      * @return Un tableau des cinq images de chiffres au format 48x64 puis 24x32
656      * @throws IOException A cause des lectures de fichiers.
657      */
658     public static BufferedImage[] getDigits(BufferedImage image) throws IOException
    {
659         int w = image.getWidth();
660         if(w > 1900) {
661             conversion(image, "Traitement/a.png", 1900, (int)
    (image.getHeight()*1900f/((float) w)), 0);
662             w = 1900;
663             image = ImageIO.read(new File("Traitement/a.png"));
664         } else
665             save(image, "Traitement/a.png");
666         BufferedImage image2 = ImageIO.read(new File("Traitement/a.png"));
667         contour(image2, "Traitement/b.png");
668         image2 = ImageIO.read(new File("Traitement/b.png"));
669         passeBande(image2, "Traitement/c.png", 6, 60, 16, 160, 20, 200, false, 0);
670         image2 = ImageIO.read(new File("Traitement/c.png"));
671         lissageG(image2, 1+w/1500, 1.3f, "Traitement/d.png");
672         image2 = ImageIO.read(new File("Traitement/d.png"));
```

```
673         seuil(image2, "Traitement/e.png", 20, false);
674         if(w > 1000)
675             lissageM(image2, 1, "Traitement/e.png");
676         image2 = ImageIO.read(new File("Traitement/e.png"));
677         separation(image2, image, "Traitement/f.png");
678         image2 = ImageIO.read(new File("Traitement/f.png"));
679         seuil(image2, "Traitement/g.png", 165, true); // 160 ?
680         seuil(image2, "Traitement/g.png", 120, false);
681         image2 = ImageIO.read(new File("Traitement/g.png"));
682         GtoB(image2, "Traitement/h.png");
683         passeBande(image2, "Traitement/h.png", 130, 255, 50, 215, 0, 200, true,
    0xFFFFFF);
684         image2 = ImageIO.read(new File("Traitement/h.png"));
685         seuil(image2, "Traitement/i.png", 254, false);
686         sep2(image2, "Traitement/j");
687         for(int i = 0; i < 5; i++) {
688             image2 = ImageIO.read(new File("Traitement/j"+i+".png"));
689             conversion(image2, "Traitement/k"+i+".png", 48, 64, 2);
690             image2 = ImageIO.read(new File("Traitement/j"+i+".png"));
691             conversion(image2, "Traitement/k"+(i+5)+".png", 24, 32, 2);
692         }
693         return new BufferedImage[] {ImageIO.read(new File("Traitement/k0.png")),
    ImageIO.read(new File("Traitement/k1.png")),
694                 ImageIO.read(new File("Traitement/k2.png")), ImageIO.read(new
    File("Traitement/k3.png")),
695                 ImageIO.read(new File("Traitement/k4.png")), ImageIO.read(new
    File("Traitement/k5.png")),
696                 ImageIO.read(new File("Traitement/k6.png")), ImageIO.read(new
    File("Traitement/k7.png")),
697                 ImageIO.read(new File("Traitement/k8.png")), ImageIO.read(new
    File("Traitement/k9.png"))};
698     }
699 
700     /**
701      * Le Main pour mettre à jour les autres dossiers lorsque des images de
    chiffres ont été ajouté dans le dossier Images2.
702      * @param args Argument inutile ici.
703      */
704     public static void main(String[] args) {
705         moveI2ToI3();
706         moveI2ToV();
707     }
708 
709 }
```

```java
import java.awt.image.BufferedImage;

public class Bayes {

    /**
     * Main qui print les chiffres que le lit le réseau bayésien sur les 5 petites images de chiffres
     * situées dans le dossier prévu à cet effet.
     * @param args Argument inutile ici.
     */
    public static void main(String[] args) {
        double[][] e = new double[10][100], v = new double[10][100];
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(new FileInputStream(new File("Images2/nums.txt"))), 32768);
            for(int n = 0; n < 10; n++) {
                int num = Integer.parseInt(reader.readLine());
                for(int i = 0; i < num; i++) {
                    BufferedImage image = ImageIO.read(new File("Images2/"+n+""+i+""+".png"));
                    float[] dat = ImageUtil.allinfo(image);
                    for(int j = 0; j < 100; j++)
                        e[n][j] += dat[j];
                }
                for(int j = 0; j < 100; j++)
                    e[n][j] /= num;
                for(int i = 0; i < num; i++) {
                    BufferedImage image = ImageIO.read(new File("Images2/"+n+""+i+""+".png"));
                    float[] dat = ImageUtil.allinfo(image);
                    for(int j = 0; j < 100; j++)
                        v[n][j] += Math.pow(dat[j] - e[n][j], 2);
                }
                for(int j = 0; j < 100; j++) {
                    v[n][j] /= (num - 1);
                    if(v[n][j] == 0)
                        v[n][j] += 0.001;
                }
            }
            reader.close();
            //Test
            for(int i = 0; i < 5; i ++) {
            BufferedImage image = ImageIO.read(new File("Traitement/k"+i+".png"));
            float[] dat = ImageUtil.allinfo(image);
            int res = -1;
            double best = - 999999999;
            for(int n = 0; n < 10; n++) {
                double p = 0;
                for(int j = 0; j < 100; j++) {
                    p += logNorm(dat[j], e[n][j], v[n][j]);
                }
                if(p > best) {
                    res = n;
                    best = p;
                }
            }System.out.println(res);}
        } catch (NumberFormatException | IOException er) {
            er.printStackTrace();
        }
    }

    /**
     * Permet d'obtenir le logarithme de la valeur de la densité de probabilité d'une loi normale
```

```
63      * de variance <b>v</b> et d'espérence <b>e</b> en <b>x</b> à une constante
  près.
64      * @param x La valeur en laquelle on évalue la densité de probabilité.
65      * @param e L'espérence de la loi.
66      * @param v La variance de la loi.
67      * @return Le logarithme de le probabilité à une constante près.
68      */
69     public static double logNorm(float x, double e, double v) {
70         return -Math.pow((x - e) / v, 2)/2 - Math.log(v);
71     }
72 }
```

```java

import java.io.Serializable;

public class Axon implements Serializable{

    private static final long serialVersionUID = 49351586429210727L;
    protected Neurone a, b; //Neurone de départ et d'arrivée.
    protected float w; //Poids.

    public static float inertie = 0.4f; //Coefficient d'inertie.
    private float lastAdd = 0; //Dernier ajout de poids (utile pour l'inertie).

    /**
     * Creé un Axone entre deux neurones.
     * @param a Le neurone de départ.
     * @param b Le neurone d'arrivée.
     * @param randomRange La longeur du demi interval centré en 0 dans lequel sera choisi aléatoirement le poids.
     */
    public Axon(Neurone a, Neurone b, float randomRange) {
        this.a = a;
        this.b = b;
        this.w = (float) ((Math.random()-0.5)*2*randomRange);
    }

    /**
     * Ajoute du poids.
     * @param add La valeur de poids à ajouter.
     */
    public void addWeight(float add) {
        lastAdd = add + lastAdd*inertie;
        w += lastAdd;
    }

    public float getValueIn() { return a.getValue();}
    public Neurone getOut() { return b; }
    public float getWeight() { return w; }

}
```

```
1
2 public class AxonD extends Axon {
3
4     private static final long serialVersionUID = 6513711911621746660L;
5     private Neurone a2; //Le second neurone de départ.
6     private float tempAdd; //L'ajout temporaire de poids en attendant l'ajout du
  second neurone.
7     private boolean addW; //Pour savoir si on fait un ajout temporaire ou pas.
8
9     /**
10     * Creé un Axone double entre deux neurones de départ et un de sortie.
11     * @param a Le premier neurone de départ.
12     * @param a2 Le second neurone de départ.
13     * @param b Le neurone d'arrivée.
14     * @param randomRange La longeur du demi interval centré en 0 dans lequel sera
  choisi aléatoirement le poids.
15     */
16    public AxonD(Neurone a, Neurone a2, Neurone b, float randomRange) {
17        super(a, b, randomRange);
18        this.a2 = a2;
19        this.addW = false;
20    }
21
22    public float getValueIn() {
23        return a.getValue() + a2.getValue();
24    }
25
26    public void addWeight(float add) {
27        if(addW)
28            w += (tempAdd + add)/2;
29        else
30            tempAdd = add;
31        addW = !addW;
32    }
33
34 }
```

```java

import java.io.Serializable;

public class Neurone implements Serializable {

    private static final long serialVersionUID = 55680251679957181L;
    protected ArrayList<Axon> axIn, axOut; //Liste des axones qui arrive à ce neurone et qui partent de ce neurone.
    private float value; //La valeur actuel.
    private float delta; //L'erreur commise.

    /**
     * Crée un nouveau neurone connecté à aucun autre neurone.
     */
    public Neurone() {
        axIn = new ArrayList<Axon>();
        axOut = new ArrayList<Axon>();
    }

    protected float sig(float x) { return (float) (1 / (1 + Math.exp(-x))); } //fonction sigmoïde
    protected float sigpOfSig(float x) { return x*(1-x); } // retourne la valeur de la dérivée de la sigmoïde connaissant la valeur de la sigmoïde.

    /**
     * Ajoute un axone en entrée.
     * @param ax L'axone à ajouter en entrée.
     */
    public void addAxIn(Axon ax) {
        this.axIn.add(ax);
    }

    /**
     * Ajoute un axone en sortie.
     * @param ax L'axone à ajouter en sortie.
     */
    public void addAxOut(Axon ax) {
        this.axOut.add(ax);
    }

    /**
     * Met à jour la valeur du neurone en fonction de ses entrées.
     */
    public void updateValue() {
        value = 0;
        for(Axon a : axIn)
            value += a.getWeight() * a.getValueIn();
        value = sig(value);
    }

    /**
     * Modifie le poids des axones de sortie en fonction de l'erreur commise actuelle.
     * @param learning Le coefficient d'apprentissage.
     */
    public void retroweight(float learning) {
        for(Axon a : axOut)
            a.addWeight(learning * a.getOut().delta * value);
    }

    /**
     * Met à jour l'erreur puis modifie le poids des axones de sortie en fonction de l'erreur commise actuelle.
     * @param learning Nouveu coeeficient d'apprentissage.
```

```
61      */
62     public void retropropagate(float learning) {
63         delta = 0;
64         for(Axon a : axOut)
65             delta += a.getWeight() * a.getOut().delta;
66         delta *= sigpOfSig(value);
67         for(Axon a : axOut)
68             a.addWeight(learning * a.getOut().delta * value);
69     }
70 
71     /**
72      * Calcule l'erreur de ce neurone qui doit être un neurone de sortie en fonction
   de la valeur qu'il devrait avoir.
73      * @param rightValue La valeur que devrait avor ce neurone.
74      */
75     public void calcDelta(float rightValue) {
76         delta = (rightValue - value) * sigpOfSig(value);
77     }
78 
79     public float getValue() { return value; }
80     public void setValue(float value) { this.value = value; }
81     public float getDelta() { return delta; }
82     public ArrayList<Axon> getAxonsIn() { return axIn; }
83 
84 }
```

```java
import java.io.*;

/**
 * @author Yoann
 * Réseau de neurones fonctionnant par rétropropagation du gradient.
 */
public class Network implements Serializable{

    private static final long serialVersionUID = 4076597283090814799L;
    private ArrayList<Neurone> neurones; //Listes des neurones du réseau
    private Neurone seuilNeu; //Neurones de seuil (sa valeur est toujours de 1)
    private int[] columns; //tableau des premiers indices de chaques colones
    private int inLen, outLen, startOut; //nombre de neurones d'entrée, de sortie et indice du premier neurone de sortie
    private float learning; //coefficient d'apprentissage


    /**
     * Création d'un nouveu réseau sans aucun neurones.
     */
    public Network() {
        neurones = new ArrayList<Neurone>();
    }

    /**
     * Génère la strucutre principale du réseau.
     * @param sizes Un tableau qui contient le nombre de neurones dans chaques colones que contient le réseau.
     * @param inLen Le nombre de neurones en entrée.
     * @param outLen Le nombre de neurones en sortie.
     * @param learning Le coefficient d'apprentissage du réseau.
     */
    public void createNetwork(int[] sizes, int inLen, int outLen, float learning) {
        columns = new int[sizes.length+1];
        columns[0] = 0;
        for(int i = 0; i < sizes.length; i++) {
            columns[i+1] = columns[i] + sizes[i];
        }
        for(int i = 0; i < columns[columns.length-1]; i++)
            neurones.add(new Neurone());
        this.learning = learning;
        this.inLen = inLen;
        this.outLen = outLen;
        this.startOut = columns[columns.length-1] - outLen;
        seuilNeu = new Neurone();
        seuilNeu.setValue(1);
    }

    /**
     * Connecte tous les neurones d'une colone <b>a</b> à tous les neurones d'une colone <b>b</b>.
     * @param a L'indice de la colone dont partent les axones.
     * @param b L'indice de la colone où arrivent les axones.
     * @param rangeWeight La longeur du demi interval centré en 0 dans lequel seront choisi aléatoirement les poids.
     */
    public void connectAll(int a, int b, float rangeWeight) {
        for(int j = columns[a]; j < columns[a+1]; j++) {
            for(int k = columns[b]; k < columns[b+1]; k++) {
                Axon ax = new Axon(neurones.get(j), neurones.get(k), rangeWeight);
                neurones.get(j).addAxOut(ax);
                neurones.get(k).addAxIn(ax);
            }
        }
```

```
62      }
63
64      /**
65       * Connecte tous les neurones de plusieurs colones à tous les neurones d'une
  colone <b>b</b>.
66       * @param a Le tableau des indices des colones dont partent les axones.
67       * @param b L'indice de la colone où arrivent les axones.
68       * @param rangeWeight La longeur du demi interval centré en 0 dans lequel
  seront choisi aléatoirement les poids.
69       */
70      public void connectAll(int a[], int b, float rangeWeight) {
71          for(int a2 : a)
72              connectAll(a2, b, rangeWeight);
73      }
74
75      /**
76       * Connecte des neurones en séquences d'une colone <b>a</b> à un en esemble de
  colonnes d'arrivée.
77       * @param a L'indice de la colone dont partent les axones.
78       * @param b Le tableau des indices des colones où arrivent les axones.
79       * @param len Le nombre de neurones consécutifs de la colone <b>a</b> qui sont
  connectés à un neurone d'une des colones d'arrivée.
80       * @param inc Le pas d'avancement du premier neurone de la colonne <b>a</b>
  connecté à un neurone d'une des colones d'arrivée.
81       * @param rangeWeight La longeur du demi interval centré en 0 dans lequel
  seront choisi aléatoirement les poids.
82       */
83      public void connectSeq(int a, int[] b, int len, int inc, float rangeWeight) {
84          int a1 = columns[a];
85          for(int b2 : b) {
86              int b3 = columns[b2];
87              for(int j = b3; j < columns[b2+1]; j++) {
88                  for(int i = a1 + (j-b3)*inc; i < a1 + (j-b3)*inc + len; i++) {
89                      Axon ax = new Axon(neurones.get(i), neurones.get(j),
  rangeWeight);
90                      neurones.get(i).addAxOut(ax);
91                      neurones.get(j).addAxIn(ax);
92                  }
93              }
94          }
95      }
96
97      /**
98       * Connecte des colones à d'autres par l'intermédiaire d'axiones doubles, ainsi
  la taille des colones d'entrée doivent être deux fois
99       * plus grande que celle des colones de sortie.
100      * @param a a Le tableau des indices des colones dont partent les axones.
101      * @param b Le tableau des indices des colones où arrivent les axones.
102      * @param rangeWeight La longeur du demi interval centré en 0 dans lequel
  seront choisi aléatoirement les poids.
103      */
104     public void connectDou(int a[], int[] b, float rangeWeight) {
105         for(int i = 0; i < a.length; i++) {
106             int a2 = columns[a[i]], b2 = columns[b[i]];
107             for(int j = 0; j < columns[b[i]+1]-b2; j++) {
108                 Axon ax = new AxonD(neurones.get(a2+2*j), neurones.get(a2+2*j+1),
  neurones.get(b2+j), rangeWeight);
109                 neurones.get(a2+2*j).addAxOut(ax);
110                 neurones.get(a2+2*j+1).addAxOut(ax);
111                 neurones.get(b2+j).addAxIn(ax);
112             }
113         }
114     }
115
```

```
116     /**
117      * Ajoute un seuil aux neurones de plusieurs colones.
118      * @param c Le tableau des indices des colones auquelles on ajoute un seuil.
119      * @param rangeWeigth La longeur du demi interval centré en 0 dans lequel
    seront choisi aléatoirement les poids.
120      */
121     public void addSeuil(int[] c, float rangeWeigth) {
122         for(int i : c) {
123             for(int j = columns[i]; j < columns[i+1]; j++) {
124                 Axon a = new Axon(seuilNeu, neurones.get(j), rangeWeigth);
125                 seuilNeu.addAxOut(a);
126                 neurones.get(j).addAxIn(a);
127             }
128         }
129     }
130 
131     /**
132      * Permet de tester un échantillon sans faire apprendre le réseau.
133      * @param in Les données d'entrée sous forme d'un tableau.
134      * @return Un tableau des valeurs des neurones de sortie.
135      */
136     public float[] test(float[] in) {
137         if(in.length != inLen) {
138             System.err.println("Le nombre d'entrées est incorrecte !!");
139             System.err.println("in : " + inLen + "   votre valeur in : " +
    in.length);
140             System.exit(0);
141         }
142         float[] res = new float[outLen];
143         for(int i = 0; i < inLen; i++)
144             neurones.get(i).setValue(in[i]);
145         for(int i = inLen; i < columns[columns.length-1]; i++)
146             neurones.get(i).updateValue();
147         for(int i = 0; i < outLen; i++)
148             res[i] = neurones.get(startOut + i).getValue();
149         return res;
150     }
151 
152     /**
153      * Permet de faire apprendre le réseau via un échantillon dont on connait la
    réponse.
154      * @param in Les données d'entrée sous forme d'un tableau.
155      * @param out Un tableau des valeurs de sorties attendues.
156      * @return Un tableau des valeurs des neurones de sortie.
157      */
158     public float[] learn(float[] in, float[] out) {
159         if(in.length != inLen || out.length != outLen) {
160             System.err.println("Le nombre d'entrées ou de sorties est incorrecte
    !!");
161             System.err.println("in : " + inLen + " out : " + outLen + "   vos
    valeurs in : " + in.length + " out : " + out.length);
162             System.exit(0);
163         }
164         float[] res = new float[outLen];
165         for(int i = 0; i < inLen; i++)
166             neurones.get(i).setValue(in[i]);
167         for(int i = inLen; i < columns[columns.length-1]; i++)
168             neurones.get(i).updateValue();
169         for(int i = 0; i < outLen; i++) {
170             neurones.get(startOut + i).calcDelta(out[i]);
171             res[i] = neurones.get(startOut + i).getValue();
172         }
173         for(int i = startOut - 1; i >= inLen; i--)
174             neurones.get(i).retropropagate(learning);
```

```
175         for(int i = inLen - 1; i >= 0; i--)
176             neurones.get(i).retroweight(learning);
177         seuilNeu.retroweight(learning);
178         return res;
179     }
180 
181     /**
182      * @return La liste de tous les poids.
183      */
184     public ArrayList<Float> getWeights() {
185         ArrayList<Float> res = new ArrayList<Float>();
186         for(Neurone neurone : neurones)
187             for(Axon axon : neurone.getAxonsIn())
188                 res.add(axon.getWeight());
189         return res;
190     }
191 
192     /**
193      * Sauvegarde le réseau dans un fichier.
194      * @param fileName Le nom du fichier.
195      * @throws IOException En cas de problème avec le fichier.
196      */
197     public void save(String fileName) throws IOException {
198         ObjectOutputStream oos = new ObjectOutputStream(new
    FileOutputStream(fileName));
199         oos.writeObject(this);
200         oos.close();
201     }
202 
203     public void setLearning(float learn) { this.learning = learn; }
204     public float getLearning() { return this.learning; }
205     public int getInLen() { return this.inLen; }
206     public int getOutLen() { return this.outLen; }
207 
208 }
```

```java
import java.io.IOException;

/**
 * Le réseau spécifique à la reconnaissance de chiffre.
 * @author Yoann
 */
public class Reseau {

    private Network network; //Le réseau.

    /**
     * Crée un nouveau réseau adapté à la reconnaissance de chiffre.
     */
    public Reseau() {
        network = new Network();
        network.createNetwork(new int[]{16, 32, 32, 20, 6, 6, 6, 14, 14, 14, 14, 14, 3, 3, 3, 7, 7, 7, 7, 7, 15, 10}, 100, 10, 0.04f);
        int[] pfo2 = new int[]{4, 5, 6}, pfdg2 = new int[]{7, 8, 9, 10, 11};
        network.connectSeq(0, pfo2, 6, 2, 0.52f);
        network.addSeuil(pfo2, 0);
        network.connectSeq(1, pfdg2, 6, 2, 0.36f);
        network.connectSeq(2, pfdg2, 6, 2, 0.36f);
        network.addSeuil(pfdg2, 0);
        int[] pfo3 = new int[]{12, 13, 14}, pfdg3 = new int[]{15, 16, 17, 18, 19};
        network.connectDou(pfo2, pfo3, 0.81f);
        network.connectDou(pfdg2, pfdg3, 0.58f);
        network.connectAll(pfo3, 20, 0.51f);
        network.connectAll(pfdg3, 20, 0.52f);
        network.connectAll(3, 20, 0.41f);
        network.connectAll(20, 21, 0.49f);
        network.addSeuil(new int[]{20,  21}, 0);
    }

    /**
     * Remplace le réseau par un autre.
     * @param net Le nouvea rééseau.
     */
    public Reseau(Network net) {
        this.network = net;
    }

    public void setLearning(float learn) { network.setLearning(learn); }
    public float getLearning() { return network.getLearning(); }

    /**
     * Entrainement du réseau.
     * @param in L'entrée.
     * @param ans Le chiffre qui doit être reconnu.
     * @return Un tableau contenant, la réponse la plus probable du réseau, le pourcentage associé à cette réponse, le pourcentage associé
     * à la bonne réponse, et l'erreur totale quadratique commise.
     */
    public float[] train(float[] in, int ans) {
        float[] out = new float[10];
        out[ans] = 1;
        float[] rep = network.learn(in, out);
        float err = 0, per = 0;
        int num = 0;
        for(int i = 0; i < 10; i++) {
            err += (rep[i] - out[i]) * (rep[i] - out[i]);
            if(rep[i] > per) {
                per = rep[i];
                num = i;
            }
```

```
63          }
64          return new float[]{num, per*100, rep[ans]*100, err};
65      }
66
67      /** Teste du réseau sur un échantillon.
68       * @param in L'entrée.
69       * @return @see {@link Network#test(float[])}
70       */
71      public float[] train2(float[] in) {
72          return network.test(in);
73      }
74
75      /**
76       * Teste du réseau sur un échantillon numéro 2.
77       * @param in L'entrée.
78       * @return Un tableau contenant le chiffre et le pourcentage associé à ce
   dernier pour les trois chiffres les plus probables daprès
79       * le réseau.
80       */
81      public float[] train3(float[] in) {
82          float[] out = network.test(in);
83          return train3Main(out);
84      }
85
86      /**
87       * Corps tu teste numéro 2.
88       * @param out La sortie du réseau qui doit être traité.
89       * @return Un tableau contenant le chiffre et le pourcentage associé à ce
   dernier pour les trois chiffres les plus probables daprès
90       * le réseau.
91       */
92      public static float[] train3Main(float[] out) {
93          float[] res = new float[6];
94          for(int i = 0; i < 3; i++) {
95              int num = 0;
96              float per = 0;
97              for(int j = 0; j < 10; j++) {
98                  if(out[j] > per) {
99                      per = out[j];
100                     num = j;
101                 }
102             }
103             out[num] = -1;
104             res[2*i] = num;
105             res[2*i+1] = per*100;
106         }
107         return res;
108     }
109
110     /**
111      * Entrainement du réseau numéro 2.
112      * @param in L'entrée.
113      * @param ans Le chiffre qui doit être reconnu.
114      */
115     public void train4(float[] in, int ans) {
116         float[] out = new float[10];
117         out[ans] = 1;
118         network.learn(in, out);
119     }
120
121     /**
122      * @see Network#save(String)
123      * @param fileName Le nom du fichier dans lequel est enregistré le réseau.
124      * @throws IOException
```

```
125      */
126     public void save(String fileName) throws IOException {
127         network.save(fileName);
128     }
129 
130 }
```

```
 1
 2 import java.awt.*;
13
14 /**
15  * Zone de dessein intégré dans le logiciel.
16  * @author Yoann
17  */
18 public class Board extends JPanel{
19
20     private static final long serialVersionUID = 2195379760914381351L;
21     private int x, y; //Dernière position de la souris.
22     private BasicStroke stroke = new BasicStroke(8), stroke2 = new BasicStroke(4);
   //Les tailles de traits.
23     private BufferedImage image = new BufferedImage(48, 64,
   BufferedImage.TYPE_INT_RGB); //L'image finale
24     private Graphics2D g2Image = image.createGraphics(); //le graphique sur lequel
   on dessine les traits pour l'image finale.
25     private JPanel panelImage; // Le paneau dans lequel on affiche l'image finale.
26     private ArrayList<Line> lines = new ArrayList<Line>(); //La listes des lignes à
   dessiner.
27     private int minX = 8000, maxX = -8000, minY = 8000, maxY = -8000; //Les
   coordonées du rectangle dans lequel se trouve le dessein.
28
29     /**
30      * Crée une nouvelle zone de dessein.
31      */
32     public Board() {
33         this.addMouseMotionListener(new MouseMotionListener() {
34             public void mouseDragged(MouseEvent e) {
35                 int x1 = x, y1 = y;
36                 x = e.getX(); y = e.getY();
37                 Graphics2D g2 = (Graphics2D) getGraphics();
38                 g2.setStroke(stroke);
39                 g2.draw(new Line2D.Float(x1, y1, x, y));
40                 lines.add(new Line(x1, y1, x, y));
41                 minX = Math.min(minX, Math.min(x1, x));
42                 maxX = Math.max(maxX, Math.max(x1, x));
43                 minY = Math.min(minY, Math.min(y1, y));
44                 maxY = Math.max(maxY, Math.max(y1, y));
45                 int w = maxX - minX + 1, h = maxY - minY + 1;
46                 if(w > 3*h/4) h = 4 *w/3;
47                 else w = 3*h/4;
48                 x1 = minX + (maxX- minX - w)/2;
49                 y1 = minY + (maxY- minY - h)/2;
50                 w = w*54/48;
51                 h = h*70/64;
52                 g2Image.setColor(Color.white);
53                 g2Image.fillRect(0, 0, 48, 64);
54                 g2Image.setColor(Color.BLACK);
55                 g2Image.setStroke(stroke2);
56                 for(Line line : lines)
57                     g2Image.draw(new Line2D.Float((line.x-x1)*48/w+3,
   (line.y-y1)*64/h+3, (line.x2-x1)*48/w+3, (line.y2-y1)*64/h+3));
58                 panelImage.repaint();
59             }
60             public void mouseMoved(MouseEvent e) {
61                 x = e.getX(); y = e.getY();
62             }
63         });
64     }
65
66     public void paintComponent(Graphics g) {
67         g.setColor(Color.white);
68         g.fillRect(0, 0, getWidth(), getHeight());
```

```
69          g.setColor(Color.BLACK);
70          g2Image.setColor(Color.white);
71          g2Image.fillRect(0, 0, 48, 64);
72          g2Image.setColor(Color.BLACK);
73          if(panelImage != null) panelImage.repaint();
74      }
75
76      public BufferedImage getImage() { return image; }
77      public void setPanelImage(JPanel pan) { panelImage = pan; }
78
79      /**
80       * Efface la zone de dessein.
81       */
82      public void erase() {
83          repaint();
84          minX = 8000;
85          maxX = -8000;
86          minY = 8000;
87          maxY = -8000;
88          lines.clear();
89      }
90
91      class Line { //Pour les lignes que l'on dessine.
92          int x, y, x2, y2;
93          public Line(int x, int y, int x2, int y2) {
94              this.x = x;
95              this.y = y;
96              this.x2 = x2;
97              this.y2 = y2;
98          }
99      }
100
101 }
102
```

```java
import java.awt.*;

/**
 * L'interface pour l'apprentissage des réseaux et la création d'image pour la base de données.
 * @author Yoann
 */
public class Interface extends JFrame {

    private static final long serialVersionUID = 3120796899747904131L;
    private JMenuBar jmb = new JMenuBar(); //La barre de menu.
    private JMenu files = new JMenu("Fichier"); //Le menu Fichier
    private JMenuItem erase = new JMenuItem("Effacer"), saveN = new JMenuItem("Sauver réseau"), loadN = new JMenuItem("Charger réseau"),
            quit = new JMenuItem("Quitter"); //Les boutons du menu Ficher.
    private JButton save = new JButton("save img"),
            era = new JButton("effacer"), autob = new JButton("auto"), detail = new JButton("détails"),
            pract = new JButton("temps réel"), eff = new JButton("test efficacité"); //Les boutons du centre de la fenêtre.
    private Board board = new Board(); //Le panneau de dessein.
    private JPanel imagePanel, centerPanel = new JPanel(); //Les panneaux pour les boutons et pour l'affichage de l'image finale.
    private BufferedImage image = board.getImage(); //L'image finale.
    private JLabel resLabel = new JLabel(" "); //La zone de texte du bas.
    private JProgressBar loading = new JProgressBar(); //La barre de progression.
    private Reseau net; // Le réseau.
    private boolean auto; //Pour savoir si on est en mode automatique.
    private Thread th; //Le Thread pour l'apprentissage automatique et le temps réel.
    private String fileName; //Le nom du fichier en cours de traitement.
    private boolean loop = false; //Pour savoir si on présente les données en boucle ou aléatoirement.

    /**
     * Crée une nouvelle interface à afficher.
     */
    public Interface() {
        net = new Reseau();
        auto = false;
        setSize(750, 420);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Reconnaissance de chiffres");
        erase.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                board.erase();
            }
        });
        erase.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, KeyEvent.CTRL_DOWN_MASK));
        saveN.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if(fileName == null)
                    fileName = JOptionPane.showInputDialog(null, "Sous quel nom voulez-vous enregistrer le fichier ?");
                try {
                    net.save("Networks/"+fileName+".net");
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        });
        saveN.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
```

```
KeyEvent.CTRL_DOWN_MASK));
        loadN.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JFileChooser fc = new JFileChooser();
                fc.setCurrentDirectory(new File("Networks"));
                fc.showOpenDialog(null);
                File file = fc.getSelectedFile();
                if(file == null)
                    return;
                try {
                    fileName = file.getName().replace(".net", "");
                    ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(file));
                    net = new Reseau((Network) ois.readObject());
                    ois.close();
                } catch (IOException | ClassNotFoundException e1) {
                    e1.printStackTrace();
                }
            }
        });
        loadN.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
KeyEvent.CTRL_DOWN_MASK));
        quit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        files.add(erase);
        files.addSeparator();
        files.add(saveN);
        files.add(loadN);
        files.add(quit);
        jmb.add(files);
        setJMenuBar(jmb);
        imagePanel = new JPanel() {
            private static final long serialVersionUID = -3757950555935332081L;
            public void paintComponent(Graphics g) {
                g.setColor(Color.white);
                g.fillRect(0, 0, getWidth(), getHeight());
                g.drawImage(image, 0, 0, this.getWidth(), this.getHeight(), null);
            }
        };
        board.setPanelImage(imagePanel);
        board.setPreferredSize(new Dimension(310, 400));
        getContentPane().add(board, BorderLayout.WEST);
        imagePanel.setPreferredSize(new Dimension(310, 400));
        getContentPane().add(imagePanel, BorderLayout.EAST);
        era.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                erase.doClick();
            }
        });
        save.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int n = Integer.parseInt(JOptionPane.showInputDialog(null, "Quelle
est le chiffre que vous avez dessiné ?"));
                save(n);
            }
        });
        detail.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                float[] in = ImageUtil.allinfo(image);
                float[] data = net.train2(in);
                String s = "";
```

```
122                 for(int i = 0; i < 10; i++)
123                     s += i + ": " + data[i] + "    ";
124                 JOptionPane.showMessageDialog(null, s, "Détails",
    JOptionPane.CANCEL_OPTION);
125             }
126         });
127         autob.addActionListener(new ActionListener() {
128             public void actionPerformed(ActionEvent e) {
129                 auto = !auto;
130                 if(auto) {
131                     int n = Integer.parseInt(JOptionPane.showInputDialog("Combien
    d'ittérations voulez-vous faire ?"));
132                     board.erase();
133                     autob.setText("Stop");
134                     enableB(false);
135                     th = new Thread(new Auto(n));
136                     th.start();
137                 }
138             }
139         });
140         loading.setMinimum(0);
141         loading.setPreferredSize(new Dimension(110, 15));
142         pract.addActionListener(new ActionListener() {
143             public void actionPerformed(ActionEvent e) {
144                 auto = !auto;
145                 if(auto) {
146                     pract.setText("Stop");
147                     autob.setEnabled(false);
148                     eff.setEnabled(false);
149                     th = new Thread(new RealTime());
150                     th.start();
151                 }
152             }
153         });
154         eff.addActionListener(new ActionListener() {
155             public void actionPerformed(ActionEvent arg0) {
156                 try {
157                     BufferedReader reader = new BufferedReader(new
    InputStreamReader(new FileInputStream(new File("Images2/nums.txt"))), 32768);
158                     float per = 0, err = 0, tot = 0;
159                     for(int n = 0; n < 10; n++) {
160                         int num = Integer.parseInt(reader.readLine());
161                         tot += num + 1;
162                         for(int i = 0; i <= num; i++) {
163                             float[] data = net.train(ImageUtil.getVars(n, i), n);
164                             if(data[0] == n)
165                                 per += 1;
166                             else
167                                 System.out.println(n+""+i);
168                             err += data[3];
169                         }
170                     }
171                     reader.close();
172                     resLabel.setText("éfficacité: " + (per*100/tot) + "%
    erreur quadratique: " + err);
173                 } catch (IOException e1) {
174                     e1.printStackTrace();
175                 }
176                 image = board.getImage();
177             }
178         });
179         centerPanel.add(era);
180         centerPanel.add(save);
181         centerPanel.add(detail);
```

```
182         centerPanel.add(autob);
183         centerPanel.add(loading);
184         centerPanel.add(pract);
185         centerPanel.add(eff);
186         getContentPane().add(centerPanel, BorderLayout.CENTER);
187         getContentPane().add(resLabel, BorderLayout.SOUTH);
188         setVisible(true);
189     }
190 
191     /**
192      * Enregistre l'image actuellement dessinée.
193      * @param n la valuer du chiffre sur l'image actuellement dessinée.
194      */
195     private void save(int n) {
196         try {
197             BufferedReader reader = new BufferedReader(new InputStreamReader(new
    FileInputStream(new File("Images2/nums.txt"))), 32768);
198             int[] nums = new int[10];
199             for(int i = 0; i < 10; i++)
200                 nums[i] = Integer.parseInt(reader.readLine());
201             nums[n] += 1;
202             reader.close();
203             ImageIO.write(board.getImage(), "png", new
    File("Images2/"+n+""+nums[n]+".png"));
204             PrintWriter writer = new PrintWriter(new BufferedWriter(new
    FileWriter("Images2/nums.txt")));
205             for(int i : nums)
206                 writer.print(i + "\n");
207             writer.close();
208         } catch (IOException e1) {
209             e1.printStackTrace();
210         }
211     }
212 
213     /**
214      * Permet d'activer ou de désactiver certains boutons.
215      * @param b true pour activer les boutons, false sinon.
216      */
217     private void enableB(boolean b) {
218         save.setEnabled(b);
219         erase.setEnabled(b);
220         era.setEnabled(b);
221         detail.setEnabled(b);
222         eff.setEnabled(b);
223     }
224 
225     /**
226      * @author Yoann
227      * Classe pour l'apprentissage automatique en arrière plan.
228      */
229     class Auto implements Runnable {
230         int itt;
231         public Auto(int itt) {
232             this.itt = itt;
233             loading.setMaximum(itt);
234             pract.setEnabled(false);
235         }
236         public void run() {
237             try {
238                 BufferedReader reader = new BufferedReader(new
    InputStreamReader(new FileInputStream(new File("Images2/nums.txt"))), 32768);
239                 int[] nums = new int[10];
240                 for(int i = 0; i < 10; i++)
241                     nums[i] = Integer.parseInt(reader.readLine());
```

```
242                 reader.close();
243                 int it = 0;
244                 while(auto && it <= itt) {
245                     it ++;
246                     loading.setValue(it);
247                     int n = loop ? (it % 10) : ((int) (Math.random()*10));
248                     net.train4(ImageUtil.getVars(n, (int) (Math.random()*
    (nums[n]+1))), n);
249                 }
250             } catch (IOException e1) {
251                 e1.printStackTrace();
252             }
253             auto = false;
254             autob.setText("auto");
255             image = board.getImage();
256             enableB(true);
257             autob.setEnabled(true);
258             pract.setEnabled(true);
259             loading.setValue(0);
260         }
261     }
262 
263     /**
264      * @author Yoann
265      *  Classe permettant d'afficher en temps réel les résultats du réseau de
    neurones.
266      */
267     class RealTime implements Runnable {
268         public void run() {
269             while(auto) {
270                 float[] in = ImageUtil.allinfo(image);
271                 float[] data = net.train3(in);
272                 String s = "";
273                 for(int i = 0; i < 3; i++)
274                     s += (int)data[2*i] + " (" + data[2*i+1] + "%)           ";
275                 resLabel.setText(s);
276                 try {
277                     Thread.sleep(100);
278                 } catch (InterruptedException e) {
279                     e.printStackTrace();
280                 }
281             }
282             auto = false;
283             pract.setText("temps réel");
284             autob.setEnabled(true);
285             eff.setEnabled(true);
286         }
287     }
288 
289     /**
290      * Le Main !
291      * @param arg Argument inutile ici.
292      */
293     public static void main(String[] arg) {
294         new Interface();
295     }
296 
297 }
```

```java
import java.io.*;

/**
 * Restricted Boltzmann Machine.
 * @author Yoann
 */
public class RBM implements Serializable {

    private static final long serialVersionUID = 6588404150235761139L;
    private int n, m; //Tailles du vecteur d'entrée et du vecteur de sortie.
    private float[] b, c, addB, addC; //Les biais des vecteurs de sortie (b) et d'entrée (c). Ainsi que leurs ajouts en mémoire.
    private float[] hp; //La sorite gardée en mémoire pour l'échantillonage de Gibbs.
    private float[][] w, addW; //Matrice des poids à m lignes et n colonnes ainsi que des derniers ajouts de poids.
    public float learning, wd, inertie; //Taux d'apprentissage, de weight decay et d'inertie.
    public byte kcd; //Nombre d'ittérations dans l'échantillonage de Gibbs.
    public byte batchSize, batchIt; //Nombre d'exemples à présenter dans un batch, et l'ittération du batch actuel.

    /**
     * Crée une machine de Boltzmann restreinte avec n entrée et m sorties. Les poids sont initialisés de manière optimal d'après
     * le rapport de stage de Quentin Fresnel et les vidéos de Hugo Larochelle.
     * @param n Le nombre d'entrées.
     * @param m Le nombre de sorties.
     */
    public RBM(int n, int m) {
        this.n = n;
        this.m = m;
        c = new float[n];
        b = new float[m];
        addC = new float[n];
        addB = new float[m];
        hp = new float[m];
        float r = (float) (4 / Math.sqrt(n*m));
        w = new float[m][n];
        addW = new float[m][n];
        for(int j = 0; j < m; j++)
            for(int k = 0; k < n; k++)
                w[j][k] = (float) ((Math.random()-1) * r);
        learning = 0.0006f;
        wd = 0.00004f;
        inertie = 0.6f;
        kcd = 4;
        batchSize = 18;
        batchIt = 0;
    }

    /**
     * Calcule la sortie de la machine de Boltzmann restreinte.
     * @param x L'entrée de taille n dont on calcule la sortie.
     * @return La sortie associée à l'entrée <b>x</b>.
     */
    public float[] calc(float[] x) {
        float[] h = new float[m];
        for(int j = 0; j < m; j++) {
            float temp = b[j];
            for(int k = 0; k < n; k++)
                temp += w[j][k]*x[k];
            h[j] = sig(temp);
        }
```

```
61          return h;
62      }
63
64      /**
65       * Calcule l'entrée de la machine de Boltzmann restreinte en fonction d'une
  sortie.
66       * @param h La sortie de taille m dont on calcule l'entrée.
67       * @return L'entrée associée à la sortie <b>h</b>.
68       */
69      public float[] inv(float[] h) {
70          float[] x = new float[n];
71          for(int k = 0; k < n; k++) {
72              float temp = c[k];
73              for(int j = 0; j < m; j++)
74                  temp += w[j][k]*h[j];
75              x[k] = sig(temp);
76          }
77          return x;
78      }
79
80      /**
81       * Pour une certaine entrée <b>x</b>, calcule sa sortie, puis à partir de cette
  cernière calcule l'entrée associée.
82       * @param x L'entrée à présenter.
83       * @return Le résultat après avoir calculer la sortie puis l'entrée en partant
  de <b>x</b>.
84       */
85      public float[] calcAndInv(float[] x) {
86          return inv(calc(x));
87      }
88
89      /**
90       * Réalise une ittération d'apprentissage avec un exemple <b>x</b>.
91       * @param x Vecteur d'apprentissage de taille n.
92       */
93      public void learn(float[] x) {
94          batchIt ++;
95          float[] h = calc(x), xp = new float[n];
96          if(batchIt == 1)
97              for(int j = 0; j < m; j++)
98                  hp[j] = h[j];
99          for(byte i = 0; i < kcd; i++) {
100             xp = inv(hp);
101             hp = calc(xp);
102         }
103         for(int j = 0; j < m; j++) {
104             addB[j] += learning*(h[j] - hp[j]);
105             for(int k = 0; k < n; k++)
106                 addW[j][k] += learning*(h[j]*x[k] - hp[j]*xp[k]);
107         }
108         for(int k = 0; k < n; k++)
109             addC[k] += learning*(x[k] - xp[k]);
110         if(batchIt == batchSize) {
111             for(int j = 0; j < m; j++) {
112                 b[j] += addB[j];
113                 addB[j] = inertie*addB[j] - wd*b[j];
114                 for(int k = 0; k < n; k++) {
115                     w[j][k] += addW[j][k];
116                     addW[j][k] = inertie*addW[j][k] - wd*w[j][k];
117                 }
118             }
119             for(int k = 0; k < n; k++) {
120                 c[k] += addC[k];
121                 addC[k] = inertie*addC[k] - wd*c[k];
```

```
122             }
123             batchIt = 0;
124         }
125     }
126 
127     /**
128      * Génère une nouvelle entrée.
129      * @param len Le nombre d'ittérations d'échantillonage de Gibbs à réaliser.
130      * @param w La largeur de la représentation de l'entrée.
131      * @param h La hauteur de la représentation de l'entrée.
132      * @param firstX L'entrée présentée initialement dans l'échantillonage de
    Gibbs.
133      * @throws IOException
134      */
135     public void generate(int len, int w, int h, float[] firstX) throws IOException
    {
136         ImageUtil.save(ImageUtil.ImageFromPixels(firstX, w, h), "test.png");
137         for(int i = 0; i < len; i++) {
138             firstX = calcAndInv(firstX);
139             ImageUtil.save(ImageUtil.ImageFromPixels(firstX, w, h),
    "test"+i+".png");
140         }
141     }
142 
143     /**
144      * Permet de tester si la machine de Boltzmann restreinte est bien entrainer en
    comparant des image de chiffres
145      * aux images obtenues après un échantillonage de Gibbs de longeur <b>len</b>
    en partant des image initiales.
146      * @param len La longueur de l'échantillonage de Gibbs à effectuer.
147      * @param xs Un tableau à deux dimensions où <b>xs</b>[<b>i</b>][<b>j</b>] est
    le <b>j</b>ième pixel de la <b>i</b>ième image.
148      * @param w La largeur des images.
149      * @param h La hauteur des images.
150      */
151     public void test(int len, float[][] xs, int w, int h) {
152         for(int a = 0; a < xs.length; a++) {
153             ImageUtil.save(ImageUtil.ImageFromPixels(xs[a], w, h), a+"init.png");
154             for(int i = 0; i < len; i++)
155                 xs[a] = calcAndInv(xs[a]);
156             ImageUtil.save(ImageUtil.ImageFromPixels(xs[a], w, h), a+"rep.png");
157         }
158     }
159 
160     private float sig(float x) { return (float) (1 / (1 + Math.exp(-x))); }
    //fonction sigmoïde
161 
162     /**
163      * Sauvegarde la machine de Boltzmann restreinte dans un fichier.
164      * @param fileName Le nom du fichier.
165      * @throws IOException En cas de problème avec le fichier.
166      */
167     public void save(String fileName) throws IOException {
168         ObjectOutputStream oos = new ObjectOutputStream(new
    FileOutputStream(fileName));
169         oos.writeObject(this);
170         oos.close();
171     }
172 
173     /**
174      * Donne les représentations des impacts des composantes de l'entrée sur la
    sortie.
175      * @param x Un tableau de taille n, contenant des tableaux d'impacts d'un
    vecteur encore antérieur sur chaque composante de l'entrée.
```

```
176      * @param size La taille du vecteur dont on cherche l'impact sur cette RBM.
177      * @return Un tableau de taille m, contenant pour chaque composante de la
    sortie de cette RBM, un tableau représentant les
178      * impacts des composantes du vecteur antérieur.
179      */
180     public float[][] getRepresentations(float[][] x, int size) {
181         float[][] res = new float[m][size];
182         for(int k = 0; k < n; k++)
183             for(int j = 0; j < m; j++)
184                 for(int a = 0; a < size; a++)
185                     res[j][a] += x[k][a] * this.w[j][k];
186         return res;
187     }
188 
189     public float[][] getWeights() { return w; }
190     public int getInLen() { return n; }
191     public int getOutLen() { return m; }
192     public void setKCD(byte k) { kcd = k; }
193 
194     /**
195      * Dernier Main utilisé.
196      * Le code de ce main a souvent changé, il m'a servi à faire des tests et des
    apprentissages.
197      * @param args Argument inutile ici.
198      */
199     public static void main(String[] args) {
200         try {
201             ObjectInputStream ois = new ObjectInputStream(new
    FileInputStream("Networks/DBN2.dbn"));
202             DBN dbn = (DBN) ois.readObject();
203             ois.close();
204             int j = (int) (Math.random()*100);
205             System.out.println(j);
206             dbn.testError(80, new float[][]{ImageUtil.pixels(ImageIO.read(new
    File("Images3/0"+j+".png"))),
207                     ImageUtil.pixels(ImageIO.read(new File("Images3/1"+j+".png"))),
208                     ImageUtil.pixels(ImageIO.read(new File("Images3/2"+j+".png"))),
209                     ImageUtil.pixels(ImageIO.read(new File("Images3/3"+j+".png"))),
210                     ImageUtil.pixels(ImageIO.read(new File("Images3/4"+j+".png"))),
211                     ImageUtil.pixels(ImageIO.read(new File("Images3/5"+j+".png"))),
212                     ImageUtil.pixels(ImageIO.read(new File("Images3/6"+j+".png"))),
213                     ImageUtil.pixels(ImageIO.read(new File("Images3/7"+j+".png"))),
214                     ImageUtil.pixels(ImageIO.read(new File("Images3/8"+j+".png"))),
215                     ImageUtil.pixels(ImageIO.read(new File("Images3/9"+j+".png"))),
216             }, 24, 32, 0);
217             dbn.getRBM(0).generate(100, 24, 32, ImageUtil.pixels(ImageIO.read(new
    File("Images3/510.png"))));
218             float[][] pix = dbn.getRepresentations(0);
219             for(int i = 0; i < pix.length; i++)
220                 ImageUtil.save(ImageUtil.ImageFromPixelsB(pix[i], 24, 32),
    "Rep/rep"+i+".png");
221         } catch (IOException e) {
222             e.printStackTrace();
223         } catch (ClassNotFoundException e) {
224             e.printStackTrace();
225         }
226     }
227 
228 }
229 
```

```java
import java.io.*;

/**
 * Deep Belief Network.
 * @author Yoann
 */
public class DBN implements Serializable {

    private static final long serialVersionUID = 1791654468121703150L;
    private RBM[] rbms; //Tableau des RBMs.
    private Network network; //Le réseau final.
    private RBM inv; //La RBM qui permet l'inversion du réseau.

    /**
     * Crée un nouveau Deep Belief Network
     * @param rbmSizes Tableau des tailles des vecteurs d'entrée des RBMs.
     * @param netSizes Tableau des tailles des couches du perceptron.
     */
    public DBN(int[] rbmSizes, int[] netSizes) {
        int len = rbmSizes.length;
        rbms = new RBM[len];
        for(int i = 0; i < len-1; i++)
            rbms[i] = new RBM(rbmSizes[i], rbmSizes[i+1]);
        rbms[len-1] = new RBM(rbmSizes[len-1], netSizes[0]);
        network = new Network();
        network.createNetwork(netSizes, netSizes[0], netSizes[netSizes.length-1], 0.01f);
        for(int i = 0; i < netSizes.length-1; i++) {
            network.connectAll(i, i+1, (float) Math.sqrt(6/(netSizes[i]+netSizes[i+1])));
            network.addSeuil(new int[]{i+1}, 0);
        }
    }

    /**
     * Apprentissage d'un exemple à une des RBMs.
     * @param i L'indice de la RBM à entraîner.
     * @param x L'exemple à présenter.
     */
    public void learnRBM(int i, float[] x) {
        for(int j = 0; j < i; j++)
            x = rbms[j].calc(x);
        rbms[i].learn(x);
    }

    /**
     * Apprentissage par rétropropagation du gradient du réseau final.
     * @param x L'exemple à présenter.
     * @param out La sortie attendue.
     * @return Un tableau représentant la sortie de ce Deep Belief Network.
     */
    public float[] learnNet(float[] x, float[] out) {
        for(int j = 0; j < rbms.length; j++)
            x = rbms[j].calc(x);
        return network.learn(x, out);
    }

    /**
     * Teste un exemple.
     * @param x L'exemple à présenter.
     * @return Un tableau représentant la sortie de ce Deep Belief Network.
     */
    public float[] test(float[] x) {
        for(int j = 0; j < rbms.length; j++)
```

```
65              x = rbms[j].calc(x);
66          return network.test(x);
67      }
68
69      /**
70       * Teste un exemple mais cette fois-ci on n'utilise pas de MLP mais la RBM
  d'inversion pour obtenir le résultat
71       * @param x L'exemple à présenter.
72       * @return Un tableau représentant la sortie de ce Deep Belief Network.
73       */
74      public float[] test2(float[] x) {
75          for(int j = 0; j < rbms.length; j++)
76              x = rbms[j].calc(x);
77          float[] x2 = new float[inv.getInLen()];
78          for(int k = x.length; k < x2.length; k++)
79              x2[k] = 1 / (float) network.getOutLen();
80          for(int k = 0; k < x.length; k++)
81              x2[k] = x[k];
82          x2 = inv.calcAndInv(x2);
83          float[] res = new float[network.getOutLen()];
84          for(int k = 0; k < res.length; k++)
85              res[k] = x2[x.length + k];
86          return res;
87      }
88
89      /**
90       * Donne les représentations des impacts des composantes du vecteur d'entrée du
  DBN sur la (<b>i</b>+1)ème couche du réseau.
91       * @param i L'indice de la RBM pour laquelle on cherche l'influence de
  l'entrée.
92       * @return Un tableau qui contient pour chaque composante de la sortie de la
  <b>i</b>ème RBM, un tableau qui contient l'impact de
93       * toutes les composantes du vecteur que l'on présente en entrée du DBN.
94       */
95      public float[][] getRepresentations(int i) {
96          float[][] res = rbms[0].getWeights();
97          for(int j = 1; j <= i; j++)
98              res = rbms[j].getRepresentations(res, res[0].length);
99          return res;
100     }
101
102     /**
103      * Ajoute une RBM qui permet d'inverser le réseau pour générer de nouveau
  exemples.
104      * @param m Le nombres de neurones dans le couche cachée de la nouvelle RBM.
105      */
106     public void addInv(int m) {
107         inv = new RBM(network.getOutLen() + network.getInLen(), m);
108     }
109
110     /**
111      * Entraîne la RBM qui gère l'inversion.
112      * @param x L'exemple à présenter à la RBM pour l'apprentissage.
113      * @param res La sortie que serait sensé retourner le DBN s'il devait
  reconnaître l'exemple.
114      */
115     public void learnInv(float[] x, int res) {
116         for(int j = 0; j < rbms.length; j++)
117             x = rbms[j].calc(x);
118         float[] x2 = new float[inv.getInLen()];
119         x2[x.length + res] = 1;
120         for(int k = 0; k < x.length; k++)
121             x2[k] = x[k];
122         inv.learn(x2);
```

```
123     }
124
125     /**
126      * Génère un nouvel exemple.
127      * @param len Le nombre d'étapes à faire pour générer le nouvel exemple. Une
    image sera crée pour chaque étape.
128      * @param w La largeur des images qui seront crée à chaque étape.
129      * @param h La hauteur des images qui seront crée à chaque étape.
130      * @param firstX La première entrée que l'on présente au réseau pour générer le
    nouvel exemple.
131      * @param res La classe du nouvel exemple, la sortie que l'on voudrait obtenir
    si l'on présentait le nouvel exemple au réseau.
132      */
133     public void generate(int len, int w, int h, float[] firstX, int res) {
134         ImageUtil.save(ImageUtil.ImageFromPixels(firstX, w, h), "test.png");
135         for(int j = 0; j < rbms.length; j++)
136             firstX = rbms[j].calc(firstX);
137         float[] x2 = new float[inv.getInLen()];
138         for(int k = 0; k < firstX.length; k++)
139             x2[k] = firstX[k];
140         for(int i = 0; i < len; i++) {
141             firstX = new float[network.getInLen()];
142             for(int k = 0; k < network.getOutLen(); k++)
143                 x2[firstX.length + k] = (k == res) ? 1 : 0;
144             x2 = inv.calcAndInv(x2);
145             for(int k = 0; k < firstX.length; k++)
146                 firstX[k] = x2[k];
147             for(int j = rbms.length - 1; j >= 0; j--)
148                 firstX = rbms[j].inv(firstX);
149             ImageUtil.save(ImageUtil.ImageFromPixels(firstX, w, h),
    "test"+i+".png");
150         }
151     }
152     /**
153      * Permet de tester si la machine de Boltzmann restreinte est bien entrainer en
    comparant des image de chiffres
154      * aux images obtenues après un échantillonage de Gibbs jusqu'à la <b>i</b>ième
    couche
155      * de longeur <b>len</b> en partant des image initiales.
156      * @param len La longueur de l'échantillonage de Gibbs à effectuer.
157      * @param xs Un tableau à deux dimensions où <b>xs</b>[<b>j</b>][<b>k</b>] est
    le <b>k</b>ième pixel de la <b>j</b>ième image.
158      * @param w La largeur des images.
159      * @param h La hauteur des images.
160      * @param i L'indice de la couche jusqu'à laquelle on fait l'échantillonage de
    Gibbs.
161      */
162     public void testError(int len, float[][] xs, int w, int h, int i) {
163         for(int a = 0; a < xs.length; a++) {
164             ImageUtil.save(ImageUtil.ImageFromPixels(xs[a], w, h), a+"init.png");
165             for(int j = 0; j < len; j++) {
166                 for(int b = 0; b <= i; b++)
167                     xs[a] = rbms[b].calc(xs[a]);
168                 for(int b = i; b >= 0; b--)
169                     xs[a] = rbms[b].inv(xs[a]);
170             }
171             ImageUtil.save(ImageUtil.ImageFromPixels(xs[a], w, h), a+"rep.png");
172         }
173     }
174
175     /**
176      * Sauvegarde ce Deep Belief Network dans un fichier.
177      * @param fileName Le nom du fichier dans lequel est fait la sauvegarde.
178      * @throws IOException Ca peut arriver ...
```

```
179     */
180    public void save(String fileName) throws IOException {
181        ObjectOutputStream oos = new ObjectOutputStream(new
   FileOutputStream(fileName));
182        oos.writeObject(this);
183        oos.close();
184    }
185
186    public void setRBM(RBM rbm, int i) { rbms[i] = rbm; }
187    public RBM getRBM(int i) { return rbms[i]; }
188
189    /**
190     * Dernier Main utilisé.
191     * Le code de ce main a souvent changé, il m'a servi à faire des tests et des
   apprentissages.
192     * @param args Argument inutile ici.
193     */
194    public static void main(String[] args) {
195        try {
196            String file = "Networks/DBN2.dbn";
197            ObjectInputStream ois = new ObjectInputStream(new
   FileInputStream(file));
198            DBN dbn = (DBN) ois.readObject();
199            ois.close();
200 //           dbn.generate(100, 24, 32, ImageUtil.randomImage(32*24), 7);
201            dbn.generate(100, 24, 32, ImageUtil.pixels(ImageIO.read(new
   File("Images3/324.png"))), 7);
202            BufferedReader reader = new BufferedReader(new InputStreamReader(new
   FileInputStream(new File("Images2/nums.txt"))), 32768);
203            int[] nums = new int[10];
204            for(int i = 0; i < 10; i++)
205                nums[i] = Integer.parseInt(reader.readLine());
206            reader.close();
207            int it = 0;
208            while(it < 300000) {
209                if(it % 5000 == 0)
210                    System.out.println(it);
211                int n = (int) (Math.random()*10);
212                int i = (int) (Math.random()*(nums[n]+1));
213                float[] out = new float[10];
214                out[n] = 1;
215 //               dbn.learnRBM(1, ImageUtil.pixels(ImageIO.read(new
   File("Images3/"+n+""+i+".png"))));
216 //               dbn.learnNet(ImageUtil.pixels(ImageIO.read(new
   File("Images3/"+n+""+i+".png"))), out);
217                dbn.learnInv(ImageUtil.pixels(ImageIO.read(new
   File("Images3/"+n+""+i+".png"))), n);
218                it++;
219            }
220            float per = 0, err = 0, tot = 0;
221            for(int n = 0; n < 10; n++) {
222                tot += nums[n] + 1;
223                for(int i = 0; i <= nums[n]; i++) {
224                    float[] rep = dbn.test2(ImageUtil.pixels(ImageIO.read(new
   File("Images3/"+n+""+i+".png"))));
225                    float pe = 0;
226                    int nu = 0;
227                    for(int j = 0; j < 10; j++) {
228                        err += (rep[j] - ((j == n) ? 1 : 0)) * (rep[j] - ((j == n)
   ? 1 : 0));
229                        if(rep[j] > pe) {
230                            pe = rep[j];
231                            nu = j;
232                        }
```

```
233                     }
234                     if(nu == n)
235                         per += 1f;
236                 }
237             }
238             System.out.println("per : " + (per*100/tot) + "   err : " + err);
239             dbn.save(file);
240         } catch (IOException e) {
241             e.printStackTrace();
242         } catch (ClassNotFoundException e) {
243             e.printStackTrace();
244         }
245     }
246
247 }
```

```java
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.*;

import javax.imageio.ImageIO;
import javax.swing.*;

/**
 * Le programme final pour la reconnaissance du code postal sur une enveloppe
 * @author Yoann
 */
public class Final extends JFrame {

    private static final long serialVersionUID = 5970352975867769051L;
    private JButton loadIm = new JButton("Ouvrir Image"), loadNet = new JButton("Choisir Réseau"),
            getIms = new JButton("Obtenir Images"), calc = new JButton("Résultats"); //Les boutons du haut de la fenêtre.
    private JPanel topPanel = new JPanel(), letterPanel, imsPanel; //Les panneaux pour les boutons du haut, et l'affichage de l'enveloppe et des chiffres.
    private BufferedImage letter; //L'image de l'enveloppe.
    private Reseau net; //Le perceptron en cours d'utilisation.
    private DBN dbn; //Le DBN en cours d'utilisation.
    private boolean isNet = true, imsCreated = false, netloaded = false; //Les booléens pour savoir si on utilise un perceptron ou un DBN, pour savoir si les imagses ont été crées et pour savoir si le réseau est chargé.
    private String title = "Reconnaissance de code postal", im = "", res = ""; //Titre de la fenêtre et noms des fichiers.
    private BufferedImage[] ims; //Tableau des  petites images de chiffre.
    private JLabel resLabel = new JLabel(" "), perLabel = new JLabel(" "); //Les labels de résultats et de pourcentage.
    private Thread th; //Le thread pour les tâches de fond.

    /**
     * Crée une interface pour le programme final.
     */
    public Final() {
        setSize(750, 750);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle(title);
        loadIm.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JFileChooser fc = new JFileChooser();
                fc.setCurrentDirectory(new File("enveloppes"));
                fc.showOpenDialog(null);
                File file = fc.getSelectedFile();
                if(file == null)
                    return;
                try {
                    letter = ImageIO.read(file);
                    getIms.setEnabled(true);
                    calc.setEnabled(false);
                    imsCreated = false;
                    ims = null;
                    letterPanel.repaint();
                    imsPanel.repaint();
                    im = file.getName();
                    setTitle(title + " : " + im + "  -  " + res);
                } catch (IOException  e1) {
                    e1.printStackTrace();
                }
```

```
58              }
59          });
60          loadNet.addActionListener(new ActionListener() {
61              public void actionPerformed(ActionEvent e) {
62                  JFileChooser fc = new JFileChooser();
63                  fc.setCurrentDirectory(new File("Networks"));
64                  fc.showOpenDialog(null);
65                  File file = fc.getSelectedFile();
66                  if(file == null)
67                      return;
68                  try {
69                      ObjectInputStream ois = new ObjectInputStream(new
  FileInputStream(file));
70                      res = file.getName();
71                      if(res.endsWith("net")) {
72                          isNet = true;
73                          net = new Reseau((Network) ois.readObject());
74                          calc.setEnabled(imsCreated);
75                          netloaded = true;
76                      } else if(res.endsWith("dbn")) {
77                          isNet = false;
78                          dbn = (DBN) ois.readObject();
79                          calc.setEnabled(imsCreated);
80                          netloaded = true;
81                      }
82                      setTitle(title + " : " + im + "  -  " + res);
83                      ois.close();
84                  } catch (IOException | ClassNotFoundException e1) {
85                      e1.printStackTrace();
86                  }
87              }
88          });
89          getIms.addActionListener(new ActionListener() {
90              public void actionPerformed(ActionEvent e) {
91                  getIms.setEnabled(false);
92                  loadIm.setEnabled(false);
93                  th = new Thread(new Runnable() {
94                      public void run() {
95                          try {
96                              ims = ImageUtil.getDigits(letter);
97                              imsCreated = true;
98                              if(netloaded)
99                                  calc.setEnabled(true);
100                             resLabel.setText("");
101                         } catch (Exception e2) {
102                             e2.printStackTrace();
103                             resLabel.setText("Erreur lors de l'obtention des
  chiffres, veuillez regarder le dossier Traitement");
104                             getIms.setEnabled(true);
105                             topPanel.repaint();
106                         }
107                         imsPanel.repaint();
108                         loadIm.setEnabled(true);
109                     }
110                 });
111                 th.start();
112             }
113         });
114         calc.addActionListener(new ActionListener() {
115             public void actionPerformed(ActionEvent ae) {
116                 resLabel.setText("                                          ");
117                 perLabel.setText("                                (     ");
118                 if(isNet) {
119                     for(int i = 0; i < 5; i++) {
```

```
120                         float[] out = net.train3(ImageUtil.allinfo(ims[i]));
121                         resLabel.setText(resLabel.getText() + (int) out[0] + "
    ");
122                         perLabel.setText(perLabel.getText() + out[1] + "    ");
123                     }
124                 } else {
125                     for(int i = 0; i < 5; i++) {
126                         float[] out = dbn.test(ImageUtil.pixels(ims[i+5]));
127                         float b = 0, s = 0;
128                         for(int j = 0; j < 10; j++) {
129                             if(out[j] > s) {
130                                 b = j;
131                                 s = out[j];
132                             }
133                         }
134                         resLabel.setText(resLabel.getText() + (int) b + "   ");
135                         perLabel.setText(perLabel.getText() + s + "    ");
136                     }
137                 }
138                 resLabel.setText(resLabel.getText() + "
    ");
139                 perLabel.setText(perLabel.getText() + "  )
    ");
140             }
141         });
142         getIms.setEnabled(false);
143         calc.setEnabled(false);
144         topPanel.add(loadIm);
145         topPanel.add(loadNet);
146         topPanel.add(getIms);
147         topPanel.add(calc);
148         topPanel.add(resLabel);
149         topPanel.add(perLabel);
150         topPanel.setPreferredSize(new Dimension(750, 75));
151         letterPanel = new JPanel() {
152             private static final long serialVersionUID = -3757950555935332081L;
153             public void paintComponent(Graphics g) {
154                 g.setColor(Color.white);
155                 g.fillRect(0, 0, getWidth(), getHeight());
156                 if(letter != null) {
157                     float w = letter.getWidth(), h = letter.getHeight(), w2 =
    getWidth(), h2 = getHeight();
158                     float r = Math.min(w2/w, h2/h);
159                     g.drawImage(letter, (int) ((w2-w*r)/2), (int) ((h2-h*r)/2),
    (int) (w*r), (int) (h*r), null);
160                 }
161             }
162         };
163         imsPanel = new JPanel() {
164             private static final long serialVersionUID = 1L;
165             public void paintComponent(Graphics g) {
166                 g.setColor(Color.white);
167                 g.fillRect(0, 0, getWidth(), getHeight());
168                 if(ims != null) {
169                     float w = getWidth(), h = getHeight();
170                     float r = Math.min((w-18)/24/5, h/32);
171                     for(int i = 0; i < 5; i++)
172                         g.drawImage(ims[i], (int) ((w-5*24*r)/6*(i+1)+i*24*r),
    (int) ((h-32*r)/2), (int) (24*r), (int) (32*r), null);
173                 }
174             }
175         };
176         imsPanel.setPreferredSize(new Dimension(750, 48));
177         getContentPane().add(topPanel, BorderLayout.NORTH);
```

```
178         getContentPane().add(letterPanel, BorderLayout.CENTER);
179         getContentPane().add(imsPanel, BorderLayout.SOUTH);
180         setVisible(true);
181     }
182 
183     /**
184      * Le Main !
185      * @param args Argument inutile ici.
186      */
187     public static void main(String[] args) {
188         new Final();
189     }
190 
191 }
```