Etat de l'art des méthodes d'apprentissage et de classification en reconnaissance d'objets

2.1 Introduction

La reconnaissance de formes visuelles, tels que des objets ou des entités assimilables, a fait l'objet de recherches actives ayant conduit à de nombreuses approches. Les systèmes les plus performants sont basés sur des méthodes d'apprentissage et de classification. Ils permettent, à partir de données brutes représentatives de la variabilité de la classe d'objet à reconnaître, d'en extraire les informations caractéristiques. Ce chapitre présente les méthodes de classification et d'apprentissage les plus souvent utilisées en détection et reconnaissance de formes.

Nous diviserons ce chapitre en deux parties constituées d'une part, par les approches génératives consistant à modéliser la classe d'objet à détecter et ne requérant donc que des exemples de cette classe et d'autre part, par les approches discriminatives consistant à modéliser finement la frontière de cette classe par rapport au 'reste du monde', et exigeant à la fois des exemples de la classe à détecter, mais aussi des contre-exemples.

Ces méthodes sont utilisées dans de nombreux domaines pour un grand nombre d'applications, allant de la reconnaissance de la parole, à la compression ou la classification d'images. En ce qui concerne la détection et la reconnaissance d'objets, certaines de ces méthodes sont utilisées pour extraire les descripteurs caractéristiques des données de référence, d'autres utilisent ces descripteurs afin d'effectuer une classification correcte des données d'entrée. L'utilisation d'une technique de classification plutôt qu'une autre est fortement dépendante des contraintes liées à la tâche que nous souhaitons effectuer. Une large part des systèmes de détection ou de reconnaissance actuels utilisent d'ailleurs une combinaison de plusieurs des méthodes présentées ci-dessous.

2.2 Méthodes Génératives

En détection d'objets, les classifieurs doivent distinguer la classe 'objet' de la classe 'non objet'. Si la classe objet peut être convenablement décrite à partir d'un nombre suffisant d'images exemples, ce n'est pas le cas de la classe 'non objet' qui correspond à l'ensemble des images qui ne représentent pas l'objet à détecter. L'intérêt des méthodes génératives est que nous n'avons pas besoin de représenter la classe 'non objet'.

2.2.1 Plus proches voisins

En classification d'images, les images utilisées sont généralement redimensionnées de manière à ce que les systèmes de classification travaillent toujours sur des données de même dimension. Ainsi les images sont représentées sous forme vectorielle $\mathbf{x} = (x_1, \ldots, x_d)^T$ de dimension $d = h \times l$, h et l représentant respectivement la hauteur et la largeur en pixels des images à classer.

La méthode de classification par plus proches voisins est non spécifique au traitement d'image et peut être utilisée pour classer toute donnée représentable sous formes de vecteurs. Elle consiste à classer les données d'entrée en fonction de leur distance à l'exemple le plus proche de la base de référence. Soit $D^N = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ un ensemble de N échantillons exemples labélisés, représentatifs des données à classer. Dans le cadre de la détection d'objets, chaque échantillon exemple $\mathbf{x_i}$ est un vecteur représentant une image de l'objet à détecter. Le principe de la méthode des plus proches voisins est d'assigner à tout échantillon à classer $\mathbf{y}$ la catégorie ω_i de l'échantillon de D^N le plus proche, ou dans le cas de la détection d'objets, à classer l'échantillon $\mathbf{y}$ dans la catégorie 'objet' si la distance à l'échantillon le plus proche est inférieure à un seuil donné et dans la catégorie 'non objet' sinon.

Cette méthode est simple à mettre en œuvre. De plus, lorsque le nombre N d'échantillons exemples tend vers l'infini, l'erreur de classification est toujours inférieure à deux fois le minimum possible défini par les lois de Bayes [26], ce qui assure d'excellents résultats à condition de disposer de suffisamment d'exemples. Enfin cette méthode ne nécessite aucune phase d'apprentissage. Cependant chaque échantillon de test $\mathbf{y}$ présenté doit être comparé à l'ensemble des échantillons exemples, ainsi la complexité de ce classifieur est directement proportionnel à N le nombre d'échantillons d'apprentissage.

2.2.1.1 Mesure de distance

La méthode des plus proches voisins implique nécessairement l'utilisation d'une notion de distance entre deux échantillons $\mathbf{x}$ et $\mathbf{y}$.

D'un point de vue mathématique, une distance D doit respecter les quatre règles suivantes :

1. définie positive : $D(\mathbf{x}, \mathbf{y}) \geq 0$
2. réflectivité : $D(\mathbf{x}, \mathbf{y}) = 0$ si et seulement si $\mathbf{x} = \mathbf{y}$
3. symetrie : $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$

4. inégalité triangulaire : $D(\mathbf{x},\mathbf{y}) + D(\mathbf{y},\mathbf{z}) \geq D(\mathbf{x},\mathbf{z})$

La mesure de distance la plus couramment utilisée est la distance euclidienne :

$$D(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{\frac{1}{2}} \tag{2.1}$$

Ou encore la distance de Minkowski, généralisation de la distance euclidienne :

$$D(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^k \right)^{\frac{1}{k}} \tag{2.2}$$

Cependant d'un point de vue pratique, la notion de distance utilisée par la méthode des plus proches voisins ne doit pas nécessairement être une distance mathématique. Ainsi nous pouvons utiliser différentes mesures de similarité afin de définir la distance entre deux échantillons. Les deux échantillons les plus proches devenant alors les plus similaires. Des systèmes de détection ou de reconnaissance basés sur la méthode des plus proches voisins peuvent utiliser des mesures de similarité mieux adaptées au traitement d'image telles que la corrélation normée centrée, utilisée dans [68] pour un système de détection de formes et par Wakahara *et al* [123] dans le cadre d'un système de reconnaissance de caractères :

$$S(\mathbf{x},\mathbf{y}) = \frac{1}{\sigma_x \sigma_y} \left(\mathbf{x}^T \mathbf{y} - m_x m_y \right) \tag{2.3}$$

m_x et m_y étant les moyennes respectives des vecteurs $\mathbf{x}$ et $\mathbf{y}$

$$m_x = \sum_{i=1}^{d} x_i \quad m_y = \sum_{i=1}^{d} y_i \tag{2.4}$$

σ_x et σ_y les variances respectives des vecteurs $\mathbf{x}$ et $\mathbf{y}$

$$\sigma_x = \left(\mathbf{x}^T \mathbf{x} - m_x^2 \right)^{\frac{1}{2}} \quad \sigma_y = \left(\mathbf{y}^T \mathbf{y} - m_y^2 \right)^{\frac{1}{2}} \tag{2.5}$$

2.2.2 Analyse en Composantes Principales

En traitement du signal et plus particulièrement en traitement d'image, la dimension des données utilisées pour décrire une information est souvent très grande. L'espace représentant les images de 100×100 pixels est déjà de dimension $d = 10000$. Lorsque nous souhaitons classifier ces données, les temps de calculs ainsi que le nombre d'exemples nécessaires à la représentation de ces données sont directement liés à leur dimension. De plus le volume de données associé à l'ajout de dimensions

supplémentaire dans un espace mathématique augmente exponentiellement avec la dimension. Ce problème nommé par Belman 'malédiction de la dimension' rend complexe l'analyse statistique de problèmes de grande dimension et nécessite l'ajout d'informations a priori dans le but de contraindre le système d'apprentissage.

Une des approches pour réduire la dimension de ces données est de projeter les vecteurs $\mathbf{x} = (x_1, \ldots, x_d)^T$ représentant chaque pixel des images de référence, dans une base de dimension plus réduite représentée par la matrice W permettant toujours de convenablement décrire l'ensemble des échantillons exemples. Ainsi, chaque vecteur $\mathbf{x}$ de dimension d est projeté dans un espace de dimension réduite k par la matrice de projection W^T. Chaque donnée d'entrée est alors représentée par un vecteur $\mathbf{s} = (s_1, \ldots, s_k)^T$ avec $k \leq d$.

$$\mathbf{s} = W^T \mathbf{x} \tag{2.6}$$

Cette approche est appelée méthode de projection linéaire et est très utilisée en traitement d'image pour son efficacité et sa simplicité.

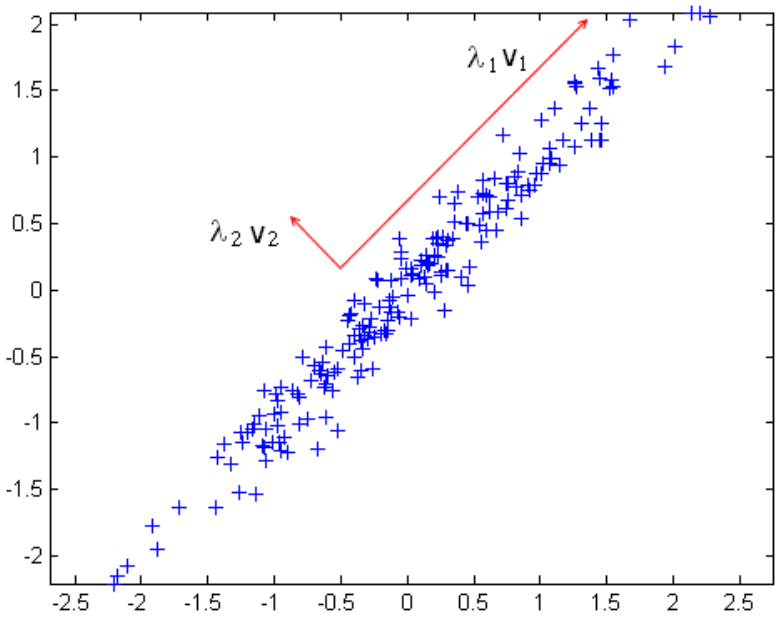


FIGURE 2.1 – Analyse en Composantes Principales d'une distribution gaussienne de points centrée sur 0. Nous remarquons que l'essentiel des informations sur la distribution des points peut être conservé en projetant l'ensemble des points sur le vecteur propre $\mathbf{v_1}$ correspondant à la valeur propre λ_1 la plus grande de la matrice de covariance

Il y a généralement une grande corrélation, non seulement entre les images d'une même classe mais aussi entre les pixels voisins d'une même image. Ainsi en traitement d'image, une des approches les plus classique pour déterminer un espace de

projection représenté par la matrice W est l'Analyse en Composantes Principales (PCA pour Principal Component Analysis). Aussi connue sous le nom de transformée de Karhunen-Loève (KLT) ou de transformée de Hotelling [44], la technique de l'Analyse en Composantes Principales est attribuée à K. Pearson [89]. Cette méthode consiste à décrire un vecteur comme la combinaison linéaire de k vecteurs orthogonaux entre eux (figure : 2.1). Ces vecteurs sont calculés de façon à minimiser la distance $L2$ entre l'ensemble des vecteurs reconstruits et les vecteurs de référence. Autrement dit, la PCA définit la matrice de projection W^T de dimension $k \times d$ telle que $E = \sum_{i=1}^{N} \|\mathbf{x_i} - W\mathbf{s_i}\|$ soit minimum.

Soit, $\{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ un ensemble de N vecteurs centrés, *i.e*, $\sum_{i=1}^{N} \mathbf{x_i} = \mathbf{0}$. La matrice de projection W définie par l'Analyse en Composantes Principales est alors formée des k vecteurs propres correspondant aux plus grandes valeurs propres de la matrice de covariance :

$$\Sigma = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x_i} \mathbf{x_i}^T \tag{2.7}$$

La matrice Σ étant symétrique, on peut, par décomposition en valeurs propres, écrire :

$$\Sigma = \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_d} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots \\ 0 & \cdots & \ddots & \cdots \\ 0 & \cdots & 0 & \lambda_d \end{pmatrix} \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_d} \end{pmatrix}^T \tag{2.8}$$

L'ensemble des vecteurs $\{\mathbf{v_1}, \cdots, \mathbf{v_d}\}$ forme une base orthonormée. *i.e*, si $i \neq j$ $\mathbf{v_i}^T \mathbf{v_j} = 0$ et $\forall\, i \in [1, d]\,, \|\mathbf{v_i}\| = 1$.

Si les valeurs propres sont classées par ordre décroissant, *i.e*, $\forall i \in [1, d-1]\,, \lambda_i \geq \lambda_{i+1}$. Alors la matrice de projection W de la PCA s'écrit :

$$W = (\mathbf{v_1}, \cdots, \mathbf{v_k}) \tag{2.9}$$

La valeur de k dépend fortement de l'application et des données sur lesquelles la PCA est utilisée. Plus k est petit, plus l'erreur de reconstruction sera grande. Ainsi, la dimension de l'espace de projection est un compromis entre le besoin de compression des données de départ et la perte d'information due à cette compression.

Cette méthode est particulièrement utilisée en reconnaissance et détection de visages. En effet Kirby et Sirovich ont montré que les images de visages pouvaient être représentées par la combinaison linéaire d'un faible nombre de vecteurs propres [53]. Ainsi dans [117], Turk et Pentland décrivent un système de reconnaissance de visages basé sur une réduction de la dimension des images de visages par l'utilisation de la PCA. Chaque visage à reconnaître est alors projeté dans le sous-espace défini par la matrice W. La classification s'effectue ensuite par la méthode des plus proches

voisins, *i.e*, chaque image de visage présentée au système de classification est associée au visage le plus proche dans le sous-espace des vecteurs propres. Dans [116] la méthode de la PCA est associée à une méthode de clustering (k-Mean) par Sung et Poggio afin de réduire la dimension des données d'entrée d'un Classifieur. Chaque imagette de 19×19 (soit 361 pixels) est projetée dans douze sous-espaces de 75 vecteurs propres. Chaque image est ensuite caractérisée par différentes distances à ces sous-espaces afin d'obtenir un vecteur caractéristique de dimension 24 qui sera classée par un Perceptron Multicouche (MLP Multi Layer Perceptron).

Pour résumer, La PCA est souvent utilisée en traitement d'image afin de représenter les images à classifier avec un minimum de descripteurs. Les images ainsi représentées pourront alors être classées dans la catégorie appropriée plus rapidement et avec moins d'exemples. Cependant la PCA est bien plus efficace pour représenter l'ensemble des variations de textures et de luminosité des objets que pour modéliser les déformations et les variations d'angle de prise de vue des images. La méthode des Modèle d'Apparence Actifs décrite ci-dessous tente de résoudre ce problème.

2.2.3 Modèles d'Apparence Actifs

La technique des Modèles d'Apparences Actifs (AAM pour Active Appearence Models) a été introduite par Cootes, Ewards et Taylor [19] dans le but de représenter avec un minimum de descripteurs des objets déformables tels que des visages. Cette méthode consiste à décrire un objet comme une combinaison linéaire de textures déformées selon une méthode basée sur les modèles actifs de forme [20]. La méthode des Modèles d'Apparence Actifs consiste à déplacer des points d'intérêt selon les principaux modes de déformation associés à l'objet. Ces modes de déformation sont déterminés par la méthode de la PCA appliquée a un grand nombre d'exemples manuellement annotés (figure :2.2).

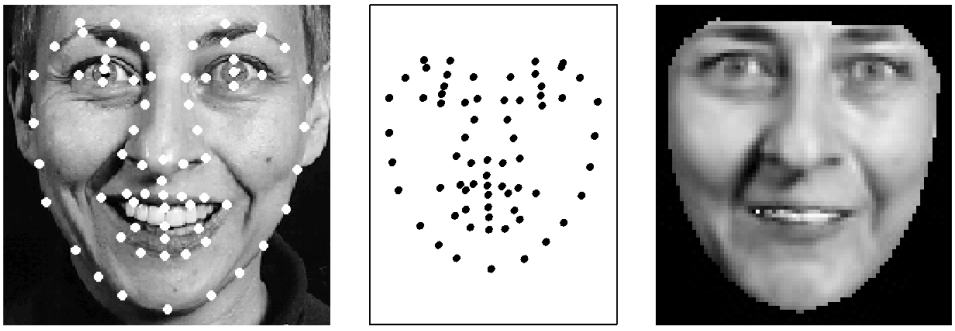


FIGURE 2.2 – Méthode des Modèles d'Apparence Actifs appliqués à un visage dans [19], à gauche, un exemple de visage utilisé avec les points d'intérêt annotés correspondant, au centre, les positions moyennes des points d'intérêt pour l'ensemble des images exemples et à droite, l'image du visage exemple reconstituée à partir de la position des points d'intérêt définis au centre

Les coordonnées $\mathbf{x}$ des points d'intérêt dans une image sont définies par l'équation

suivante :

$$\mathbf{x} = \bar{\mathbf{x}} + P_s \mathbf{b_s} \tag{2.10}$$

$\bar{\mathbf{x}}$ correspond aux coordonnées moyennes des point d'intérêt sur l'ensemble des images de référence, *i.e*, si $\mathbf{x_i}$ représente les coordonnées de l'ensemble des points d'intérêt associés à la i^{ieme} image de référence alors :

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x_i} \tag{2.11}$$

P_s est le sous-espace formé par les k_s premiers vecteurs propres de la matrice de covariance des points d'intérêt. $\mathbf{b_s}$ est donc un vecteur de dimension k_s représentant la position relative des points d'intérêt par rapport à leur position moyenne.

De même, il est possible de représenter avec l'aide de la méthode de la PCA la texture $\mathbf{g}$ qui sera déformée de façon à correspondre au mieux à l'objet à représenter. $\mathbf{g}$ correspond à l'image de l'objet à représenter avec les points d'intérêt dans la position définie par $\bar{\mathbf{x}}$ (image de droite dans la figure : 2.2).

$$\mathbf{g} = \bar{\mathbf{g}} + P_g \mathbf{b_g} \tag{2.12}$$

La forme et la texture associées aux objets à représenter sont généralement corrélées. C'est pourquoi, la méthode des AAM effectue une troisième PCA appliquée à la concaténation des vecteurs b_s et b_g. Ainsi, la méthode des Modèles d'Apparence Actifs décrit à la fois la forme et la texture des objets à représenter par un unique vecteur c.

$$\mathbf{x} = \bar{\mathbf{x}} + Q_s \mathbf{c} \tag{2.13}$$

$$\mathbf{g} = \bar{\mathbf{g}} + Q_g \mathbf{c} \tag{2.14}$$

Connaissant le vecteur $\mathbf{c}$ la méthode des AAM permet de synthétiser l'objet représenté en déformant la texture $\mathbf{g}$ selon la position des points d'intérêt $\mathbf{x}$. Les matrices Q_s et Q_g sont calculées à partir d'une base de référence manuellement annotée et représentent les déformations et variations de textures associées aux objets que nous souhaitons représenter.

2.2.3.1 Décrire une image avec les Modèles d'Apparence Actifs

Les matrices Q_s et Q_g étant à présent connues, le problème est maintenant de déterminer l'image synthétisée qui se rapproche le plus de l'image que nous souhaitons représenter, *i.e*, trouver le vecteur $\mathbf{c}$ qui minimise la distance entre la texture modélisée représentée par le vecteur $\mathbf{g_m}(\mathbf{x}) = \bar{\mathbf{g}} + Q_g \mathbf{c}$ et la texture $\mathbf{g_s}(\mathbf{x})$ correspondant à l'image déformée que nous souhaitons décrire. La méthode utilisée est une optimisation itérative basée sur la descente par gradient.

Soit $\mathbf{r}(\mathbf{c}) = \mathbf{g_s} - \mathbf{g_m}$, alors décrire une image par la méthode des Modèles d'Apparence Actifs revient à minimiser la fonction :

$$E(\mathbf{c}) = \mathbf{r}(\mathbf{c})^T \mathbf{r}(\mathbf{c}) = \|\mathbf{r}(\mathbf{c})\| \tag{2.15}$$

Ainsi nous cherchons $\mathbf{c}$ tel que $\mathbf{r}(\mathbf{c}) = 0$. Ce problème est résolu par la méthode de Gauss Newton. Un développement limité au premier ordre nous donne :

$$\mathbf{r}(\mathbf{c} + \delta\mathbf{c}) = \mathbf{r}(\mathbf{c}) + \frac{\partial \mathbf{r}}{\partial \mathbf{c}} \delta\mathbf{c} \tag{2.16}$$

$$\frac{\partial \mathbf{r}}{\partial \mathbf{c}} = \begin{pmatrix} (\nabla_{\mathbf{c}} r_i)^T \\ \vdots \end{pmatrix} = \begin{pmatrix} \frac{\partial r_i}{\partial c_j} & \cdots \\ \vdots & \ddots \end{pmatrix} \tag{2.17}$$

r_i et c_j représentant respectivement les éléments des vecteurs $\mathbf{r}$ et $\mathbf{c}$.
Cette équation est résolue par l'utilisation de la matrice pseudo inverse R :

$$\delta\mathbf{c} = R\mathbf{r}(\mathbf{c}) \tag{2.18}$$

$$R = \left(\frac{\partial \mathbf{r}}{\partial \mathbf{c}}^T \frac{\partial \mathbf{r}}{\partial \mathbf{c}} \right)^{-1} \frac{\partial \mathbf{r}}{\partial \mathbf{c}}^T \tag{2.19}$$

L'approximation linéaire est ensuite itérativement répétée jusqu'à convergence, *i.e*, nous réestimons une nouvelle valeur de $\mathbf{c}$ à partir de la valeur précédente jusqu'a ce que $E(\mathbf{c})$ soit minimum.

$$\mathbf{c_{i+1}} = \mathbf{c_i} + R_i \mathbf{r}(\mathbf{c_i}) \tag{2.20}$$

Cette méthode est particulièrement utilisée pour localiser précisément des visages ou des éléments des visages (yeux, bouche, nez), déterminer l'orientation d'un visage ou estimer la forme d'un visage dans une pose donnée à partir d'images du visage dans une pose différente [21]. En reconnaissance de visages, elle permet d'effectuer la reconnaissance en minimisant les variations dues aux angles de prise de vue et aux déformations inhérentes à un visage [9, 29]. Cette méthode est aussi utilisée pour la compression puisqu'elle permet de représenter une catégorie d'objets avec peu de descripteurs. Les Modèles d'Apparence Actifs sont aussi utilisés en détection [30], un objet étant détecté s'il est suffisamment bien représenté par la méthode des AAM, *i.e*, si la différence entre l'image réelle de cet objet et l'image modélisée est suffisamment petite, le seuil de détection étant fixé de manière pratique. Cette méthode est très efficace pour représenter et décrire les petits objets déformables tels que des visages, car elle modélise aussi bien les textures de l'objet que les déformations possibles associées. Cependant, elle nécessite un procédé d'optimisation qui la rend bien moins rapide qu'une simple PCA. De plus, il est nécessaire d'annoter manuellement de nombreux points d'intérêt sur plusieurs centaines d'images exemples.

2.2.4 Maximum de Vraisemblance

La méthode du Maximum de Vraisemblance consiste à représenter une classe d'exemples donnés comme la densité de probabilité qu'un échantillon représenté par le vecteur $\mathbf{x}$ appartienne à la classe à modéliser. Cette méthode suppose que nous connaissons la forme de la densité de probabilité recherchée à l'exception d'un ensemble de paramètres représenté par le vecteur $\boldsymbol{\theta}$, ainsi la probabilité qu'un vecteur $\mathbf{x}$ appartienne à la classe que nous souhaitons représenter s'écrit :

$$P(\mathbf{x}|\boldsymbol{\theta}) \tag{2.21}$$

La méthode du Maximum de Vraisemblance consiste à déterminer les paramètres $\boldsymbol{\theta}$ qui maximisent la vraisemblance que la densité de probabilité $P(\mathbf{x}|\boldsymbol{\theta})$ génère l'ensemble $D^N = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ d'échantillons exemples labellisés, *i.e* : trouver $\boldsymbol{\theta}$ tel que $P(D|\boldsymbol{\theta})$ soit maximum. Si les échantillons $\{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ sont indépendants, nous pouvons écrire :

$$P(D|\boldsymbol{\theta}) = \prod_{i=1}^{N} P(\mathbf{x_i}|\boldsymbol{\theta}) \tag{2.22}$$

Résoudre ce problème analytiquement consiste à trouver les valeurs de $\boldsymbol{\theta}$ pour lesquels le gradient de $P(D|\boldsymbol{\theta})$ est nul. Si on définit le logarithme de la vraisemblance par la fonction $l(\boldsymbol{\theta}) = \ln P(D|\boldsymbol{\theta})$, alors :

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{N} \ln P(\mathbf{x_i}|\boldsymbol{\theta}) \tag{2.23}$$

La valeurs de $\boldsymbol{\theta}$ maximisant la vraisemblance se déduit alors en résolvant l'équation suivante :

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ln P(\mathbf{x_i}|\boldsymbol{\theta}) = 0 \tag{2.24}$$

Par exemple, si on définit $P(\mathbf{x}|\boldsymbol{\theta})$ comme une loi normale :

$$P(\mathbf{x}|\boldsymbol{\theta}) = P(x|m, \sigma) = \frac{1}{\sigma(2\pi)^{\frac{1}{2}}} e^{-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2} \tag{2.25}$$

$\boldsymbol{\theta}$ est un vecteur à deux dimensions contenant la moyenne m et la variance σ de la loi normale. Par la méthode du Maximum de Vraisemblance, nous trouvons des résultats intuitivement satisfaisants. En effet les valeurs estimées m et σ correspondent respectivement à la moyenne et à l'écart type des échantillons exemples D.

Une loi normale est cependant généralement trop simple pour modéliser des classes complexes d'échantillons telles que des images d'objets. La méthode des mélanges de Gaussiennes (GMM Gaussian Mixture Models) a été introduite dans le but de pouvoir représenter des densités de probabilité complexes et ainsi de pouvoir s'adapter à un grand nombre de problèmes de modélisation et de décision.

2.2.4.1 Mélange de Gaussiennes

La méthode des Mélanges de Gaussienne (GMM) consiste à représenter l'ensemble des échantillons exemples comme une densité de probabilité formée par la somme pondérée de k Gaussiennes *i.e* :

$$P(\mathbf{x}|\boldsymbol{\theta}) = \sum_{i=1}^{k} a_i G(\boldsymbol{\mu_i}, \Sigma_i) \tag{2.26}$$

$$G(\boldsymbol{\mu_i}, \Sigma_i) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_i|^{\frac{1}{2}}} e^{(\mathbf{x}-\boldsymbol{\mu_i})^T \Sigma_i^{-1} (\mathbf{x}-\boldsymbol{\mu_i})} \tag{2.27}$$

$$\sum_{i=1}^{k} a_i = 1 \tag{2.28}$$

Les paramètres $\boldsymbol{\theta} = \{a_1, \boldsymbol{\mu_1}, \Sigma_1, \ldots, a_k, \boldsymbol{\mu_k}, \Sigma_k\}$ sont calculés de façon à maximiser la vraisemblance des échantillons $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ *i.e*, trouver $\boldsymbol{\theta}$ tel que $P(D|\boldsymbol{\theta})$ soit maximum :

$$P(D|\boldsymbol{\theta}) = \prod_{i=1}^{N} P(\mathbf{x_i}|\boldsymbol{\theta}) \tag{2.29}$$

Cette méthode à été utilisée avec succès notamment dans le domaine de la reconnaissance de locuteurs indépendante du texte prononcé [95]. La voix est découpée en segments d'environs 20 ms, et un vecteur $\mathbf{x}$ caractéristique de ce segment en est extrait, généralement en utilisant la méthode de la LPC (Linear Predictive Coding). Un Mélange de Gaussiennes est ensuite utilisé afin de modéliser la probabilité pour une personne donnée d'émettre un son caractérisé par un vecteur $\mathbf{x}$. Ainsi, si nous disposons d'un échantillon de voix d'une personne, il devient possible de déterminer la probabilité que cette voix corresponde à celle modélisée par le Mélange de Gaussiennes.

Dans le domaine du traitement d'image, les Mélanges de Gaussiennes ont de multiples applications, on peut citer notamment la classification d'images [91]. La méthode des GMM est utilisée afin de modéliser la probabilité de présence de certaines textures et couleurs dans différentes catégories d'images. Par exemple, une classe d'images représentant la mer aura de fortes chances de voir apparaître la couleur bleue et certaines textures particulières, ce qui sera modélisé par le Mélange de Gaussiennes. Si nous devons ensuite classer une image présentant une grande proportion de bleu,

elle aura alors une probabilité importante d'être classée dans cette catégorie. Dans [77], Moghaddam et Pentland présentent une méthode basée sur les GMM permettant de déterminer la probabilité $P(\mathbf{x}|\Omega)$ qu'une image $\mathbf{x}$ représente un objet de la classe Ω et l'appliquent à la détection de visages et de mains dans une image.

Une des difficultés posée par la méthode des Mélanges de Gaussiennes est que l'expression (2.29) n'est pas une fonction linéaire des paramètres $\boldsymbol{\theta}$, ainsi une maximisation directe est impossible en pratique. Une des méthodes les plus classique permettant de résoudre ce problème est un algorithme itératif appelé algorithme EM pour Expectation Maximization [25].

2.2.4.2 Algorithme EM

L'idée de base de l'algorithme EM est d'utiliser les paramètres $\boldsymbol{\theta^t}$ à la t^{ieme} itération afin d'estimer de nouvelles valeurs des paramètres $\boldsymbol{\theta^{t+1}}$ telles que $P(D|\boldsymbol{\theta^{t+1}}) \geq P(D|\boldsymbol{\theta^t})$.
Considérons un ensemble d'échantillons $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$. Supposons que des descripteurs soient manquants, un échantillon peut alors s'écrire sous la forme $\mathbf{x_i} = (\mathbf{x_{ig}}, \mathbf{x_{ib}})$ avec $\mathbf{x_{ig}}$ les 'bons' (good) descripteurs et $\mathbf{x_{ib}}$ les 'mauvais' (bad). On notera D_g l'ensemble des 'bons' descripteurs et D_b l'ensemble des 'mauvais'(descripteurs manquants)

Soit $Q(\boldsymbol{\theta}; \boldsymbol{\theta^t})$, l'espérance du logarithme de la vraisemblance de $P(D_g, D_b)$ connaissant une précédente estimation $\boldsymbol{\theta}^t$ des paramètres de la loi de probabilité de $P(\mathbf{x})$ et les 'bons' descripteur D_g. Alors si on note :

$$Q(\boldsymbol{\theta}; \boldsymbol{\theta^t}) = E_{D_b}[\ln P(D_g, D_b|\boldsymbol{\theta})|D_g; \boldsymbol{\theta^t}] \tag{2.30}$$

Algorithm 1 Expectation Maximum

1: initialisation : $\boldsymbol{\theta}^0, T, t = 0$
2: **repeat**
3: $\quad Q(\boldsymbol{\theta}; \boldsymbol{\theta^t})$
4: $\quad \boldsymbol{\theta^{t+1}} = \arg\max Q(\boldsymbol{\theta}; \boldsymbol{\theta^t})$
5: **until** $Q(\boldsymbol{\theta^{t+1}}; \boldsymbol{\theta^t}) - Q(\boldsymbol{\theta^t}; \boldsymbol{\theta^{t-1}}) \leq T$
6: **return** $\boldsymbol{\theta} = \boldsymbol{\theta^{t+1}}$

Cet algorithme garantit que la vraisemblance des échantillons D_g augmente à chaque itération. C'est pourquoi il est très souvent utilisé dans les problèmes de maximisation de vraisemblance.

Dans le cas du problème du GMM, si on note, $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ l'ensemble des échantillons exemples. On définit $z_i \in \{1, \ldots, k\}$ l'information manquante déterminant de quelle Gaussienne du mélange est issue l'échantillon $\mathbf{x_i}$. $\boldsymbol{\theta}$ est l'ensemble des paramètres à estimer du Mélange de Gaussiennes. Alors la fonction $Q(\boldsymbol{\theta}; \boldsymbol{\theta^t})$ est déterminée comme suit :

$$
\begin{aligned}
P(D_g, D_b|\boldsymbol{\theta}) &= P(z_i = j, \mathbf{x_i}|\boldsymbol{\theta}) \qquad (2.31)\\
Q(\boldsymbol{\theta}; \boldsymbol{\theta^t}) &= E_z[\ln P(D_g, D_b; \boldsymbol{\theta})|D_g; \boldsymbol{\theta^t}] \\
&= \sum_{i=1}^{N} E_z\left[\ln P(\mathbf{x_i}, z|\boldsymbol{\theta})|\mathbf{x_i}, \boldsymbol{\theta^t}\right] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{k} P(z_i = j|\mathbf{x_i}, \boldsymbol{\theta^t}) \ln P(\mathbf{x_i}, z_i = j|\boldsymbol{\theta}) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{k} P(z_i = j|\mathbf{x_i}, \boldsymbol{\theta^t}) \ln \left(P(\mathbf{x_i}|z_i = j, \boldsymbol{\theta}) P(z_i = j|\boldsymbol{\theta})\right)
\end{aligned}
$$

Avec :

$$P(\mathbf{x_i}|z_i = j, \boldsymbol{\theta}) = G(\mathbf{x_i}; \boldsymbol{\mu_j}, \Sigma_j) \qquad (2.32)$$

$$P(z_i = j|\boldsymbol{\theta}) = a_j \qquad (2.33)$$

$$P(z_i = j|\mathbf{x_i}, \boldsymbol{\theta^t}) = \frac{a_j^t G(\mathbf{x_i}; \boldsymbol{\mu_j^t}, \Sigma_j^t)}{\sum_{i=1}^{k} a_i^t G(\mathbf{x_i}; \boldsymbol{\mu_j^t}, \Sigma_j^t)} \qquad (2.34)$$

La nouvelle estimation des paramètres $\boldsymbol{\mu_i}$ et Σ_i peut alors être effectuée de manière analytique par le calcul de la dérivée de $Q(\boldsymbol{\theta}; \boldsymbol{\theta^t})$. Et dans le cas des paramètres a_i, par la méthode du Lagrangien afin de respecter la contrainte $\sum_{i=1}^{k} a_i = 1$.

La méthode des Mélanges de Gaussiennes est efficace pour modéliser les distributions statistiques d'échantillons exemples et représenter ainsi, des images, ou un signal quelconque par la probabilité d'apparition d'un certain nombre d'éléments caractéristiques. Cependant un tel modèle ne tient pas compte d'éventuelles relations temporelles ou spatiales entre les différents échantillons. La section suivante présente une approche toujours basée sur la maximisation de la vraisemblance mais permettant de tenir compte des transformations locales d'un signal en modélisant celui-ci comme une séquence d'événements.

2.2.5 Modèles de Markov Cachés

La méthode des Modèles de Markov cachés (HMM pour Hidden Markov Models) a été introduite par Rabinet [94]. Cette méthode de modélisation permet de représenter un signal, ou plus généralement une information, comme une séquence d'états successifs. L'état du système à l'instant t dépendant directement de l'état à l'instant $t-1$. Ainsi, cette méthode est particulièrement efficace pour représenter des informations présentant un aspect temporel. Elle est notamment très utilisée en traitement de la parole ou elle permet de modéliser la succession temporelle des phonèmes qui composent les mots. Bien que les images ne présentent à la base pas d'aspect temporel, celles ci peuvent être représentées comme une succession unidimensionnelle d'observations, modélisables par un Modèle de Markov Caché. Le plus souvent les

images sont divisées en sous-régions, chaque sous-région de l'image correspondant à une observation [106], la méthode des HMM permet alors de modéliser quelles successions de sous-régions sont les plus probables pour la classe d'image que nous avons modélisée. Une autre solution consiste à utiliser des versions modifiées de la méthode des HMM telle que les 2D-HMM ou encore les pseudo 2D-HMM afin de tenir compte de la bidimensionnalité des images.

2.2.5.1 Architecture d'un Modèle de Markov

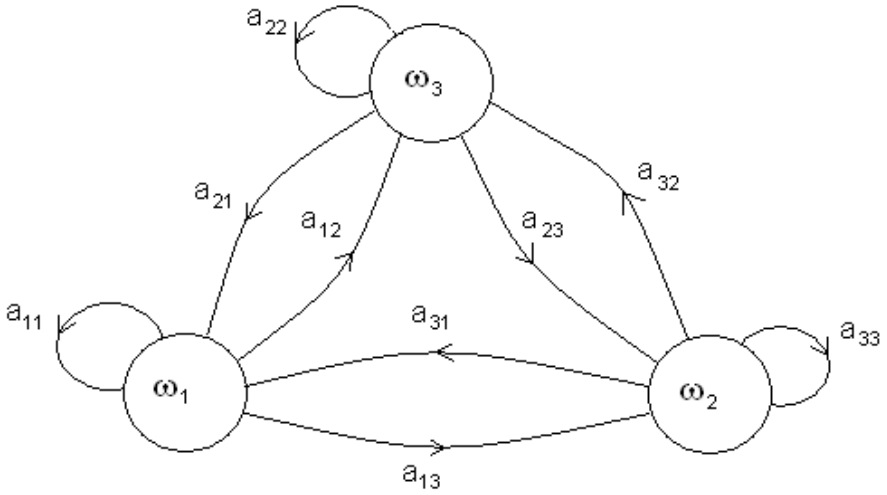


FIGURE 2.3 – Modèle de Markov basique. Les noeuds représentent les états que peut prendre le système à un instant t. Les probabilités de transition a_{ij} sont représentées par les liens entre les noeuds.

Un Modèle de Markov (figure : 2.3) est défini ainsi :
Considérons un ensemble de T états successifs $\omega^T = \{\omega(1), \omega(2), \ldots, \omega(T)\}$, $\omega(t)$ représentant l'état du système à l'instant t. A chaque instant t, un Modèle de Markov ne peut se trouver que dans un état ω_i donné parmi un nombre fini N d'états. Ainsi une séquence particulière de longueur six pour un Modèle de Markov à trois états pourrait être $\omega^6 = \{\omega_1, \omega_3, \omega_3, \omega_2, \omega_1, \omega_2\}$. Dans un Modèle de Markov du premier ordre, la probabilité de se trouver dans l'état ω_j à l'instant t dépend directement de l'état ω_i du système à l'instant $t-1$. Ainsi pour décrire un Modèle de Markov, nous devons disposer des probabilités $P(\omega_j|\omega_i) = a_{ij}$, d'obtenir l'événement ω_j à l'instant t connaissant l'état ω_i à $t-1$. De plus nous devons connaître les probabilités $P(\omega_i(0)) = \pi_i$ correspondant à la probabilité que l'état initial du système soit ω_i. Un Modèle de Markov ainsi défini permet de connaître la probabilité $P(\omega^T)$ de n'importe quelle séquence d'état et ainsi de savoir si la séquence donnée d'événements correspond bien au signal, mot, ou objet que nous avons modélisés.

L'état ω_i d'un système à modéliser n'est pas toujours observable. En effet, il est courant qu'un signal ou une information à modéliser dépende d'un certain nombre d'états non directement observables. Ainsi une voix est un signal sonore observable qui dépend d'un certain nombre d'états cachés (position de la langue, vitesse de vibration des cordes vocales, forme de la cavité nasale...). Le Modèle de Markov Caché est une généralisation du Modèle de Markov qui permet de tenir compte des états cachés du système à modéliser.

2.2.5.2 Architecture d'un Modèle de Markov Caché

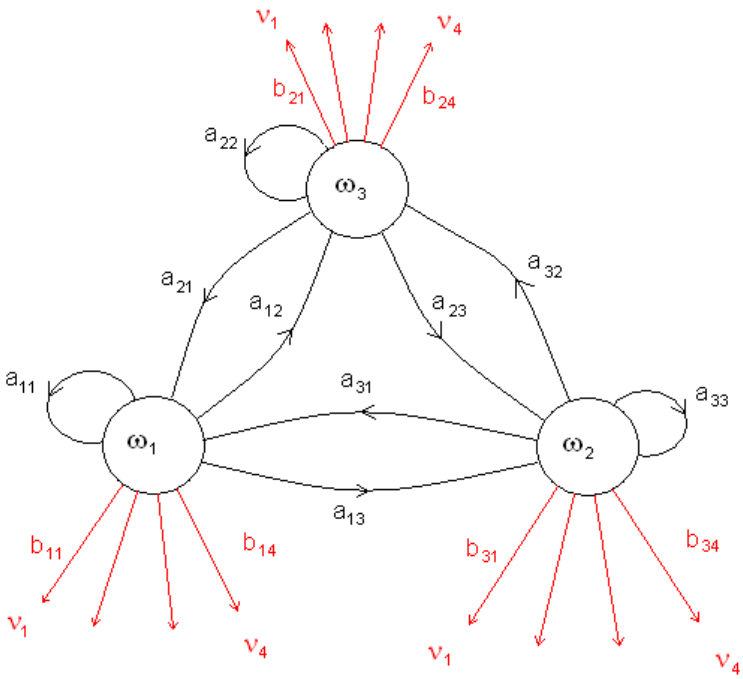


FIGURE 2.4 – Représentation d'un Modèle de Markov Caché ou Automate probabiliste à nombre d'états finis. Les nœuds représentent les états cachés que peut prendre le système à un instant t. Les probabilités de transition a_{ij} sont représentées par les liens entre ces nœuds. Les probabilités b_{jk} des états observables ν_k dépendent directement des états cachés ω_j et sont représentées par les flèches dirigées vers l'extérieur

Comme pour un Modèle de Markov basique, nous continuons de supposer que le système à l'instant t ne peut se trouver que dans un seul état $\omega(t)$, cependant nous ajoutons l'hypothèse que cet état $\omega(t)$ n'est pas observable (caché) et que pour chaque état $\omega(t)$ il y a une probabilité donnée d'émettre un état visible $\nu_k(t)$ que nous pouvons observer. Ainsi la probabilité b_{jk} d'observer l'état ν_k à l'instant t dépend uniquement de l'état caché $\omega(t)$ du système $b_{jk} = P(\nu_k|\omega_j)$. Comme pour le Modèle

de Markov nous définissons une séquence particulière d'états visibles d'un HMM par $V^T = \{\nu(1), \nu(2), \ldots, \nu(t), \ldots, \nu(T)\}$.

Pour résumer un Modèle de Markov Caché est défini par $\lambda = \{W, V, A, B, \Pi\}$

- $W = \{\omega_1, \ldots, \omega_N\}$ L'ensemble des N états cachés possibles
- $V = \{\nu_1, \ldots, \nu_L\}$ L'ensemble des L états visibles possibles
- $A = \{a_{ij}\}_{i,j=1\ldots N}$ Les probabilité de transition d'un état $\omega_i(t-1)$ vers l'état $\omega_j(t)$ ($a_{ij} = P(\omega_j|\omega_i)$ et $\sum_{j=1}^{N} a_{ij} = 1$).
- $B = \{b_{jk}\}_{j=1\ldots N,k=1\ldots L}$ Les probabilités d'observer l'état ν_k si le système est dans l'état caché ω_j ($b_{jk} = P(\nu_k|\omega_j)$ et $\sum_{k=1}^{L} b_{jk} = 1$).
- $\Pi = \{\pi_1, \ldots, \pi_N\}$ Les probabilités que l'état initial du système soit ω_i ($\pi_i = P(w_i(0))$)

La structure d'un Modèle de Markov Caché étant à présent définie, nous allons nous focaliser sur les trois principaux problèmes posés par un tel système :

Le problème d'évaluation Supposons que nous avons un HMM complètement défini dont nous connaissons l'ensemble des paramètres $\lambda = \{W, V, A, B, \Pi\}$. Quelle est la probabilité qu'une séquence d'états visibles V^T soit générée ?

Le problème de décodage Supposons comme pour le problème précédent que nous disposons d'un HMM complet ainsi que d'une séquence d'observation V^T. Quelle est la séquence d'états cachés ω^T la plus probable ayant conduit à ces observations ?

Le problème d'apprentissage Supposons à présent que nous disposions seulement de la structure du HMM c'est à dire son nombre d'états cachés et visibles. Comment déterminer les probabilités a_{ij} et b_{jk} à partir d'un ensemble d'observations V^T ?

2.2.5.3 Evaluation d'un HMM

La probabilité qu'un Modèle de Markov Caché produise une séquence V^T est :

$$P(V^T) = \sum_{r=1}^{r_{max}} P(V^T|\omega_r^T)\, P(\omega_r^T) \tag{2.35}$$

r indexe une séquence particulière ω^T, ainsi, si N est le nombre d'états cachés, le nombre de séquences ω^T possibles est $r_{max} = N^T$. La probabilité d'obtenir une séquence d'états visibles donnée est la somme des probabilités d'obtenir cette séquence pour toutes les combinaisons d'états cachés ω^T possibles pondérés par la probabilité d'obtenir une telle combinaison. Cette probabilité peut être calculée directement à partir des probabilités a_{ij} et b_{jk} et des conditions initiales π_i. La complexité d'un tel calcul étant en $O(N^T T)$, il devient donc très rapidement impossible à effectuer en pratique. Il existe cependant un algorithme de calcul en $O(N^2 T)$:

Posons $\alpha_j(t) = P(V^t, \omega_j(t))$, la probabilité d'avoir t états visibles donnés et l'état caché $\omega_j(t)$ à l'instant t. Nous pouvons alors définir une relation de récurrence pour le calcul de $\alpha_j(t)$:

$$\alpha_j(t) = b_{jk} \sum_{i=1}^{N} \alpha_i(t-1)\, a_{ij} \tag{2.36}$$

$$\tag{2.37}$$

Alors :

$$P(V^t) = \sum_{j=1}^{N} \alpha_j(t) \tag{2.38}$$

Si nous connaissons l'état du système à $t = 0$ on peut alors écrire un algorithme en $O(N^2T)$ permettant de calculer $P(V^T)$:

Algorithm 2 HMM Forward

1: initialisation : a_{ij},b_{jk},V^T,$\alpha_j(0)$
2: **for** $t = 1$ to T **do**
3: $\quad \alpha_j(t) = b_{jk} \sum_{i=1}^{N} \alpha_i(t-1)\, a_{ij}$
4: **end for**
5: **return** $P(V^T) = \sum_{j=1}^{N} \alpha_j(T)$

2.2.5.4 Décodage d'un HMM

Le décodage consiste à trouver ω_r^* tel que $P(\omega_r^*|V^T)$ soit maximum. Comme pour le problème précédent, ceci peut être résolu en énumérant toutes les séquences ω_r possibles, mais la complexité d'une telle méthode est en $O(N^T T)$. D'après, le théorème de Bayes, il est possible de reformuler ce problème ainsi :

$$P(\omega_r|V^T) = \frac{P(V^T|\omega_r)P(\omega_r)}{P(V^T)} \tag{2.39}$$

La probabilité $P(V^T)$ se basant uniquement sur une séquence d'observations fixées, elle ne dépend donc pas de la séquence d'états cachés ω_r. On peut donc formuler le problème de maximisation comme suit :

$$\omega_r^* = \arg\max_{\omega_r} P(V^T|\omega_r)P(\omega_r) \tag{2.40}$$

Par un résonnement récursif similaire à celui pour l'évaluation, l'algorithme de Viterbi [131] permet alors d'effectuer ce calcul avec une complexité en $O(N^2T)$.

2.2.5.5 Méthode d'apprentissage

Le but de l'apprentissage d'un HMM est de déterminer les probabilités a_{ij} et b_{jk} qui permettent de représenter au mieux l'ensemble des exemples d'apprentissage, *i.e*, maximiser la vraisemblance d'obtenir l'ensemble des échantillons exemples. Bien qu'il n'existe pas de méthode permettant d'obtenir les paramètres a_{ij} et b_{jk} conduisant à une représentation optimale de l'ensemble de référence, de nombreuses techniques permettent d'obtenir une bonne solution à ce problème. La technique la plus couramment utilisée est une extension de l'algorithme EM et est connue sous le nom d'algorithme de Baum-Welch [51], ou bien encore d'algorithme 'Forward/Backward'.

Les paramètres $\hat{\mathsf{a}}_{ij}$ et $\hat{\mathsf{b}}_{jk}$ sont estimés itérativement en utilisant leur estimation à l'itération précédente et l'ensemble des observation d'entraînement. Si on pose, $\gamma_{ij}(t)$ la probabilité que le modèle soit dans l'état caché ω_i à l'instant $t-1$ et ω_j à l'instant t connaissant une séquence d'états visibles V^T :

$$\gamma_{ij}(t) = P(\omega_i(t-1), \omega_j(t)|V^T) \tag{2.41}$$

Nous pouvons alors aisément démontrer que :

$$\gamma_{ij}(t) = \frac{\alpha_i(t-1)a_{ij}b_{jk}\beta_j(t)}{P(V^T)} \tag{2.42}$$

Avec $\beta_i(t) = P(V^{t+1,T}|\omega_i(t))$ la probabilité d'avoir les états visibles $V^{t+1,T}$ donnés entre les instants $t+1$ et T et l'état caché $\omega_i(t)$ à l'instant t. $\beta_i(t) = P(V^{t+1,T}|\omega_i(t))$ se calcule de manière similaire à $\alpha_j(t)$ par une simple relation de récurrence :

$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1)\, a_{ij}\, b_{jk} \tag{2.43}$$

Alors :

$$\hat{\mathsf{a}}_{ij} = \frac{Esperance\ du\ nombre\ de\ transitions\ de\ l'evenement\ \omega_i\ vers\ \omega_j}{Esperance\ du\ nombre\ d'evenements\ \omega_i} \tag{2.44}$$

$$= \hat{\mathsf{a}}_{ij} = \frac{\sum_{t=1}^{T}\gamma_{ij}(t)}{\sum_{t=1}^{T}\sum_{k=1}^{N}\gamma_{ik}(t)} \tag{2.45}$$

$$\hat{\mathsf{b}}_{jk} = \frac{Esperance\ du\ nombre\ d'evenements\ \omega_i\ conduisant\ a\ l'observation\ de\ \nu_k}{Esperance\ du\ nombre\ d'evenements\ \omega_i}$$

$$= \hat{\mathsf{b}}_{jk} = \frac{\sum_{t=1,\nu(t)=\nu_k}^{T}\gamma_{ij}(t)}{\sum_{t=1}^{T}\sum_{k=1}^{N}\gamma_{ik}(t)} \tag{2.46}$$

Ainsi, à chaque itération, les paramètres a_{ij} et b_{jk} vont évoluer de façon à rendre plus probable l'obtention de l'ensemble des observations d'entraînement par le HMM.

2.2.5.6 HMM appliqué à l'analyse d'images

Les Modèles de Markov Cachés sont des systèmes monodimensionnels initialement utilisés pour le traitement du son et plus particulièrement, la reconnaissance de paroles. Nous nous proposons dans cette section de décrire brièvement les approches utilisées pour adapter les HMM aux données bidimensionnelles que sont les images.

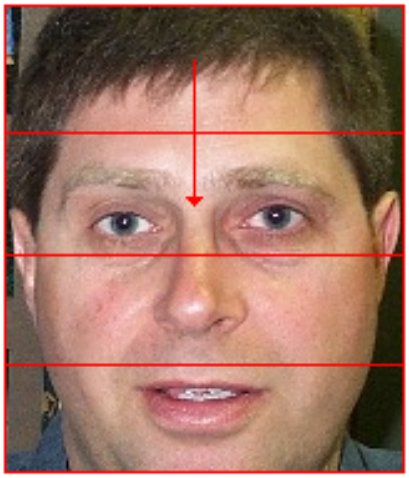


FIGURE 2.5 – Image de visage découpée en bandes horizontales afin de pouvoir être modélisée par un HMM.

Les premières solutions ont consisté à décrire une image comme une succession de sous-régions pouvant éventuellement se recouvrir. Dans [106] des images de visages sont divisées en bandes horizontales du haut vers le bas (figure : 2.5). Les états observables sont les descripteurs de ces sous-régions (front, yeux, nez, bouche). Le HMM décrit ainsi la probabilité d'apparition d'une sous-région (bande verticale) en fonction de la sous-région qui la précède. Ainsi un visage est reconnu si la succession des différentes bandes verticales est suffisamment probable (les yeux devraient être suivis du nez et ainsi de suite).

Un tel modèle ne tient cependant pas compte de l'aspect bidimensionnel des images, et donc de la corrélation entre plusieurs sous-régions voisines. Les Modèles de Markov Cachés bidimensionnels (2D-HMM) permettent d'en tenir compte et de modéliser la probabilité d'apparition d'une sous-région donnée en fonctions des sous-régions voisines. Cependant, la complexité des algorithmes de Baum-Welsh et Viterbi pour un 2D-HMM est exponentielle, ce qui rend cette méthode difficilement utilisable en pratique.

La méthode des Pseudo 2D-HMM (P2D-HMM) a été introduite pour palier à la complexité des 2D-HMM, tout en tenant compte de l'aspect bidimensionnel des images. Elle a été introduite par Agazzi *et al* dans le cadre de la reconnaissance de caractère [1, 57]. Elle utilise la notion de super-états. Ces derniers forment une séquence verticale de bandes d'images. Chaque super-état contient un HMM monodimensionnel permettant de modéliser une bande horizontale de l'image (figure :

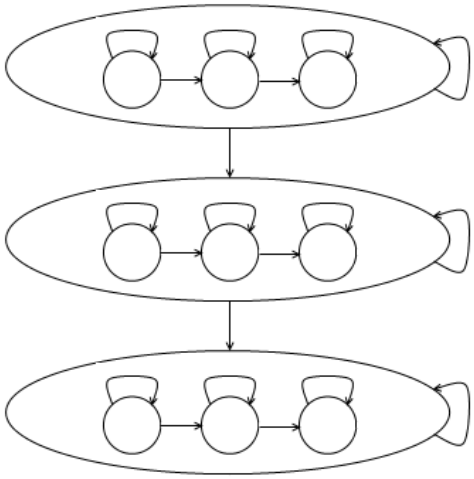


FIGURE 2.6 – Illustration d'un Pseudo 2D-HMM, l'image est modélisée verticalement par des super-états, eux même formés par un HMM classique permettant la modélisation horizontale d'une bande d'image.

2.6).

Dans [92] la méthode des 2D-HMM est approximée par une combinaison de HMM monodimensionnels verticaux et horizontaux. La méthode est appelée Turbo HMM (T-HMM) car elle utilise une technique de décodage directement empruntée aux turbo codes. Dans ces travaux, le T-HMM est utilisé afin de modéliser des visages. Ces visages sont divisés en sous-régions pouvant subir des déformations apprises par le T-HMM selon le principe des EGM (Elastic Graph Matching) [58]. Comme pour les AAM, cette méthode de modélisation tient compte de l'aspect élastique des visages et s'avère très efficace pour les représenter, donnant ainsi des résultats intéressants en reconnaissance de visages.

2.3 Méthodes discriminatives

Les systèmes de détection d'objets les plus performants sont tous basés sur des méthodes de classification discriminatives. Nous nous proposons dans cette section de présenter les méthodes de classification discriminatives les plus courantes ainsi que la façon dont elles sont utilisées en détection d'objets.

Contrairement aux méthodes génératives qui cherchent à modéliser une classe donnée à partir d'un ensemble d'exemples représentatifs, les méthodes discriminatives tentent de modéliser finement les frontières entre différentes classes et exigent à la fois des exemples de la classe à détecter, mais aussi des contre-exemples (classe 'non objet'). L'ensemble de la classe 'non objet' ne pouvant en pratique pas être représentée par un ensemble d'échantillons exemples, seuls les échantillons proches de la frontière à modéliser sont utilisés. Nous commencerons dans cette section par décrire la méthode de 'BootStrapping' utilisée dans la quasi totalité des systèmes

de détection afin de sélectionner les échantillons de la classe 'non objet'. Nous décrirons ensuite une des premières méthodes discriminatives utilisées, *i.e*, L'Analyse Discriminante Linéaire (LDA Linear Discriminant Analysis) qui modélise la frontière entre deux classes par un hyperplan. Finalement nous exposerons les trois méthodes les plus utilisées aujourd'hui, à savoir, AdaBoost, les Séparateurs à Vastes Marges (SVM) et les Réseaux de Neurones (ANN Artificials Neural Networks).

2.3.1 BootStrapping

La méthode de BootStrapping dont le nom désigne aussi une méthode dévaluation des systèmes de classification, a été introduite dans les systèmes de détection par Sung et Poggio dans [116]. Il est en pratique impossible de réunir un ensemble d'échantillons suffisent pour être représentatifs de la classe 'non objet'. En effet chaque imagette de n'importe quelle dimension, et à n'importe quelle position dans une image est un exemple valide pour la classe 'non objet'. La méthode de Sung et Poggio consiste à se focaliser sur les exemples proches de la frontière avec la classe 'objet' en accumulant itérativement les échantillons mal classés par le classifieur discriminatif :

1. Réunir un petit nombre d'échantillons exemples de la classe 'non objet' et un ensemble d'échantillons représentatif de la classe 'objet'
2. Entraîner le classifieur discriminatif à partir de la base d'échantillons exemples courante.
3. Utiliser le système de détection d'objets sur une base d'images ne contenant pas la classe à détecter. Ajouter les fausses détections à la classe 'non objet'.
4. Retourner à l'étape 2 jusqu'à accumulation d'un nombre suffisant d'exemples

A chaque itération, la base d'échantillons de la classe 'non objet' est agrandie par les 'non objet' précédemment mal classés, ce qui permet au classifieurs de corriger les erreurs précédentes. Cette Méthode est utilisée dans la majorité des systèmes de détection, avec parfois, quelques variations. Ainsi dans [39] un BootStrapping est utilisé afin de collecter des images de 'non visage', avec la particularité que le seuil de détection permettant de déterminer si une détection est une fausse alarme diminue avec le nombre d'itérations. Ainsi le système se concentre sur les erreurs les plus significatives lorsque ce dernier commet encore un grand nombre d'erreurs pour ensuite tenir compte des erreurs moins significatives lorsque le système devient plus performant.

2.3.2 Analyse Discriminante Linéaire

L'Analyse Discriminante Linéaire (LDA pour Linear Discriminant Analysis) a été introduite par Fisher [34] en 1936 et a été généralisée plus tard sous le nom d'Analyse Discriminante de Fisher. Le but de la LDA est de déterminer les directions dans lesquelles les données à classer sont le plus aisément séparables. Tout comme la PCA, la LDA est une méthode de projection linéaire dans un sous-espace. Cependant contrairement à la PCA qui détermine le sous-espace le plus adapté à la représentation

des données, la LDA détermine le sous-espace qui permet la meilleure séparation des données à classer (figure : 2.7).

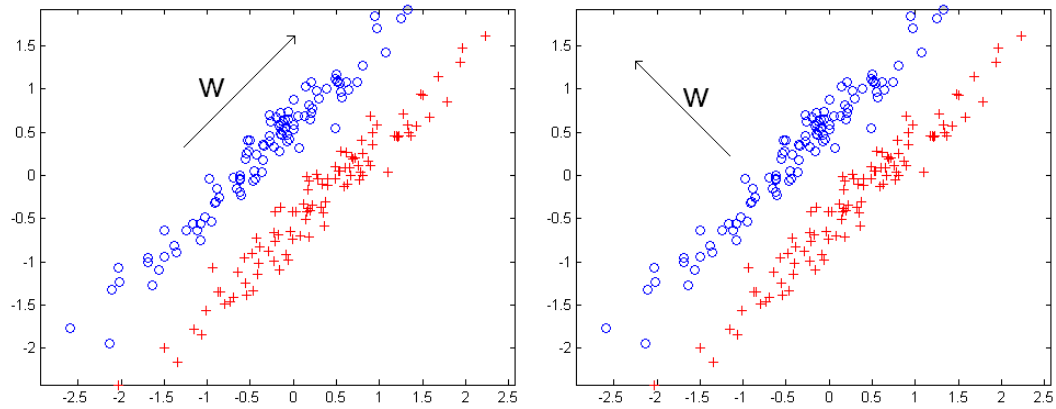


FIGURE 2.7 – Projection du même ensemble de points sur deux lignes différentes dans la direction **w**. La figure sur la gauche montre la direction du premier vecteur propre permettant de représenter l'ensemble des points par la méthode de la PCA. A droite la direction donnée par la LDA permettant la meilleure séparation des échantillons.

Soit un ensemble d'échantillons $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ représentatifs des données à classer dans c classes distinctes $\{D_1, \ldots, D_c\}$. Contrairement à la méthode de la PCA, la dimension du sous-espace de projection est fixée par le nombre de classes à discriminer. Ainsi la dimension de l'espace de projection est égale au nombre de classes à discriminer moins un $(c-1)$. L'espace de projection W est déterminé en maximisant le critère de Fisher [34, 8] :

$$J(W) = \frac{variance\ inter-classe}{variance\ intra-classe} = \frac{\left|W^T \Sigma_B W\right|}{\left|W^T \Sigma_W W\right|} \tag{2.47}$$

Avec Σ_W la somme des matrices de covariance de chaque classe D_i comprenant N_i échantillons :

$$\Sigma_W = \sum_{i=1}^{c} \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m_i}) (\mathbf{x} - \mathbf{m_i})^T \tag{2.48}$$

$$\mathbf{m_i} = \frac{1}{N_i} \sum_{\mathbf{x} \in D_i} \mathbf{x} \tag{2.49}$$

Et Σ_B la matrice de covariance des moyennes de chaque classe pondéré par leur nombre d'échantillons :

$$\Sigma_B = \frac{1}{N}\sum_{i=1}^{c} N_i \left(\mathbf{m_i} - \mathbf{m}\right)\left(\mathbf{m_i} - \mathbf{m}\right)^T \tag{2.50}$$

$$\mathbf{m} = \frac{1}{N}\sum_{\mathbf{x}\in D} \mathbf{x} \tag{2.51}$$

Ainsi, le critère de Fisher est maximum lorsque la variance des échantillons de chaque classe est minimum et que la variance des moyennes est maximum, *i.e*, ce critère maximise la distance entre chaque classe tout en minimisant 'l'espace' qu'elles occupent.

La solution à ce problème de maximisation est obtenue en calculant les vecteurs propres $\mathbf{v_i}$ de la matrice $\Sigma_W^{-1}\Sigma_B$. En effet, les colonnes de la matrice W qui maximisent J correspondent aux vecteurs propres $\mathbf{v_i}$.

Le vecteur propre avec la plus grande valeur propre est connu comme le discriminant de Fisher. La projection sur cet axe est donc une mesure permettant de déterminer à quelle classe appartient un échantillon donné. Cette méthode permet donc en fixant un seuil, de déterminer un hyperplan frontière entre les classes que nous souhaitons séparer. Cependant un simple hyperplan est généralement insuffisant pour convenablement séparer les images d'un 'objet' à détecter du 'reste du monde' [114]. En détection, la méthode est plus souvent utilisée afin de déterminer un sous-espace de projection limitant la dimension des données à classer. Dans [130] Yang *et al* utilisent la LDA afin d'effectuer une détection de visages. Les classes 'visage' et 'non visage' sont divisées en c sous classes plus aisément linéairement séparables. La LDA est ensuite utilisée afin de déterminer un espace de projection de dimension $c-1$ qui permet une meilleure séparation des échantillons. En reconnaissance de visages, la projection dans un sous-espace déterminé par la LDA a montré des résultats supérieurs à l'utilisation de la PCA dans la très grande majorité des cas [73].

L'Analyse Discriminante Linéaire a donc pour avantage la simplicité et la linéarité. Cependant, en détection et en reconnaissance d'objets, les classes ne sont généralement pas linéairement séparables et la LDA s'avère insuffisante pour effectuer une classification efficace. Les Séparateurs à Vaste Marge, que nous nous proposons de décrire ci-dessous permettent de modéliser des frontières plus complexes et sont souvent utilisés dans les problèmes de reconnaissance de formes.

2.3.3 Séparateurs à Vastes Marges (SVM)

La méthode des Séparateurs à Vastes Marges [10, 121, 22, 15] est une technique de classification très générale permettant de déterminer une frontière séparant deux classes. Cette frontière est définie par le principe de la minimisation structurelle du risque formulée par Vapnik [120]. Soit $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ un ensemble de N échantillons exemples labellisés par $z_i = \{1, -1\}$ en fonction de la classe correspondante de l'échantillon $\mathbf{x_i}$. Le but de cette méthode est de trouver un hyperplan de séparation qui maximise la marge entre les échantillons les plus proches de chaque classe (figure : 2.8).

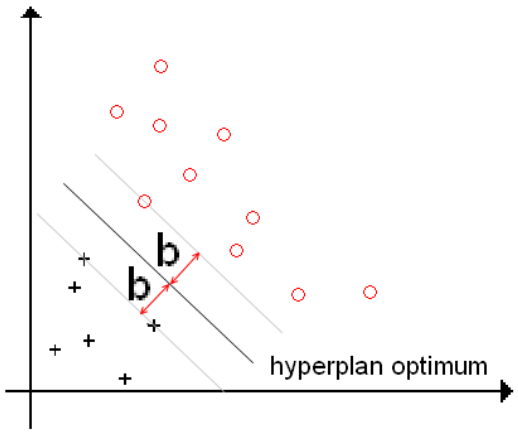


FIGURE 2.8 – Entraîner un SVM consiste à trouver l'hyperplan qui maximise la distance avec les échantillons les plus proches de chaque classe. Les vecteurs supports correspondent aux échantillons à la distance b de l'hyperplan de séparation des deux classes. Dans le cas ci-dessus, il y a trois vecteurs supports

Soit :

$$\mathbf{a} = \begin{bmatrix} \omega_0 \\ \boldsymbol{\omega} \end{bmatrix} \quad \mathbf{y_i} = \begin{bmatrix} 1 \\ \mathbf{x_i} \end{bmatrix} \tag{2.52}$$

Alors la distance d_i entre un échantillon $\mathbf{x_i}$ et l'hyperplan séparateur défini par le vecteur $\mathbf{a}$ s'écrit :

$$d_i = \frac{z_i \mathbf{a}^T \mathbf{y}_i}{\|\boldsymbol{\omega}\|} \tag{2.53}$$

Ainsi, trouver l'hyperplan qui maximise la distance aux échantillons les plus proches de chaque classe, revient à trouver le vecteur $\mathbf{a}$ maximisant b avec $\forall\, i,\ d_i \geq b$. L'hyperplan de séparation étant inchangé par un changement d'échelle de $\boldsymbol{\omega}$, on peut définir que $b\,\|\omega\| = 1$. Le problème revient alors à minimiser $\|\boldsymbol{\omega}\|$ sous la contrainte suivante :

$$z_i \mathbf{a}^T \mathbf{y}_i \geq 1 \tag{2.54}$$

L'utilisation de la méthode du multiplicateur de Lagrange [121] nous permet alors de formuler le problème comme la maximisation du Lagrangien :

$$L(\boldsymbol{\alpha}) = \sum_{I=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j z_i z_j \mathbf{y}_j^T \mathbf{y}_i \tag{2.55}$$

Sous les contraintes :

$$\forall\, i \in [1, N], \alpha_i \geq 0 \tag{2.56}$$

$$\sum_{i=1}^{N} \alpha_i z_i = 0 \tag{2.57}$$

Ce problème de maximisation est en général résolu par des méthodes d'optimisation quadratiques. La direction de l'hyperplan séparateur définie par la méthode des SVM est alors représentée par le vecteur $\boldsymbol{\omega}$:

$$\boldsymbol{\omega} = \sum_{i=1}^{N} \alpha_i z_i \mathbf{x}_i \tag{2.58}$$

Notons que tous les α_i sont nuls exceptés ceux correspondants aux vecteurs supports. Connaissant $\boldsymbol{\omega}$ et les échantillons correspondants aux vecteurs supports, on déduit aisément ω_0 de l'équation (2.53). Finalement les nouveaux échantillons pourront être classifiés en déterminant de quel coté de l'hyperplan séparateur ils se trouvent :

$$z = \mathrm{sgn}(\mathbf{a}^T \mathbf{y}) \tag{2.59}$$

Dans la pratique, il est cependant courant que le problème ne soit pas complètement linéairement séparable. L'équation (2.54) n'a alors plus de solution. Afin de résoudre cette difficulté la contrainte formulée par l'équation (2.54) est remplacée par l'équation suivante :

$$z_i \mathbf{a}^T \mathbf{y}_i \geq 1 - \tau \tag{2.60}$$

$$\tau \geq 0 \tag{2.61}$$

La condition de l'équation 2.57 devient alors :

$$\forall\, i \in [1, N], 0 \leq \alpha_i \leq \tau \tag{2.62}$$

$$\sum_{i=1}^{N} \alpha_i z_i = 0 \tag{2.63}$$

2.3.3.1 SVM non linéaire

En traitement d'image, et plus particulièrement en détection d'objets, les frontières entre les classes sont généralement trop complexes pour pouvoir être modélisées efficacement par un simple hyperplan. Afin de remédier au problème de l'absence de séparateur linéaire, Boser *et al* [10] ont proposé l'utilisation de fonctions noyaux permettant de reconsidérer le problème dans un espace de dimension supérieure. Dans

ce nouvel espace, il est alors probable qu'il existe un séparateur linéaire. L'idée des fonctions noyaux a été introduite par Mercer en 1909 [75] alors que leur utilité dans le contexte des systèmes d'apprentissage a été montrée en 1964 par Aizermann, Bravermann et Rozoener [2].

Les fonctions noyaux permettent de transformer un produit scalaire dans un espace de grande dimension, ce qui est coûteux, en une simple évaluation ponctuelle d'une fonction. Ainsi les équations (2.52) deviennent :

$$\mathbf{a} = \begin{bmatrix} \omega_0 \\ \boldsymbol{\omega} \end{bmatrix} \quad \mathbf{y_i} = \begin{bmatrix} 1 \\ \mathbf{g}(\mathbf{x_i}) \end{bmatrix} \tag{2.64}$$

Le problème n'est alors plus linéaire par rapport à $\mathbf{x}$ mais par rapport à $\mathbf{g}(\mathbf{x})$ sa projection dans un espace de dimension supérieure. Le problème est alors résolu en utilisant la même procédure que pour le cas linéaire, les échantillons $\mathbf{x}$ étant remplacés par leur projection $\mathbf{g}(\mathbf{x})$. Le problème d'optimisation revient alors à maximiser :

$$L(\boldsymbol{\alpha}) = \sum_{I=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j z_i z_j \mathbf{g}(\mathbf{x}_j)^T \mathbf{g}(\mathbf{x}_i) \tag{2.65}$$

Sous les contraintes :

$$\forall\, i \;\in\; [1, N]\,, \alpha_i \geq 0 \tag{2.66}$$

$$\sum_{i=1}^{N} \alpha_i z_i = 0 \tag{2.67}$$

Le problème d'une telle méthode est qu'elle implique un produit scalaire entre vecteurs dans l'espace de redescription, de dimension élevée, ce qui peut s'avérer très coûteux en termes de calculs. Pour résoudre ce problème, on utilise une astuce connue sous le nom de 'Kernel Trick', qui consiste à utiliser une fonction noyau $\mathbf{g}(.)$, qui vérifie :

$$K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{g}(\mathbf{x_i})^T \mathbf{g}(\mathbf{x_j}) \tag{2.68}$$

Ainsi, il n'est plus nécessaire pour résoudre le problème, ni de calculer un produit scalaire dans un espace de grande dimension, ni de connaître explicitement la fonction $\mathbf{g}(.)$, puisque seul nous est nécessaire le résultat du produit scalaire $K(\mathbf{x_i}, \mathbf{x_j})$.

En pratique, même si nous n'avons pas besoin de connaître la fonction $\mathbf{g}(.)$, nous devons nous assurer que pour la fonction $K(\mathbf{x_i}, \mathbf{x_j})$ choisie, il existe toujours une fonction $\mathbf{g}(.)$ vérifiant l'équation (2.68). Ceci est assuré par la conditions de Mercers [121]. Les noyaux usuels respectant ces conditions employés avec les SVM sont :

- Noyaux polinomiaux :

$$K(\mathbf{x_i}, \mathbf{x_j}) = \left(\mathbf{x_i}^T \mathbf{x_j} + 1\right)^n$$

- Noyaux Gaussiens :

$$K(\mathbf{x_i}, \mathbf{x_j}) = e^{-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2}$$

- Noyaux Tangeante Hyperbolique :

$$K(\mathbf{x_i}, \mathbf{x_j}) = \tanh\left(\kappa(\mathbf{x_i}^T \mathbf{x_j}) - \tau\right)$$

Depuis leur développement dans les années 1990, les SVM ont été très largement utilisés dans de très nombreux domaines, y compris en traitement d'image pour des système de détection [107, 84]. Cependant, les résultats les plus remarquables en détection sont généralement basés sur les méthodes de classification que nous allons présenter dans les deux sections suivantes, *i.e*, AdaBoost [122] et les Réseaux de Neurones [39].

2.3.4 Boosting

L'idée du Boosting est de combiner plusieurs classifieurs dit 'faibles' (peu efficaces) afin d'obtenir un classifieur 'fort', *i.e*, performant. En pratique, un classifieur 'faible' doit simplement classer mieux que le hasard. Une méthode simple pour effectuer un Boosting est la suivante (figure : 2.9) :

Considérons que nous disposons de trois classifieurs 'faibles' pour un problème de classification dans deux catégories. Soit $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ l'ensemble des échantillons exemples. Un premier classifieur C_1 est entraîné avec une sélection D_1 de $N_1 = N/3$ échantillons de D. Un second classifieur C_2 est entrainé avec une sélection D_2 de $N_2 = N/3$ échantillons nous apportant le maximum d'information par rapport aux échantillons D_1. Concrètement la moitié seulement des échantillons de D_2 doivent être bien classés par C_1. Finalement le troisième classifieur C_3 est entrainé à partir d'une sélection d'échantillons correspondant aux échantillons de D pour lesquels les classifieurs C_1 et C_2 ne donnent pas la même classification. Ainsi D_3 représente l'ensemble des échantillons mal représentés par la combinaison des classifieurs C_1 et C_2.

Il existe de nombreuses variations sur les méthodes de Boosting, un des algorithmes les plus utilisés s'appelle AdaBoost et permet de combiner les classifieurs faibles jusqu'à obtenir l'erreur de classification désirée pour les échantillons d'entraînement.

2.3.4.1 AdaBoost

AdaBoost (contraction de Adaptative Boosting) est une méthode de Boosting introduite par Freund et Schapire [35]. Les performances et la simplicité de cette technique font qu'elle est utilisée dans un grand nombre de problèmes de classification et notamment pour le détecteur d'objets de Viola-Jones [122]. Comme pour la

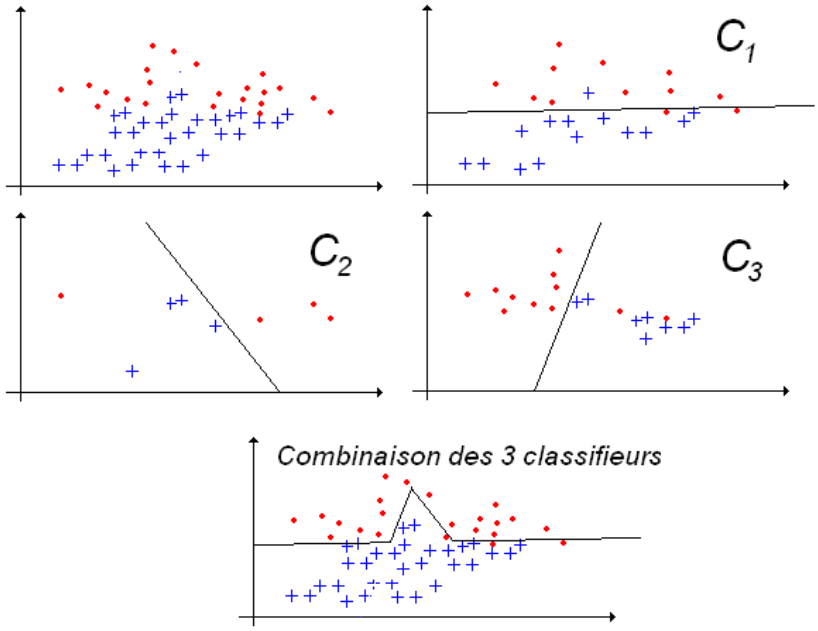


FIGURE 2.9 – Exemple de méthode de Boosting utilisant trois classifieurs linéaires dans un espace de dimension deux. Nous obtenons par la combinaison des trois classifieurs, une frontière et un classement bien plus précis qu'avec un seul classifieur linéaire.

méthode de boosting basique, l'idée de cette méthode est de focaliser les classifieurs 'faibles' sur les exemples les plus difficiles à classer.

Bien qu'il existe des extensions de la méthode AdaBoost pour le cas des problèmes multi-classes [36, 108], nous nous focaliserons ici sur le problème de classification entre deux classes. Dans cette configuration, le classifieur 'fort' est constitué de k_{max} classifieurs 'faibles' binaires successifs $C_k = \{-1, +1\}$. Chaque échantillon exemple $\mathbf{x_i}$ associé a la classe $z_i = \{-1, +1\}$ reçoit pour chaque classifieur 'faible' C_k un poids w_i^k qui détermine sa probabilité d'être sélectionné pour entraîner le classifieur. Chaque échantillon mal classé par le classifieur C_k verra son poids associé augmenter pour le classifieur suivant C_{k+1} et diminuer dans le cas ou il serait bien classé.

$$w_i^{k+1} = \frac{w_i^k}{W_{k+1}} \times e^{-z_i C_k(\mathbf{x_i})\alpha_k} \tag{2.69}$$

$$W_{k+1} = \sum_{i=1}^{N} w_i^{k+1} \tag{2.70}$$

$$\alpha_k = \frac{1}{2} \ln \left(\frac{1 - E_k}{E_k} \right) \tag{2.71}$$

$$E_k = \sum_{z_i \neq C_k(\mathbf{x_i})} w_i^k \tag{2.72}$$

Ainsi, les exemples souvent mal classés voient leur poids augmenter par rapport à ceux souvent bien classés. De plus, chaque classifieur faible se voit associé un poids α_k lié a son erreur de classification. Le score final $s(\mathbf{x})$ donné par l'association de l'ensemble des classifieurs se calcule ainsi :

$$s(\mathbf{x}) = \sum_{k=1}^{k_{max}} \alpha_k C_k(\mathbf{x}) \tag{2.73}$$

Le score $s(\mathbf{x})$ associé à l'exemple $\mathbf{x}$ est la somme pondérée du score des classifieurs $C_k(\mathbf{x})$. Le poids associé à chaque classifieur dépend directement du taux d'erreur E_k du classifieur. Si le classifieur est parfait $E_k = 0$ et α_k tend vers l'infini. Si le classifieur a un taux d'erreur $E_k = 0.5$ (taux de détection du hasard) alors, $\alpha_k = 0$, son poids sera nul dans l'association de classifieurs. Ainsi la méthode d'AdaBoost maximise l'importance des 'meilleurs' classifieurs 'faibles'.

Algorithm 3 AdaBoost

1: p, q nombre d'échantillons associés à la classe $z_i = \{-1, +1\}$
2: $\forall i = 1 \dots N$, $w_i^1 = \frac{1}{p}$ si $z_i = -1$, $w_i^1 = \frac{1}{q}$ sinon
3: **for** $k = 1$ to k_{max} **do**
4: Entraîner le classifieur 'faible' C_k avec les poids w_k^i et les échantillons exemples $\mathbf{x_i}$
5: $E_k = \sum_{z_i \neq C_k(\mathbf{x_i})} w_i^k$
6: $\alpha_k = \frac{1}{2} \ln\left(\frac{1-E_k}{E_k}\right)$
7: $w_i^{k+1} = \frac{w_i^k}{W_{k+1}} \times e^{-z_i C_k(\mathbf{x_i})\alpha_k}$
8: **end for**
9: **return** $s(\mathbf{x}) = \text{sgn}\left(\sum_{k=1}^{k_{max}} \alpha_k C_k(\mathbf{x})\right)$

Cette méthode de Boosting est particulièrement intéressante car nous pouvons choisir le nombre k_{max} de classifieurs de manière à atteindre le taux d'erreur souhaité sur les échantillons exemples. De plus, il est démontré [36] que le taux d'erreur E associé au classifieur 'fort' décroît exponentiellement avec le nombre de classifieurs 'faibles' utilisés. En effet, si on écrit l'erreur de classification du k^{ieme} classifieur 'faible' sous la forme $E_k = \frac{1}{2} - \gamma_k$, la variable γ_k supérieure à 0 représentant l'amélioration du taux d'erreur du classifieur par rapport au hasard. Alors on peut écrire :

$$E = \prod_{k=1}^{k_{max}} \left(1 - 4\gamma_k^2\right)^{\frac{1}{2}} \leq e^{\left(-2\sum_{k=1}^{k_{max}} \gamma_k^2\right)} \tag{2.74}$$

Une valeur trop grande de k_{max}, outre des problèmes de complexité de calcul peut, en théorie conduire à un sur-apprentissage, conduisant à une bonne classification sur les images exemples mais une très mauvaise généralisation du problème dans le cas pratique. Les nombreuses expérimentations sur AdaBoost ont cependant montré que

le sur-apprentissage est en pratique très rare avec cette méthode. Les propriétés de généralisation d'AdaBoost en font donc une méthode de classification très efficace en pratique. Elle est très utilisée pour des problèmes de détection 'd'objets réels', tels que des piétons [64] ou même des formes plus théoriques tels que des croisements de lignes dans une image [70]. Enfin le détecteur d'objets de Viola-Jones [122] est l'un des plus utilisé aujourd'hui et est directement basé sur la méthode de classification AdaBoost.

2.3.4.2 Détecteur d'objet de Viola-Jones

Le détecteur d'objet de Viola-Jones constitue aujourd'hui une référence pour les systèmes de détection d'objets. Il permet d'effectuer une détection robuste et en temps réel des objets dans une image. Pour cela il s'appuie sur trois idées fondamentales.

Les Descripteurs : Ils sont basés sur les fonctions de Haar utilisées par Papageorgiou *et al* [87]. Concrètement, ces descripteurs se divisent en trois types différents consistant en sommer les valeurs des pixels dans les rectangles blancs et soustraire ceux dans les rectangles grisés (figure : 2.10). Les dimensions et la position de ces descripteurs pouvant varier dans l'image de l'objet que l'on souhaite détecter, nous arrivons pour une image d'objet de dimension 24×24 pixels à un nombre de descripteurs possibles de 45396.

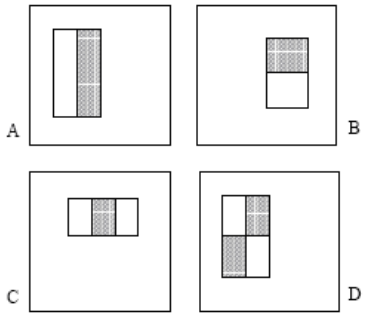


FIGURE 2.10 – Exemples de descripteurs utilisés par le détecteur d'objets de Viola-Jones. La somme des pixels dans les zones grisées est soustraite de la somme des pixels dans les zones blanches. Ces descripteurs se divisent en trois catégories, ceux formés de deux régions (A et B), ceux formés de trois régions comme par exemple C et enfin ceux formés de quatre régions comme D

AdaBoost avec sélection des descripteurs : Le nombre de descripteurs possible pour une image de 24×24 pixels est très grand. Cependant les expérimentations montrent qu'un petit nombre de ces descripteurs est suffisant pour entraîner un classifieur efficace. Afin de sélectionner les meilleurs descripteurs permettant de décrire un objet donné, le système de Viola-Jones utilise un algorithme basé sur la méthode AdaBoost. Les classifieurs faibles C_k sont de simples hyperplans séparateurs utilisant

un seul descripteur f_j, un seuil θ_j et une parité $p_j = \{1, -1\}$ permettant de déterminer la direction de la normale à l'hyperplan séparateur constituant la frontière :

$$C_k(\mathbf{x}) = \text{sgn}(p_j f_j(\mathbf{x}) - \theta_j) \tag{2.75}$$

Dans cette variante de la méthode AdaBoost, chaque classifieur C_k formant le classifieur 'fort' est associé au descripteur f_j permettant de minimiser le taux d'erreur sur l'ensemble des échantillons exemples $\mathbf{x}_i$ pondérés par les poids w_i^k (Algorithme : 4). Ainsi chaque classifieur C_k utilise le descripteur f_j (figure : 2.11) permettant de différencier au mieux les échantillons mal classés par les classifieurs précédents $(C_1 \dots C_{k-1})$.

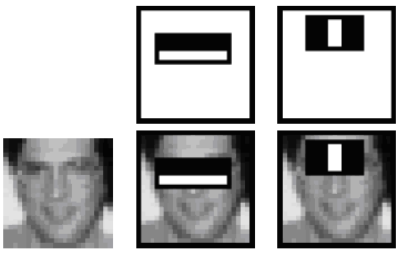


FIGURE 2.11 – Deux premiers/meilleurs descripteurs utilisés pour la détection de visages par l'algorithme de Viola-Jones. Les yeux semblent être la partie du visage permettant la meilleure discrimination par rapport au 'reste du monde'.

Algorithm 4 AdaBoost Viola-Jones

1: p, q nombre d'échantillons associés à la classe $z_i = \{-1, +1\}$
2: $\forall i = 1 \dots N$, $w_i^1 = \frac{1}{p}$ si $z_i = -1$, $w_i^1 = \frac{1}{q}$ sinon
3: **for** $k = 1$ to k_{max} **do**
4: Entraîner l'ensemble des classifieurs h_j^k associé aux descripteurs f_j avec les poids w_k^i et les échantillons exemples $\mathbf{x_i}$
5: Choisir le classifieur $h_j^k = C_k$ avec le plus faible taux d'erreur E_k
6: $E_k = \sum_{z_i \neq C_k(\mathbf{x_i})} w_i^k$
7: $\alpha_k = \frac{1}{2} \ln\left(\frac{1-E_k}{E_k}\right)$
8: $w_i^{k+1} = \frac{w_i^k}{W_{k+1}} \times e^{-z_i C_k(\mathbf{x_i})\alpha_k}$
9: **end for**
10: **return** $s(\mathbf{x}) = \text{sgn}\left(\sum_{k=1}^{k_{max}} \alpha_k C_k(\mathbf{x})\right)$

Cascade de classifieurs : La méthode expliquée ci-dessus permet d'obtenir un classifieur efficace, cependant, le nombre de classifieurs faibles et donc de descripteurs nécessaires pour obtenir un taux de détection acceptable peut être important

(plusieurs centaines pour un détecteur de visages), ce qui ne permet pas une détection en temps réel. Afin de rendre l'algorithme de détection plus rapide, le système de Viola-Jones utilise une cascade de classifieurs (figure : 2.12). Chaque classifieur élimine un certain nombre de 'non objet' tout en sélectionnant pratiquement 100% des objets à détecter. Les premiers classifieurs de la cascade sont basés sur un petit nombre de descripteurs et sont donc très rapides mais peu discriminatifs, puis chaque classifieur successif de la cascade devient de plus en plus complexe, permettant une réduction importante des fausses détections. Ainsi le dernier classifieur qui utilise plusieurs centaines de descripteurs, n'est utilisé que sur un nombre très réduit d'imagettes à tester, limitant au minimum le nombre d'opérations requises pour la détection.

Afin de collecter des échantillons de 'non objet' pertinents, cette méthode utilise une variante de la méthode de BootStrapping adaptée à la cascade de classifieurs. Le premier classifieur est entraîné avec p_1 'non objet' choisies aléatoirement (p_1=10000 pour le détecteur de visage). Le second classifieur est entraîné avec p_2 images de fausses détections acquise par la méthode de BootStrapping appliqué au premier classifieur. Une nouvelle itération permet d'entraîner le classifieur suivant à partir de nouvelles fausses détections effectuées par la cascade formée des deux premiers classifieurs. Ainsi chaque classifieur de la cascade se spécialise sur les erreurs faites par les classifieurs précédents.

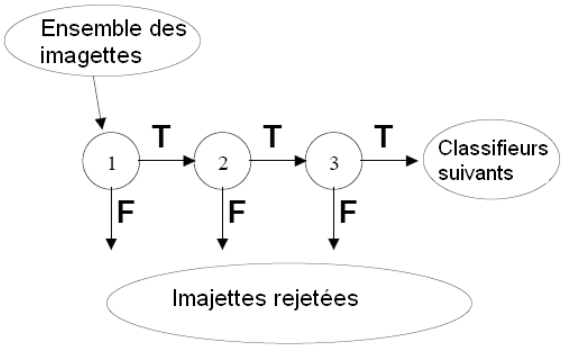


FIGURE 2.12 – Schéma d'une cascade de classifieurs. Chaque imagette présentée au système est classée successivement par chacun des classifieurs. Si un classifieur classe l'imagette dans la catégorie 'non objet' les calculs s'arrêtent et l'imagette est classée dans cette catégorie. Sinon elle est classée dans la catégorie 'objet'. Un système de détection d'objets traitant beaucoup moins d'échantillons 'objet' que 'non objet', ce système permet de grandement diminuer les temps de calculs.

Les trois idées exposées ci-dessus permettent d'obtenir un système de détection rapide et robuste et font de ce système de détection un des plus utilisés actuellement.

2.3.4.3 Méthodes dérivées du détecteur d'objets de Viola-Jones

La méthode de détection de Viola-Jones a permis, grâce à l'utilisation de descripteurs basés sur les fonctions de Haar, de la méthode de classification AdaBoost et d'une cascade de classifieurs, d'obtenir un système de détection d'objets à la fois robuste, rapide et performant. Ces dernières années, de nombreuses méthodes de détections se sont inspirées du détecteur de Viola-Jones et les dernières évolutions aussi bien, en terme de taux de détection que de complexité de calcul sont presque toutes dérivées de cette méthode. Les améliorations apportées à l'algorithme de Viola-Jones portent aussi bien sur la méthode AdaBoost [115, 109, 45, 63], les descripteurs utilisées [65, 126, 110] et la forme de la cascade de classifieurs [12, 45, 127, 125].

AdaBoost : Parmi les améliorations apportées à la méthode AdaBoost, on notera en particulier la méthode du FloatBoost [63] qui consiste à ajouter une phase de vérification de l'utilité de l'ensemble des classifieurs faibles et si possible de supprimer les classifieurs faibles inutiles. Cela permet de diminuer le nombre de classifieurs faibles utilisés et donc d'augmenter la vitesse de calcul, sans pour autant diminuer le taux de détection. Une autre amélioration nommée RealAdaBoost [109] utilisé par Huang *et al* dans [45] consiste à utiliser des classifieurs faibles donnant non plus une réponse binaire $\{0, 1\}$ mais une réponse appartenant à $[0, 1]$ caractérisant ainsi la confiance que l'on peut avoir dans le résultat du classifieur faible. Cette méthode permet de minimiser le nombre de classifieurs faibles comparé à la méthode de Viola-Jones.

Les descripteurs : Les fonctions de Haar utilisées par Viola-Jones permettent de décrire efficacement les objets à détecter de manière extrêmement rapide grâce à l'utilisation de la méthode des images intégrales. Les descripteurs utilisés restent généralement toujours basés sur les fonctions de Haar. La principale évolution est basée sur l'utilisation de la méthode des 'features centrics evaluation' par Shneiderman [110] et par Yan *et al* [126]. Cette méthode consiste à utiliser chaque descripteur, non pas à une position de l'imagette à classifier mais à toutes les positions possibles. Ainsi, un seul descripteur apporte plus d'informations que la méthode classique 'window centric evaluation' ce qui permet une nette réduction du nombre de classifieurs faibles nécessaires ainsi qu'une amélioration des taux de détection.

Cascade de classifieurs : Les évolutions les plus notables sont dues aux progrès fait sur la cascade de classifieurs. Dans la méthode de Viola-Jones, les classifieurs forts de chaque étage de la cascade de classifieurs sont indépendants les uns des autres. Ainsi, dans la méthode de Viola-Jones, chaque étage de la cascade de classifieurs n'utilise pas l'information apporté par les étages précédents. La méthode des 'Boosting Chain Learning' utilisée par Xiao *et al* dans [125], par Bourdev et Brandt dans [12] et par Huang *et al* dans [45] permet d'utiliser le score des classifieurs forts des étages précédents de la cascade pour entrainer le classifieur fort suivant.

Ces méthodes dérivées de Viola-Jones forment aujourd'hui les systèmes de détection parmi les plus rapides et performants. Une alternative à AdaBoost est basée sur les réseaux de neurones qui constituent aujourd'hui grâce à la variété des techniques possibles, une des méthodes de classification les plus performantes, y compris pour les problèmes de détection. La section suivante présentera les principes des réseaux de neurones, et les différents types de réseaux de neurones utilisés pour les problèmes de détection.

2.3.5 Réseaux de Neurones

Les réseaux de neurones constituent aujourd'hui une des techniques de classification les plus populaires et les plus performantes, y compris pour les problèmes de reconnaissance de formes visuelles et de détection. Cette méthode, inspirée par le fonctionnement du cerveau humain consiste à interconnecter par des synapses, des neurones symbolisés par des fonctions mathématiques simples φ, ceci dans le but d'effectuer des tâches de classification complexes. A chaque connexion à un neurone est associée un poids w_i symbolisant l'importance de l'information x_i associée à cette connexion. La réponse y du neurone est alors le résultat de la fonction φ de la somme des signaux d'entrée $\mathbf{x} = (x_1 \dots x_d)^T$ pondérés par les poids $\mathbf{w} = (w_1 \dots w_d)^T$ et un biais additionel b (figure : 2.13). Ainsi, $y = \varphi(\mathbf{w} \cdot \mathbf{x} + b)$.

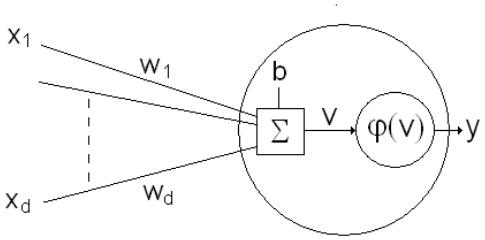


FIGURE 2.13 – Schéma d'un neurone. Les entrées $\mathbf{x}$ sont pondérées par les poids $\mathbf{w}$ et sommées au biais b donnant le résultat v. la réponse du neurone $\mathbf{y}$ est alors donnée par la fonction $\varphi(v)$.

Entraîner un réseau de neurones consiste à trouver l'ensemble des poids $\mathbf{w}$ et le biais b de chaque neurone qui minimise l'erreur E de sortie du réseau de neurone. Dans la très grande majorité des cas, E représente l'erreur quadratique, *i.e*, si $\mathbf{z_i} = \mathbf{g}(\mathbf{x_i})$ est la sortie du réseau de neurones pour le i^{ieme} échantillon exemple et $\mathbf{t_i}$ les valeurs objectifs souhaitées pour l'échantillon, alors :

$$E = \sum_i (z_i - t_i)^2 \tag{2.76}$$

Il existe un très grand nombre d'architectures possibles pour un réseau de neurones. Cette dernière est caractérisée par les fonctions φ associées à chaque neurone, le nombre de neurones du réseau et les connexions entre ces neurones. Les architectures les plus utilisées en reconnaissance de formes et détection d'objets sont présentées dans les sections suivantes.

2.3.6 Perceptron

Inspiré par les travaux sur la théorie cognitive de Friedrich Hayek et Donald Hebb, le Perceptron a été introduit en 1958 par Rosenblatt [97]. Il constitue le modèle de réseau de neurones le plus simple puisqu'il n'est constitué que d'un seul neurone.

Ainsi la sortie du système est la réponse du neurone induite par la somme pondérée par $\mathbf{w}$ des éléments de l'entrée $\mathbf{x}$:

$$net = \mathbf{w} \cdot \mathbf{x} + b \tag{2.77}$$

$$y = \varphi(net) \tag{2.78}$$

La fonction d'activation φ utilisée est une simple fonction heaviside $H(net) = 0$ si $net < 0$ sinon, $H(net) = 1$. Ainsi le perceptron est un simple hyperplan séparateur, l'échantillon $\mathbf{x}$ est classé dans la catégorie '0' ou '1' en fonction de sa position par rapport à l'hyperplan définie par $\mathbf{w}$ et b. L'utilisation d'un algorithme d'optimisation basé sur la minimisation du nombre d'échantillons exemples mal classées [98] permet de traiter tout problème de classification à deux classes. Un simple séparateur linéaire est cependant généralement insuffisant pour traiter des problèmes de classification complexes tels que la détection d'objets dans une image. De plus, Minsky et Papert [76] ont montré que le perceptron est inefficace pour des problèmes simples tels que celui du OU-exclusif où il est impossible de séparer par une droite les points $(0,0)$ et $(1,1)$ des points $(0,1)$ et $(1,0)$ (figure : 2.14). Le Perceptron Multicouche utilise plusieurs neurones interconnectés afin de pouvoir modéliser des frontières plus complexes permettant de traiter des problèmes de classification plus difficiles ainsi que les problèmes de plus de deux classes.

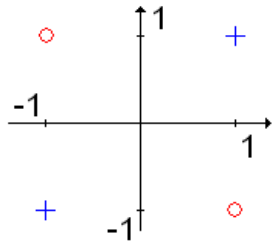


FIGURE 2.14 – Problème de classification OU-Exclusif. Malgré la simplicité du problème qui ne comporte que 2×2 échantillons bidimensionnels à classer, ce problème n'est pas linéairement séparable.

2.3.7 Perceptron Multicouche

Le Perceptron Multicouche (MLP pour Multilayer Perceptron) est la forme de réseau de neurones la plus couramment utilisée. Un tel réseau est constitué d'un minimum de trois couches de neurones. Les neurones d'une couche donnée ne sont connectés qu'aux neurones de la couche suivante (figure : 2.15), ainsi l'activation des neurones est propagée à travers les différentes couches, de l'entrée vers la sortie. Disposant d'une méthode d'apprentissage simple et efficace [98], Le MLP est en plus capable d'approximer n'importe quelle fonction décision et donc n'importe quelle forme de frontière, à condition de disposer de suffisamment de neurones cachés

(neurones non situés sur la première ou la dernière couche du réseau). En détection d'objets, le MLP dispose généralement d'un seul neurone de sortie dont la valeur de sortie $z = g(\mathbf{x})$ permet de définir si un échantillon $\mathbf{x}$ appartient ou pas à la catégorie 'objet'. Un tel système est entraîné de façon à ce que $z = 1$ si l'échantillon d'entrée appartient à la catégorie 'objet' et $z = -1$ dans le cas contraire, la frontière séparant les deux catégorie correspond aux valeurs de $\mathbf{x}$ telles que $z = g(\mathbf{x}) = 0$ (figure : 2.22). Enfin, la souplesse du MLP permet aussi de traiter le cas multi-catégories en faisant correspondre une catégorie à chaque neurone de sortie.

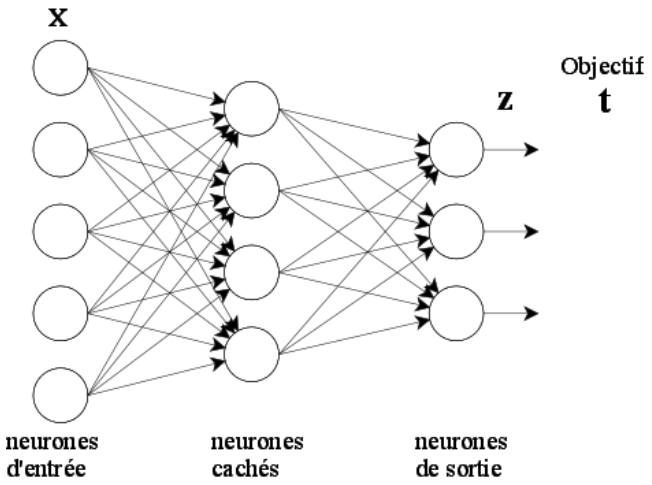


FIGURE 2.15 – Perceptron Multicouche. Il est constitué d'au minimum trois couches de neurones. Une couche d'entrée contenant le vecteur $\mathbf{x}$ à classer, une ou plusieurs couches cachées, et une couche de sortie dont le vecteur résultat $\mathbf{z} = (z_1, \ldots, z_c)$ permet de classer l'échantillon correspondant au vecteur $\mathbf{x}$. Dans cet exemple le MLP est dit entièrement connecté car chaque neurone est relié à l'ensemble des neurones de la couche précédente.

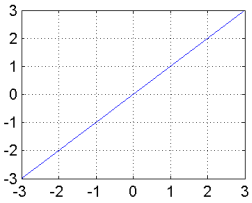
En ce qui concerne les fonctions d'activation φ des neurones du MLP, la seule contrainte théorique est induite par l'algorithme d'apprentissage de Rétropropagation (ou Backpropagation) utilisé. En effet ce dernier est basé sur la méthode de descente par gradient et impose donc que les fonctions d'activation $\varphi(\mathbf{x}, \mathbf{w}, b)$ soient continues et dérivables par rapport à $\mathbf{w}$ et b. En pratique, on retrouve le plus souvent les trois fonctions suivantes (figure : 2.16) :

$$\varphi(net) = net \qquad \text{linéaire} \tag{2.79}$$

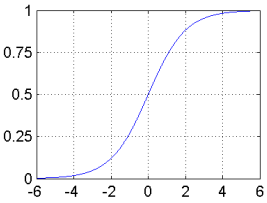
$$\varphi(net) = \frac{1}{1 + e^{-net}} \qquad \text{sigmoid} \tag{2.80}$$

$$\varphi(net) = tanh(net) = \frac{1 - e^{-net}}{1 + e^{-net}} \qquad \text{tangeante hyperbolique} \tag{2.81}$$

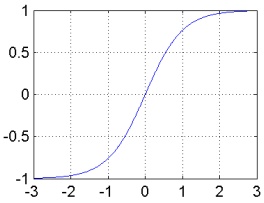
$$net = \mathbf{w} \cdot \mathbf{x} + b \tag{2.82}$$



(a) Linéaire



(b) Sigmoid



(c) Tangeante Hyperbolique

FIGURE 2.16 – fonctions d'activation utilisées dans les réseaux de neurones

2.3.7.1 Perceptron Multicouche appliqué à la détection

Le MLP est très largement utilisé en traitement d'image, pour des tâches aussi variées que l'interpolation [46], la reconnaissance de visages [83, 31, 32], la reconnaissance de mots [16] ou encore la classification d'images [74]. En ce qui concerne le problème de la détection, ont peut citer la méthode mise au point par Sung et Poggio [116] appliquée à la détection de visages. Le système commence par déterminer à l'aide d'un algorithme de clustering dérivé de la méthode du k-Mean, douze clusters caractérisés par six images de visages et six de 'non visage' caractéristiques de la base d'exemples. Pour chaque imagette, ce système extrait deux distances à chacun des douze clusters constituant ainsi pour chaque entrée un vecteur de dimension 24 (figure : 2.17). La première distance D_1 est la distance de Mahalanobis dans un sous-espace de dimension 75 déterminé par la PCA (Σ_i étant la matrice de covariance des exemples associés au i^{ieme} cluster) :

$$D_1(\mathbf{x}, \boldsymbol{\mu_i}) = \frac{1}{2}\left(d\ln(2\pi) + \ln|\Sigma_i| + (\mathbf{x} - \boldsymbol{\mu_i})^T\Sigma_i^{-1}(\mathbf{x} - \boldsymbol{\mu_i})\right) \tag{2.83}$$

La seconde distance D_2 est la distance euclidienne entre une imagette représentée par le vecteur $\mathbf{x}$ et sa projection dans le sous-espace d'un cluster.

Les vecteurs de dimension 24 ainsi obtenus sont ensuite classés à l'aide d'un perceptron à trois couches entièrement connecté possédant 24 neurones 'cachés' et un neurone de sortie. La fonction d'activation choisie est la fonction Sigmoid. Le MLP est entraîné de façon à renvoyer 1 si l'échantillon d'entrée est un visage et 0 sinon.

Une autre méthode utilisant avec succès un Perceptron Multicouche en détection est celle de Rowley *et al* [42]. Cette méthode a la particularité d'utiliser un Perceptron Multicouche avec comme entrée un vecteur $\mathbf{x}$ correspondant à une image de 20×20 pixels en Niveaux de Gris, prétraitée de façon à corriger les problèmes de luminosité. Un Perceptron Multicouche entièrement connecté sur une image en Niveaux de Gris engendre un très grand nombre de connexions et donc de coefficients à entraîner. C'est pourquoi dans cette méthode le MLP utilisé possède trois couches mais n'est pas entièrement connecté. Nous retrouvons 4 neurones cachés reliés à 4 sous-régions de 10×10 pixels, 16 reliés à 16 sous-régions de 5×5 pixels et enfin 6 neurones reliés à 6 sous-régions de 20×5 pixels (figure : 2.18).

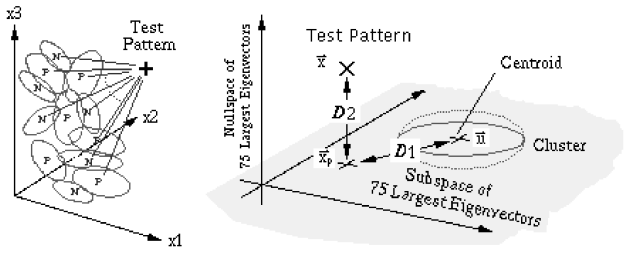


FIGURE 2.17 – Distances utilisées pour le détecteur de visages de Sung et Poggio, à gauche, la distance de Mahalanobis entre un vecteur $\mathbf{x}$ et le centroïde de chacun des 12 clusters, à droite, la distance entre le vecteur $\mathbf{x}$ et le sous-espace de dimension 75 d'un cluster.

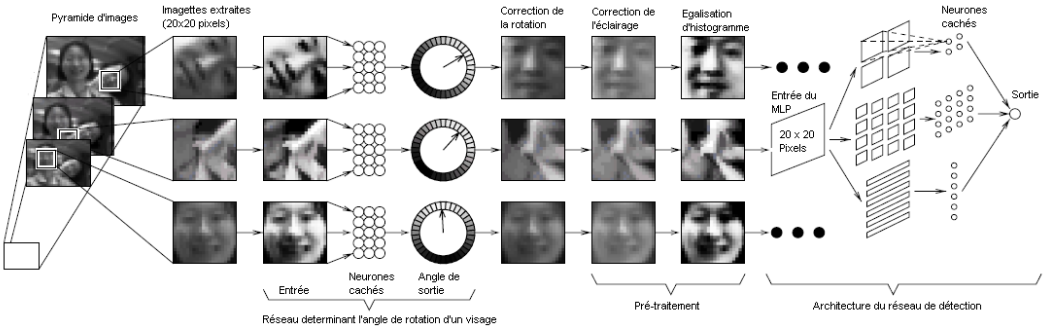


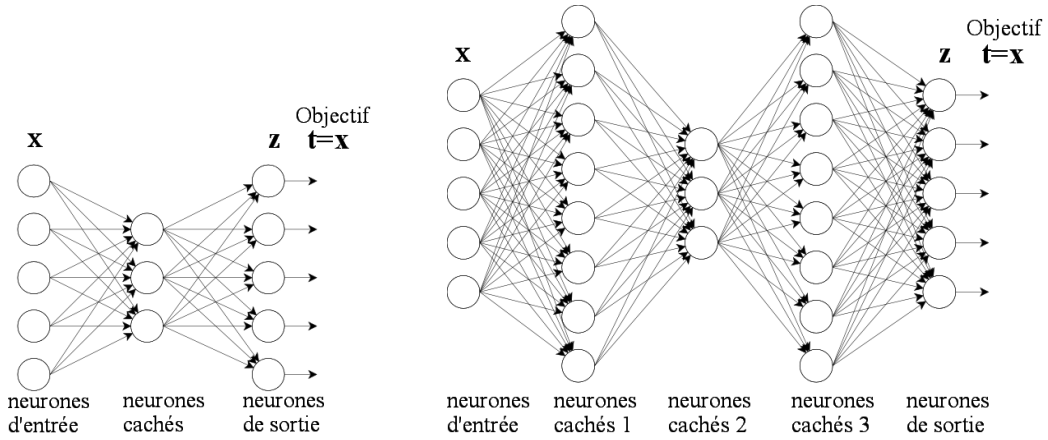
FIGURE 2.18 – Système de détection de visages invariant par rotation de Rowley *et al* . Un premier MLP est utilisé pour déterminer l'angle de rotation d'un visage présenté en entrée du système. La rotation est ensuite corrigée et après un prétraitement des images, un second MLP est utilisé pour déterminer si l'image d'entrée est un visage ou pas.

Dans [43] Rowley *et al* ajoutent au système de détection, un MLP capable de détecter l'angle de rotation d'un visage. Ainsi chaque imagette présentée en entrée du système voit son orientation corrigée de manière à pouvoir détecter des images de visages ayant subi une rotation (figure : 2.18). Inspiré par Baluja [5], ce MLP utilise une image en Niveaux de Gris en entrée et 36 neurones de sortie 'k' représentant chacun un angle de rotation de $k \times 10°$.

La grande variété d'architectures possibles pour un MLP en fait donc un outil extrêmement performant en traitement d'image. Un MLP peut aussi être utilisé afin d'extraire des descripteurs d'une image. De tels réseaux sont appelé réseaux auto-associatifs et effectuent une opération parfois appelé PCA non linéaire pour ses similitudes avec L'Analyse en Composantes Principales.

2.3.7.2 Perceptron Multicouche appliqué à l'extraction de descripteurs, réseaux auto-associatifs

Les réseaux de neurones auto-associatifs (AANN Auto-Associative Neural Networks) sont des Perceptrons Multicouches contenant autant de neurones d'entrée que de neurones de sortie (figure : 2.19). Ils sont entraînés de façon à ce que la sortie soit égale à l'entrée du réseau de neurones. La couche de neurones cachés centrale contient moins de neurones que l'entrée et la sortie. La sortie de la couche centrale **u** peut donc être utilisée comme descripteurs permettant de compresser l'information des vecteurs d'entrée. Le vecteur d'entrée **x** pouvant être reconstruit à partir de la partie droite du réseau et du vecteur **u**.



(a) AANN 3 couches (b) AANN 5 couches

FIGURE 2.19 – Réseaux de neurones auto-associatifs à trois couches et cinq couches. La sortie de la couche centrale aussi appelée couche de représentation permet de décrire les vecteurs d'entrée dans un espace de dimension plus réduite.

Dans le cas d'un AANN à trois couches (figure : 2.19a), si les fonctions d'activation sont linéaires, alors Balbi et Hornik [4] ont montré que la projection effectuée est équivalente à une PCA. Le vecteur **u** contient les plus grandes valeurs propres de la matrice de covariance et les poids des vecteurs de sortie correspondent aux vecteurs propres. Dans le cas où les fonctions d'activation sont non linéaires [50], l'AANN n'est plus équivalent à une PCA [11], les résultats restent cependant proches de ceux obtenus avec la PCA. L'utilisation de fonctions d'activation non linéaires n'améliorant notamment pas le taux de compression par rapport à une PCA [11, 48]. Ainsi, un tel système n'apportant pas de réels avantages par rapport à la PCA, on lui préfère généralement l'architecture à cinq couches (figure : 2.19b). Cette architecture permet avec l'utilisation de fonctions d'activation non linéaires de modéliser toute projection non linéaire des données d'entrée, ainsi que la fonction reconstruction correspondante. Cette méthode a été utilisée à de nombreuses reprises en traitement d'image, principalement pour des problèmes de compression et de reconnaissance [81, 79, 56, 118, 23, 85].

Une des raisons qui fait le succès des MLP, outre la variété des architectures possibles est qu'il existe un algorithme d'apprentissage relativement simple et efficace nommé algorithme de Rétropropagation que nous allons détailler dans la section suivante.

2.3.7.3 Entraînement d'un Perceptron Multicouche

Nous avons vu que toute fonction peut être modélisée à partir d'un Perceptron Multicouche. Dans cette section nous abordons le problème de déterminer les poids du MLP à partir d'un ensemble d'échantillons exemples $\mathbf{x_i}$ et de la sortie souhaitée $\mathbf{t_i}$ correspondante.

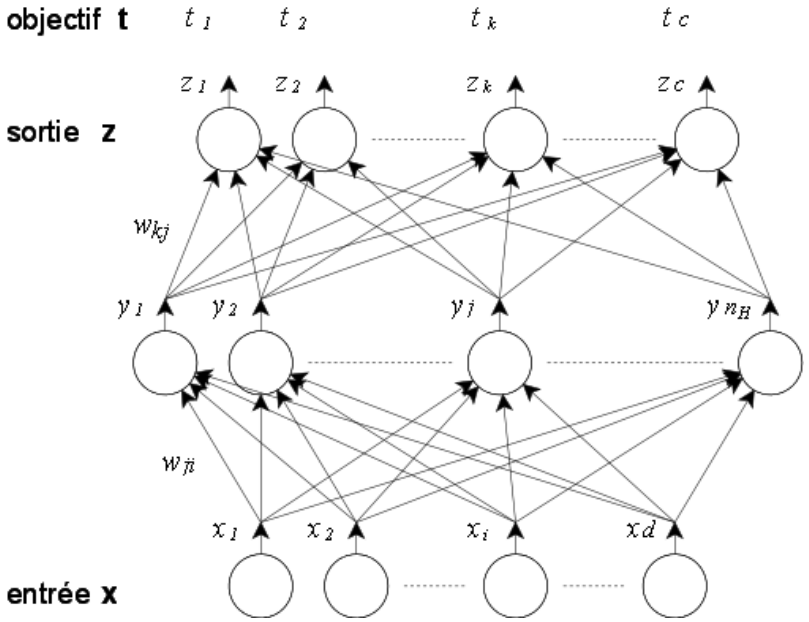


FIGURE 2.20 – MLP à trois couches entièrement connecté et notations utilisées. $\mathbf{x} = (x_1, \ldots, x_d)^T$ est un échantillon présenté au système. $\mathbf{z} = (z_1, \ldots, z_c)^T$ est la sortie correspondante et $\mathbf{t} = (t_1, \ldots, t_c)^T$ est la sortie souhaitée pour l'échantillon $\mathbf{x}$. Les poids w_{kj} correspondent aux synapses reliant le neurone de sortie k au neurone caché j. De même, les poids w_{ji} sont liés aux synapses reliant le neurone d'entrée i au neurone caché j. y_j est la sortie du j^{ieme} neurone caché. net_j est le produit scalaire entre les entrées du j^{ieme} neurone caché et les poids correspondants $y_j = \varphi(net_j)$. De même $z_k = \varphi(net_k)$.

Il existe de nombreuses méthodes permettant d'entraîner un réseau de neurones. L'utilisation d'un algorithme d'apprentissage plutôt qu'un autre dépend aussi bien de l'architecture du réseau et de la forme des exemples d'apprentissage que du problème à traiter. Cependant la méthode de Rétropropagation introduite par Rumhelhart *et al* [102] reste l'algorithme le plus utilisé et sans doute le plus universel pour entraîner un MLP. Afin de simplifier les notations, et les explications, nous détaillerons

ici la méthode pour un MLP à trois couches entièrement connecté (figure : 2.20), la méthode se généralisant aisément à toute architecture de MLP. L'algorithme de Rétropropagation est basé sur la minimisation par la méthode de descente par gradient de l'erreur quadratique E entre la sortie souhaité du MLP $\mathbf{t}$ et sortie réelle $\mathbf{z}$:

$$E(\mathbf{w}) = \frac{1}{2} \|\mathbf{z} - \mathbf{t}\|^2 \tag{2.84}$$

$\mathbf{w}$ étant l'ensemble des poids associés aux neurones du réseau. Le biais b des fonctions d'activation d'un neurone est souvent considéré comme un poids $w_0 = b$, le vecteur d'entrée du neurone ayant été augmenté par la valeur $x_0 = 1$. Afin de simplifier les notations nous noterons dorénavant w_{pq} les poids associés aux synapses reliant le neurone p d'une couche supérieure au neurone q de la couche inférieure. w_{p0} représentant le biais du neurone p de la couche supérieure. Les poids sont initialisés avec des valeurs aléatoires et sont ensuite itérativement modifiés dans la direction permettant de réduire l'erreur $E(\mathbf{w})$:

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E \tag{2.85}$$

Ainsi chaque poids w_{pq} est itérativement modifié de la valeur :

$$\Delta w_{pq} = -\eta \frac{\partial E}{\partial w_{pq}} \tag{2.86}$$

Ou $\eta > 0$ est le taux d'apprentissage et détermine le changement relatif des poids entre chaque itération. Sa valeur doit être choisie suffisamment grande pour minimiser le nombre d'itérations nécessaires pour atteindre un minimum, et suffisamment faible pour assurer la convergence. L'algorithme de Rétropropagation permet de calculer simplement $\frac{\partial E}{\partial w_{pq}}$ malgré la compléxité et le nombre de paramètres de la fonction $E(\mathbf{w})$.

En effet, soit le produit scalaire entre les poids et les entrées d'un neurone $net_k = \mathbf{w_k} \cdot \mathbf{y}$, et $\mathbf{net} = (net_1, \ldots, net_k, \ldots, net_c)^T$, alors on peut écrire pour les poids reliés à la derniere couche de neurone du MLP :

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}} \tag{2.87}$$

Avec $\delta_k = -\frac{\partial E}{\partial net_k}$ la sensibilité du k^{ieme} neurone. Si $\varphi(.)$ la fonction d'activation du neurone est dérivable, alors :

$$\delta_k = -\frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial net_k} = (t_k - z_k)\, \varphi_k'\,(net_k) \tag{2.88}$$

Et $\frac{\partial net_k}{\partial w_{kj}} = y_j$ la j^{ieme} entrée du neurone (correspondant pour un MLP entièrement connecté à la sortie du j^{ieme} neurone de la couche précédente).

Ainsi, nous pouvons aisément calculer la mise à jour des poids associés aux synapses liés aux neurones de sortie :

$$\Delta w_{kj} = \eta \delta_k y_j = \eta \left(t_k - z_k\right) \varphi'_k \left(net_k\right) y_j \tag{2.89}$$

La règle d'apprentissage pour les poids w_{ji} entre la couche d'entrée et la couche cachée est plus subtile. L'intérêt de l'algorithme de Rétropropagation est que l'on peut déduire Δw_{ji} de la sensibilité δ_k et des poids w_{kj} des neurones de la couche supérieure :

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \underbrace{\frac{\partial y_j}{\partial net_j}}_{\varphi'_j(net_j)} \underbrace{\frac{\partial net_j}{\partial w_{ji}}}_{x_i} \tag{2.90}$$

$$\frac{\partial E}{\partial y_j} = \nabla_{\mathbf{net}} E \cdot \frac{\partial \mathbf{net}}{\partial y_j} = -\sum_{k=1}^{c} \delta_k w_{kj} \tag{2.91}$$

$$\Delta w_{ji} = \eta \underbrace{\left[\sum_{k=1}^{c} \delta_k w_{kj}\right] \varphi'_j \left(net_j\right)}_{\delta_j} x_i \tag{2.92}$$

$$\Delta w_{ji} = \eta \delta_j x_i \tag{2.93}$$

Des équations 2.89 et 2.93, on peut déduire l'algorithme d'apprentissage pour un MLP à trois couches. On retrouve généralement deux variantes, la méthode de Rétropropagation stochastique (Stochastic Backpropagation : Algo 5) et la Rétropropagation par lots (Batch Backpropagation : Algo 6). L'algorithme stochastique met à jour les poids $\mathbf{w}$ du MLP à chaque présentation d'un échantillon exemple $\mathbf{x}$ au système. L'algorithme par lots ne met à jour les poids du réseau, qu'une fois l'ensemble des N échantillons exemples présentés au système. L'algorithme stochastique est généralement préféré car il converge plus vite que l'algorithme par lots, surtout pour des bases d'apprentissage contenant de nombreux échantillons. L'algorithme par lots permet quand à lui d'utiliser des méthodes d'apprentissage du second ordre difficilement utilisables avec la méthode stochastique, et se révélera ainsi plus performant pour le traitement de certain problèmes.

Par analogie avec l'équation 2.93, il est aisé de généraliser les algorithmes à n'importe quel MLP. En effet, la sensibilité δ_j d'un neurone se déduisant de la sensibilité δ_k des neurones de la couche supérieure (figure : 2.21), on peut calculer par récurrence les variations des poids du réseaux en partant de la dernière couche jusqu'a l'entrée du réseau pour un nombre quelconque de couches du MLP.

Différentes conditions d'arrêt peuvent être utilisées pour l'algorithme de Rétropropagation. Nous avons choisie ici, une condition d'arrêt liée à un nombre maximum d'itérations et une valeur seuil θ de l'erreur quadratique sur l'ensemble de la base référence $J(\mathbf{w})$. Un autre critère possible est par exemple $\|\nabla J(\mathbf{w})\| < \theta$. L'algorithme s'arrête alors quand l'erreur quadratique $J(\mathbf{w})$ approche un minimum local.

Algorithm 5 Stochastic Backpropagation

initialisation : η, $\mathbf{w}$, n_H, θ, max_{iter}, $m = 0$
while $m < max_{iter}$ and $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} \|\mathbf{z_i} - \mathbf{t_i}\|^2 > \theta$ **do**
  $m = m + 1$
  $\mathbf{x} \leftarrow i^{ieme}$ échantillon exemple
  $\delta_k = (t_k - z_k)\,\varphi'_k\,(net_k)$
  $\delta_j = [\sum_{k=1}^{c} \delta_k w_{kj}]\,\varphi'_j\,(net_j)$
  $w_{kj} = w_{kj} + \eta\delta_k y_j$
  $w_{ji} = w_{ji} + \eta\delta_j x_i$
end while
return $\mathbf{w}$

Algorithm 6 Batch Backpropagation

initialisation : η, $\mathbf{w}$, n_H, θ, max_{iter}, $m = 0$
while $m < max_{iter}$ and $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} \|\mathbf{z_i} - \mathbf{t_i}\|^2 > \theta$ **do**
  $m = m + 1$
  $\Delta w_{kj} = \Delta w_{ji} = 0$
  for i=1 to N **do**
    $\mathbf{x} \leftarrow$ échantillon exemple choisie aléatoirement
    $\delta_k = (t_k - z_k)\,\varphi'_k\,(net_k)$
    $\delta_j = [\sum_{k=1}^{c} \delta_k w_{kj}]\,\varphi'_j\,(net_j)$
    $\Delta w_{kj} = \Delta w_{kj} + \eta\delta_k y_j$
    $\Delta w_{ji} = \Delta w_{ji} + \eta\delta_j x_i$
  end for
  $w_{kj} = w_{kj} + \Delta w_{kj}$
  $w_{ji} = w_{ji} + \Delta w_{ji}$
end while
return $\mathbf{w}$

De nombreuses variations de l'algorithme de Rétropropagation ont été proposées dans la littérature. Le but principal étant d'accélérer la convergence, ou d'améliorer la généralisation du MLP (améliorer la classification sur des exemples ne figurant pas dans la base d'apprentissage). Parmi ces variantes on peut citer :

Validation croisée/(**Cross validation**), elle permet de minimiser les problèmes des sur-apprentissage et d'améliorer ainsi la généralisation du classifieur, cette méthode consiste à calculer $J(\mathbf{w})$ à partir d'une partie de la base d'exemples non utilisée pour l'apprentissage du MLP.

Weight Decay [124], est une autre méthode permettant d'améliorer la généralisation du MLP. Elle consiste à ajouter un terme de régularisation à l'erreur quadratique :

$$E = \frac{1}{2}\|\mathbf{z} - \mathbf{t}\|^2 + \frac{\alpha}{2}\|\mathbf{w}\|^2 \tag{2.94}$$

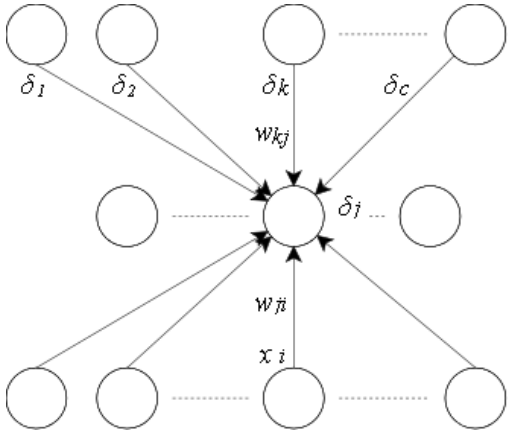


FIGURE 2.21 – La sensibilité δ_j d'un neurone se déduit directement de la sensibilité δ_k des neurones de la couche supérieure et des poids associés w_{kj} et w_{ji} liés au neurone : $\delta_j = [\sum_{k=1}^{c} \delta_k w_{kj}]\, \varphi'_j\,(net_j)$.

Avec $\alpha > 0$. Cette méthode permet ainsi d'éviter de trop grandes valeurs des poids w_{pq} avec pour conséquence une meilleure généralisation du MLP.

Momentum [88], est une variante qui permet d'accélérer la convergence de la Rétropropagation. L'idée est ici de calculer la variations $\Delta w_{pq}(n)$ des poids w_{pq} à la n^{ieme} itération en tenant compte des variations des poids à l'itération précédente.

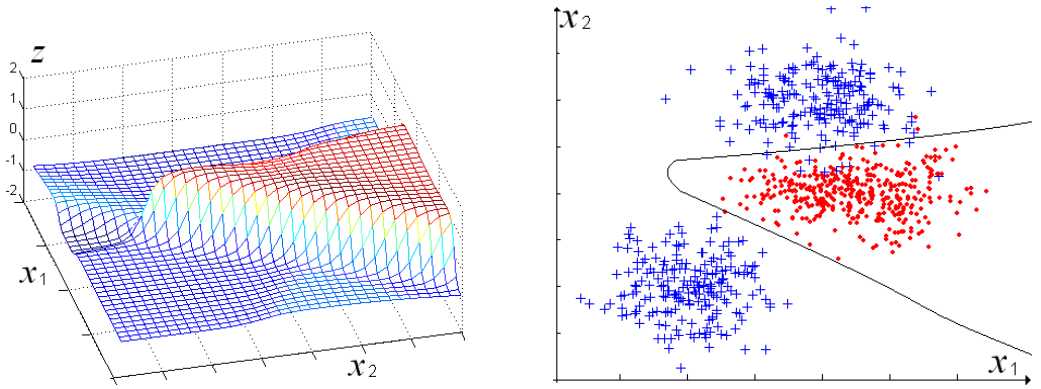
$$\Delta w_{pq}(n) = -\eta \frac{\partial E}{\partial w_{pq}(n)} + \alpha \Delta w_{pq}(n-1) \quad 0 < \alpha < 1 \tag{2.95}$$

Ceci a un effet de lissage sur les variations des poids entre chaque itération et permet généralement d'améliorer la vitesse de convergence.

Enfin on peut trouver de nombreuses méthodes agissant sur la valeur du taux d'apprentissage η [105, 6, 7, 86, 69, 49]. Le principe étant de faire varier la valeur de η au cours des itérations successives de l'algorithme de Rétropropagation afin d'accélérer et d'améliorer la convergence.

Ces méthodes d'optimisation de l'algorithme de Rétropropagation ne sont pas toujours compatibles avec l'algorithme stochastique. De plus, leur intérêt est fortement dépendant de la forme du MLP, du nombre d'échantillons exemples ainsi que du type d'information à classer. Dans la majorité des problèmes, l'algorithme de Rétropropagation stochastique est préféré. La figure 2.22 montre la sortie obtenue pour un MLP entraîné par la méthode de Rétropropagation stochastique pour un problème à deux classes dans un espace bidimensionnel. Nous pouvons voir que les frontières définies par le MLP sont proches de ce que l'on pouvait espérer.

Le Perceptron Multicouche constitue aujourd'hui un des classifieurs les plus utilisé. La variété de ses architectures et l'existence de méthodes d'apprentissages simples et efficaces en font un outil performant et adaptable à de très nombreux problèmes, notamment la détection de visages. Une autre méthode de classification basée sur les



(a) courbe tridimensionnelle de la sortie d'un MLP

(b) frontière générée par le MLP

FIGURE 2.22 – Exemple de sortie d'un Perceptron Multicouche utilisé pour séparer deux classes dans un espace bidimensionnel. Le MLP possède 10 neurones cachés et un seul neurone de sortie. Les fonctions d'activation utilisées sont des tangentes hyperboliques. Le système est entraîné de manière à ce que la sortie z du MLP soit -1 ou 1 en fonction de la classe. La frontière correspond aux valeurs de l'entrée $\mathbf{x}$ telles que $g(\mathbf{x}) = z = 0$.

réseaux de neurones ayant donné de bons résultats en détection est le SNOW (Sparse Network Of Winnows).

2.3.8 SNOW

L'architecture de réseau de neurones appelée SNOW à été mise au point en 1998 par Roth [99] et appliquée à la détection de visages en 2000 [100]. Le SNOW est un réseau de neurones consistant en une seule couche de neurones avec une fonction d'activation linéaire. A la différence d'un MLP, le vecteur d'entrées ne contient que des valeurs binaires (0 ou 1). La sortie d'un neurone j est la somme des poids w_{ji} avec $x_i = 1$ (figure : 2.23).

Si nous prenons l'exemple du détecteur de visages, l'image d'un visage en Niveaux de Gris de dimension $h \times l$ et pour lequel l'intensité I de chaque pixel en (u, v) est codée sur 8 bits ($0 \leq I(u, v) < 256$). Le vecteur d'entrée $\mathbf{x}$ du SNOW est alors de dimension $h \times l \times 256$. la valeur du i^{ieme} élément de $\mathbf{x}$ est alors 1 pour les valeurs de $i = (u \times l + v) \times 256 + I(u, v)$ et 0 sinon. Sur une image de 20×20 pixels on arrive à un vecteur d'entrée de dimension $d = 102400$. Cependant, seulement 400 éléments de $\mathbf{x}$ sont égaux à 1 (actifs), limitant le calcul de la sortie d'un neurone à une somme de 400 éléments. Ainsi, les temps de calcul du réseau de neurones ne sont pas impactés par la grande dimension du vecteur d'entrée. Le système de détection de visages de Yang *et al* [100] comporte seulement deux neurones, si la sortie net_1 du premier neurone est supérieure à la sortie net_2 du second, alors, un visage est détecté. Dans le cas contraire l'échantillon d'entrée est classé dans la catégorie 'non visage'. Avec seulement deux neurones, et une base d'apprentissage de 1681 visages,

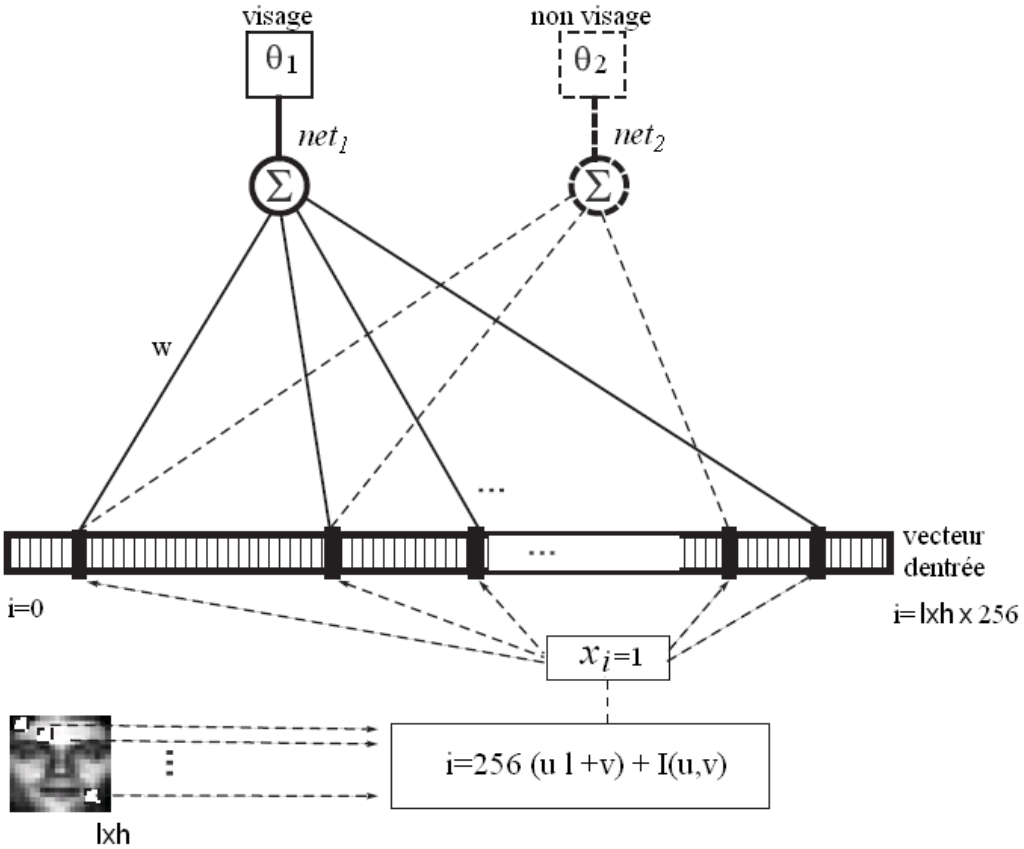


FIGURE 2.23 – SNOW utilisé dans un système de détection de visages. L'image en 256 Niveaux de Gris est représentée sous la forme d'un vecteur **x** binaire de dimension hauteur × largeur × 256.

ce système a montré de très bon résultats en détection de visages avec des taux de détection équivalents à ceux obtenus avec un MLP [42] ou AdaBoost [122].

2.3.8.1 Apprentissage

Différentes méthodes d'apprentissage peuvent être utilisées pour entraîner un SNOW, y compris l'algorithme de Rétropropagation, ce réseau de neurone pouvant être considéré comme un MLP à deux couches avec des fonctions d'activation linéaires. Nous décrirons ici la méthode utilisée dans [100], elle même dérivée de la méthode de mise à jour des poids introduite par Littlestone dans [66]. La méthode utilisée est dite en ligne et basée décision, c'est à dire que les poids sont mis à jour à la présentation de chaque nouvel échantillon labellisé, uniquement dans le cas ou cet échantillon est mal classé (on dira de cet échantillon qu'il est *pertinent* pour l'apprentissage).

A chaque neurone est associé un seuil θ_j. la sortie z_j du neurone prend la valeur 1 si $net_j = \sum_i x_i w_{ji} > \theta_j$ et 0 sinon. On définie en plus deux paramètres de mises à

jour $\alpha > 1$ et $0 < \beta < 1$. Enfin, à chaque échantillon exemple $\mathbf{x}$ présenté au système est associé un objectif t_j pour chaque neurone de sortie. La règle de mise à jour des poids est alors définie par l'algorithme 7.

Algorithm 7 Littlestone Winnow Update Rule

initialisation : α, β, θ_j
$\mathbf{x} \leftarrow$ échantillon exemple choisie aléatoirement
if $z_j \neq t_j$ **then**
  if $z_j = 1$ **then**
    $\forall i \ / x_i = 1, \quad w_{ji} = \beta x_i w_{ji}$
  else
    $\forall i \ / x_i = 1, \quad w_{ji} = \alpha x_i w_{ji}$
  end if
end if
return $\mathbf{w} = (\ldots, w_{ji}, \ldots)^T$

Ainsi, si un échantillon est mal classé par le neurone j car la somme des poids liés aux éléments actifs de l'entrée $\mathbf{x}$ est trop grande, alors la valeur de ces poids est diminuées d'un facteur β. Si un échantillon est mal classé car la somme de ces poids est trop faible, ces derniers voient leur valeur augmentée d'un facteur α.

Cet algorithme d'apprentissage s'avère très efficace malgré le très grand nombre de poids associés à un neurone. En effet, à chaque itération seule une très faible partie des poids est mise à jour (400 sur les 102400 que compte un neurone pour le détecteur de visages). De plus, le nombre d'échantillons pertinents pour l'apprentissage (mal classés) ne croit que de façon logarithmique par rapport au nombre total d'échantillons à classer. Enfin cette méthode est connue pour apprendre efficacement toute fonction de séparation linéaire et pour être robuste en présence de divers types de bruit [67, 54].

Une variante de ce détecteur de visages décrite dans [100] consiste à extraire les descripteurs booléens de l'image à tester, non pas en discrétisant directement la valeur du niveau de gris de chaque pixel, mais en utilisant des descripteurs représentant une information multi-échelle. Une image de 20×20 pixels est ainsi divisée en sous-régions de 1×1, 2×2, 4×4 et 10×10 pixels conduisant à un nombre de descripteurs actifs (élément de $\mathbf{x}$ égaux à 1) de $400 + 100 + 25 + 4 = 529$. Chaque sous-région est décrite par la valeur moyenne de ses pixels ainsi que leur variance. Les moyennes et les variances sont encodées dans 100 classes distinctes conduisant à un vecteur binaire d'entrée de dimension 529×100 éléments. Cette méthode à montré des résultats très légèrement supérieurs à celle utilisant directement la valeur des pixels en Niveaux de Gris.

2.3.9 Reseaux Convolutionnels

Une image a une structure bidimensionnelle ou la valeur de chaque pixel est fortement corrélée avec les pixels voisins. Les meilleurs systèmes de détection dans une image tiennent d'ailleurs généralement compte des particularités liées à une image.

Viola et Jones [122] utilisent des fonctions de Haar effectuant ainsi des corrélations locales, Sung et Poggio [116] projettent les images dans des sous-espaces et Rowley *et al* [42] utilisent un MLP non complètement connecté découpant l'image en sous-régions (bandes horizontales et sous-régions rectangulaires). La structure du réseau convolutionnel (CNN pour Convolutional Neural Networks) inspiré du système visuel du chat [47] permet de l'utiliser directement sur une image en Niveaux de Gris sans prétraitement d'image ou extraction de descripteurs. Pour ce faire, l'architecture du réseau convolutionnel combine successivement des étapes de convolutions et de sous échantillonnages. L'étape de convolution est effectuée par la combinaison de deux techniques. Les champs réceptifs locaux (local receptive fields) permettant d'extraire des formes élémentaires tels que des contours orientés ou des coins [37, 38, 78, 59, 60] et le partage des poids (shared weights) [102, 38, 59] permettant de limiter le nombre de paramètres du réseau de neurones.

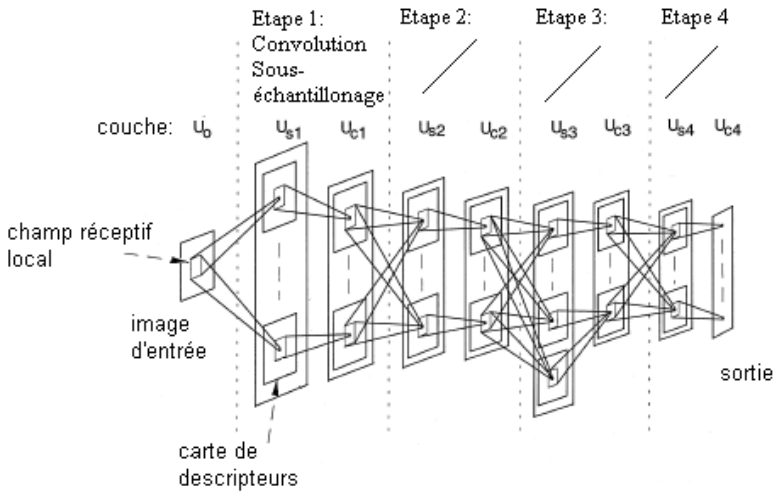


FIGURE 2.24 – Structure de base du Neocognitron : L'image de départ est filtrée, puis sous échantillonnée par des filtres appris (correspondant aux poids des neurones), l'opération est ensuite répétée afin de réduire progressivement la dimension des images (ou carte de descripteurs) et de faire en sorte que les filtres utilisés correspondent à une information de plus en plus spécifique aux images à classer.

La première implémentation d'un réseau de neurones convolutionnel a été proposée par Fukushima [37, 38] et a été appliquée au problème de la reconnaissance de caractères écrits manuellement. Chaque neurone n'est connecté qu'à une sous-région (champ réceptif local) correspondant à un certain nombre de neurones voisins dans la couche de neurones précédente. Ceci constitue donc un extracteur de descripteur local. Cet extracteur de descripteur pouvant être utilisé dans différentes zones d'une image, les poids sont partagés pour toutes les positions possibles du champ réceptif local, conduisant donc à convoluer l'image de la couche précédente avec la matrice

constituée des poids de chaque neurone. Chaque neurone formant ainsi à la couche suivante l'image filtrée (carte de descripteurs) d'une image de la couche précédente. L'image obtenue est ensuite sous-échantillonnée permettant ainsi de réduire la sensibilité du CNN aux variations d'échelle ou aux translations de l'image d'entrée. Le CNN consiste donc en une succession d'images filtrées puis sous-échantillonnées (le plus souvent par un facteur 2) (figure : 2.24).

De part la structure en couches successives du CNN, l'algorithme de Rétropropagation peut être utilisé pour entraîner ce dernier. dans [61], LeCun *et al* présentent le premier CNN entraîné grâce à l'algorithme de Rétropropagation et l'appliquent à la reconnaissance de caractères. L'architecture de ce CNN plus simple que le Neocognitron est présentée figure 2.25 et constitue souvent une base pour de nombreux systèmes utilisant des CNN.

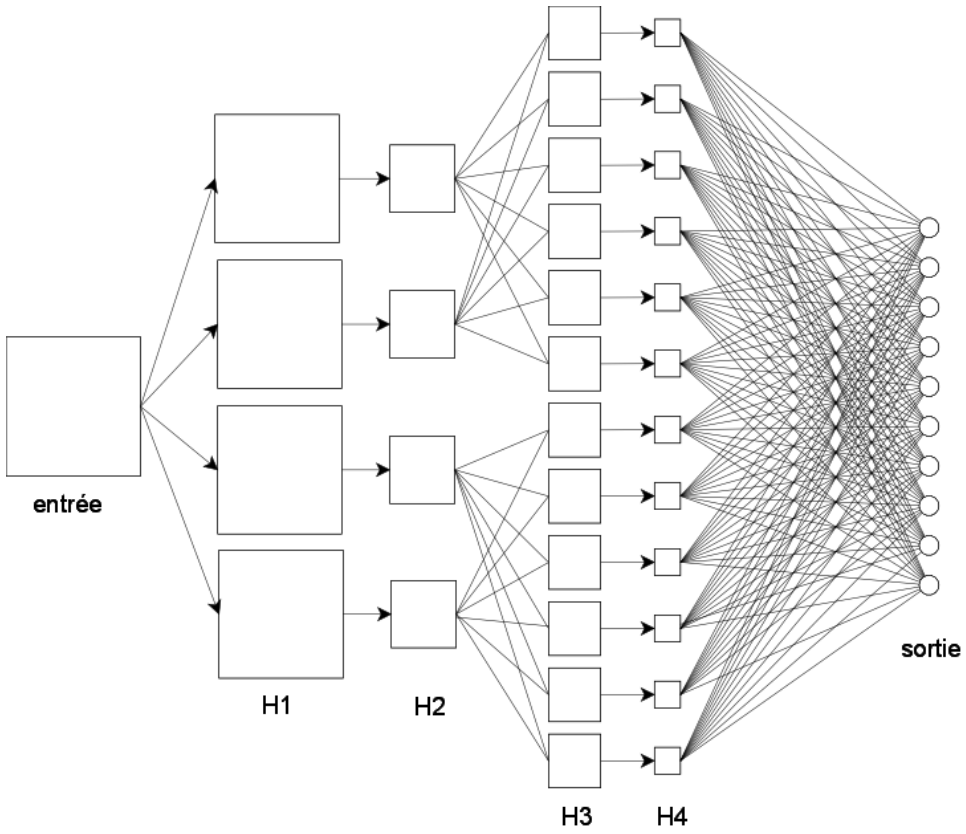


FIGURE 2.25 – Structure du CNN de LeCun pour la reconnaissance d'écriture manuelle de chiffres : L'image de départ est filtrée puis sous-échantillonnée par un facteur 2, respectivement par la couche H1 et H2. L'opération est ensuite réitérée sur les images correspondant à la couche H2 par les couches de neurones H3 et H4. On retrouve ensuite en sortie 10 neurones (un pour chaque chiffre à reconnaître).

2.3.9.1 Réseau de neurones convolutionnel pour la détection

Afin d'illustrer l'utilisation des CNN, nous présentons dans cette section le système de détection de visages de Garcia et Delakis [39] qui constitue aujourd'hui, un des systèmes de détection de visages les plus performant, autant en vitesse de calculs qu'au niveau du taux de détection. L'architecture du CFF (Convolutional Face Finder) (figure : 2.26) est inspirée de celle de LeCun *et al* [61]. Le réseau est entraîné pour renvoyer la valeur -1 ou 1 en fonction de la catégorie 'non visage' ou 'visage' des images de 32×36 pixels présentées en entrée. La première couche du CFF est constituée de quatre cartes de dimension 28×32. Chaque carte correspondant à l'image d'entrée convoluée à l'aide d'un filtre appris de dimension 5×5 suivie de l'ajout d'un biais b. Le filtre et le biais correspondent aux poids et au biais d'un neurone localement connecté avec 5×5 poids. Ainsi la première couche du CFF est calculée à partir de seulement $(5 \times 5 + 1) \times 4 = 104$ paramètres. Le sous-échantillonnage est effectué à partir d'une lentille 2×2. Chaque neurone calcule la moyenne des quatre entrées multiplié par un coefficient a, ajoute un biais b et renvoie la tangente hyperbolique de ce résultat. Les champs réceptifs locaux (lentilles 2×2) des neurones ne se recouvrent pas et partagent les mêmes poids. Ainsi la carte de descripteurs obtenue est de dimension 14×16 et le nombre de paramètres liés à la couche de sous-échantillonnage est de $2 \times 4 = 8$. Une nouvelle convolution est ensuite effectuée à partir de deux lentilles 3×3 sur chacune des quatre cartes de descripteurs de la couche S1. De plus, une lentille de dimension $3 \times 3 \times 2$ est appliquée à chaque paire de cartes possibles afin de fusionner les informations des différents descripteurs de la couche précédente. La couche C2 renvoie alors $8 + 6 = 14$ cartes de descripteurs et contient 194 paramètres $((3 \times 3 + 1) \times 8 + (3 \times 3 \times 2 + 1) \times 6)$. Enfin la dernière Couche S2 sous-échantillonne de la même manière que la couche S1, conduisant donc à $14 \times 2 = 28$ paramètres supplémentaires. Chacune des 14 cartes de dimension 7×5 est ensuite reliée à un des 14 neurones de la couche N1 eux même connectés au neurone de sortie.

Ainsi un tel réseau de neurones est capable de distinguer une image de 'visage' d'une image de 'non visage' avec seulement 951 paramètres entraînables, ce qui reste très faible au regard de la dimension des données d'entrée $(36 \times 32 = 1152)$.

De plus les étapes de convolutions et de sous échantillonnages peuvent être effectuées directement sur l'ensemble d'une image et non sur chaque rétine correspondant à l'image de 36×32 pixels traité par le CFF. Ainsi, d'un point de vue complexité de calcul, le réseau convolutionnel est très adapté aux problèmes de détection.

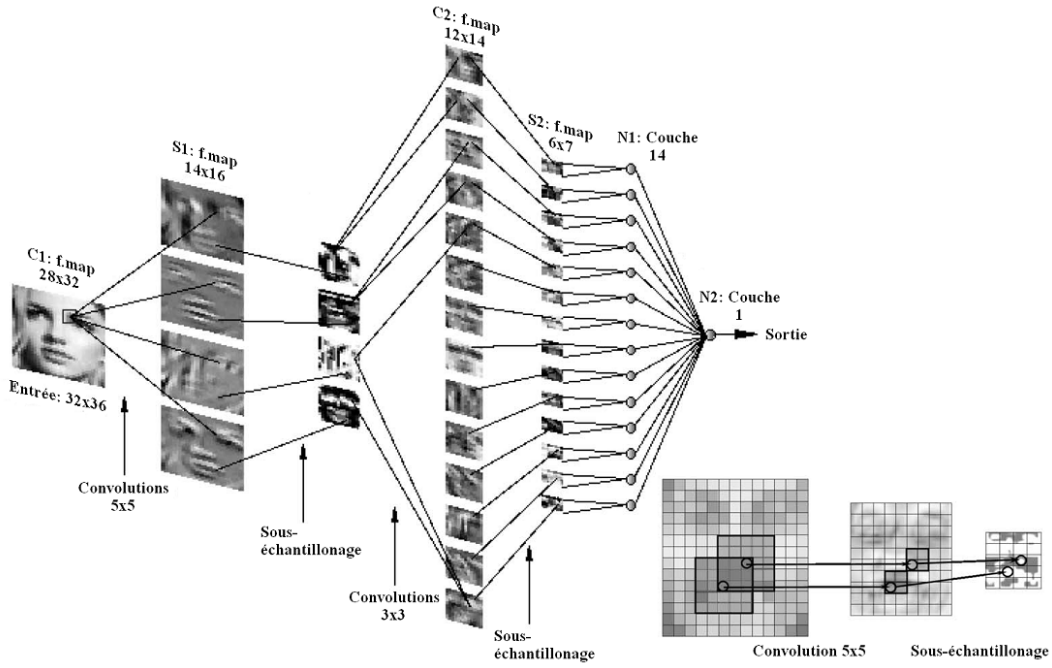


FIGURE 2.26 – Structure du CFF : L'image de départ est filtrée par quatre lentilles 5×5 puis sous-échantillonnée par un facteur 2. L'opération est ensuite réitérée avec deux lentilles 3×3 sur chaque image correspondante de la couche S1 ainsi qu'une lentille $3 \times 3 \times 2$ pour chaque paire d'images possibles de la couche S1, conduisant à 14 cartes de descripteurs dans la couche S2. Finalement le CFF se termine par un perceptron à trois couches constitué des couches S2, N1 et N2.