Université de Lorraine

Ecole Nationale Supérieure
d'Electricité de Mécanique

Filière : Ingénierie Systèmes Numériques (ISN SLE)

Compte rendu du TP : Traitement d'images

Détection de Piétons

Préparé par :

Brini Walid

Enseignant :

LE CAM Steven

Année académique :

2023/2024

Table des matières

1 Introduction

1.1 Mise en contexte

Ce TP a pour objectif de développer une méthode de détection de piétons à partir d'images fixes, en utilisant la méthode de Dalal et Triggs . Cette méthode introduit les histogrammes de gradients orientés (HOG) pour caractériser les silhouettes des personnes debout dans les images. Nous utiliserons une base de données fournie par les auteurs (`http://pascal.inrialpes.fr/data/human/`) contenant des milliers d'images avec ou sans piétons. Une partie des images positives a été normalisée pour créer des ensembles d'apprentissage et de test. Ces images, de taille 64 par 128 pixels, montrent des personnes centrées.

1.2 Approche et méthode de travail

Afin d'achiver l'objectif souhaité Nous divisons ce rapport en 4 étapes :

1. Calcul des gradients d'une image
2. Calcul des histogrammes de gradients
3. Évaluation des caractéristiques extraites
4. Interprétation et amélioration

Les prochains chapitres couvriront la méthode théorique de chaque étape et leur implémentation sur MATLAB.

2 Travail effectué

2.1 Calcul des gradients d'une image

La première étape consiste au calcul du gradient d'une image. Pour cela, nous avons implémenté une fonction sur MATLAB, `function [Or, Grad] = gradient(I)`, qui calcule le gradient d'une image et retourne l'orientation et la magnitude du gradient.

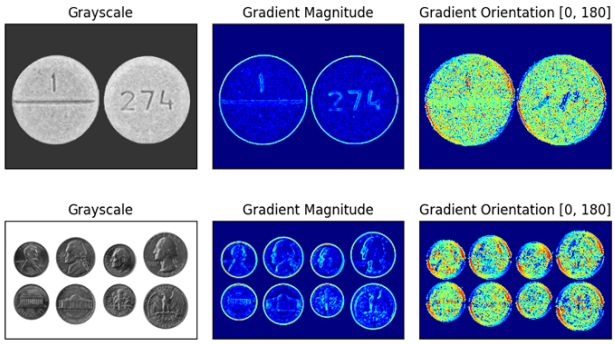


Figure 1: Exemple du calcul du gradient d'une image

La fonction `gradient` implémente le calcul des gradients d'une image en termes de magnitude et d'orientation. Voici un aperçu de son fonctionnement :

1. **Filtres de Sobel** : Les filtres de Sobel hx et hy sont utilisés pour calculer les gradients horizontal et vertical de l'image.
2. **Padding de l'image** : Une marge est ajoutée à l'image pour gérer les bords lors de la convolution.
3. **Calcul des gradients** : Les gradients horizontal et vertical sont calculés à l'aide des filtres de Sobel et de la convolution.
4. **Magnitude du gradient** : La magnitude du gradient est calculée en combinant les gradients horizontal et vertical.
5. **Orientation du gradient** : L'orientation du gradient est déterminée à l'aide de la fonction `atan2`.
6. **Normalisation** : La magnitude et l'orientation du gradient sont normalisées.
7. **Filtrage des orientations** : Les orientations de gradient avec une magnitude inférieure à un seuil sont fixées à zéro.
8. **Définition du seuil** : val est un seuil utilisé pour filtrer les orientations de gradient. Les orientations de gradient ayant une magnitude inférieure à ce seuil seront considérées comme nulles.

Le code ci-dessous, commenté, présente l'implémentation de la fonction gradient en MATLAB.

```
% Fonction qui calcul le gradient d'une image
% Retourne Or, Grad

function [Or, Grad] = gradient(I)
    hy = [1 0 -1];
    hx = [1 0 -1].';

    val = 0.15;

    % Ajouter un padding al'image pour gerer les bords
    padSize = 1;
    I_padded = padarray(I, [padSize, padSize], 'replicate');

    % Calculer les gradients
    GradE = conv2(I_padded, hy, 'same');
    GradN = conv2(I_padded, hx, 'same');

    % Enlever le padding
    GradE =GradE(padSize+1:end-padSize, padSize+1:end-padSize);
    GradN =GradN(padSize+1:end-padSize, padSize+1:end-padSize);

    % Magnitude du gradient
    Grad_sqrt = sqrt(GradE.^2 + GradN.^2);

    % Orientation du gradient
    Or = atan2(GradN, GradE);

    % Normaliser La Magnitude
    Grad = Grad_sqrt / max(Grad_sqrt(:));

    % Normaliser l'orientation
    Or = Or / (2 * pi) + 0.5;
    Or(Grad < val) = 0;

    % Afficher
    montage({Grad, Or});
end
```

Dans le fichier *main.m*, cette fonction est appelée après avoir transformé l'image en niveaux de gris avant de calculer le gradient.

```
image = imread('person_202d_new.png');
ing = im2gray(image);
% Calcul du gradient
[Or, G] = gradient(ing);
```

Le résultat des gradients appliqués à l'image `person202dnew.png` est le suivant : à gauche, on trouve la magnitude du gradient et à droite, l'orientation du gradient.

Figure 2: Résultat du calcul du gradient

2.2 Calcul des HOG

Les HOG (Histograms of Oriented Gradients) sont une technique d'extraction de caractéristiques basée sur le calcul du gradient orienté en chaque point de l'image. Ces gradients sont localement regroupés en histogrammes sur des fenêtres recouvrant l'ensemble de l'image. L'ensemble des profils d'orientation ainsi obtenus permettent de caractériser les formes d'individus. L'objectif est de créer une fonction permettant d'extraire ces histogrammes de gradient d'une image, qu'on supposera divisible en blocs de taille $B \times B$.

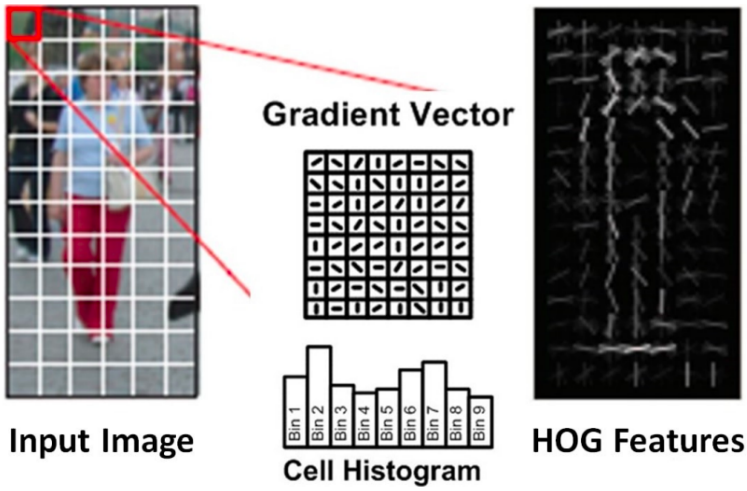


Figure 3: Extraction de caractéristiques basée sur le calcul du gradient

Les différentes étapes que doit réaliser cette fonction sont les suivantes :

1. Calcul de la norme du gradient et de l'orientation pour l'image I avec le filtre h. Reprendre le code précédent en répliquant les premières et dernières lignes afin d'éviter les effets de bord.

2. On considère les orientations modulo π. En effet, seul l'orientation du contour est importante, et le passage du noir au blanc ou du blanc au noir ne nous intéresse pas. Ramener les valeurs des orientations dans l'intervalle $[0, \pi]$.

3. Parcourir l'image avec une fenêtre de taille $B \times B$ (sans recouvrement), et pour chaque fenêtre :

   - Extraire les valeurs des gradients et des orientations correspondantes (sous-matrice de taille $B \times B$).
   - Créer l'histogramme d'orientation (pondéré par leur norme) pour cette fenêtre. Cet histogramme est constitué de $nbins$ bins de même taille $\frac{\pi}{nbins}$.

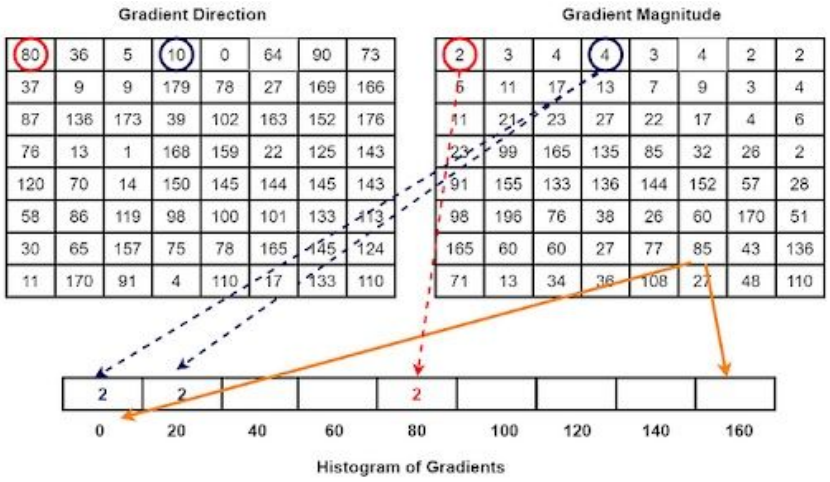


Figure 4: Création de l'histogramme d'orientation

   - Normaliser cet histogramme par la norme de ces valeurs, en s'assurant au préalable que cette norme n'est pas nulle.
   - Ajouter l'histogramme au vecteur de caractéristiques HOG de l'image.

On implémente cet algorithme sur MATLAB à l'aide d'une fonction `hogfeatures(I, h, B, nbins)`.

```
function hogs = hogfeatures(I, h, B, nbins)
    GradE = conv2(I, h, 'same');
    GradN = conv2(I, h.', 'same');
    % Calcul gradient
    Grad_sqrt = sqrt(GradE.^2 + GradN.^2);
    Or = atan2(GradN, GradE);
    Or(Or < 0) = Or(Or < 0) + pi;
    [H, W] = size(I);
    % Calcul N et M bloc val entiere
    blocH = floor(H / B);
    blocW = floor(W / B);
    % Init
    hogs = [];
    % Definir les bornes
    bin_edges = linspace(0, pi, nbins+1);
    for j = 1: blocW
        for i = 1: blocH
            % Extraire fenetre taille BxB
            window_norme = Grad_sqrt((i-1)*B+1 : i*B, (j-1)*B+1
                : j*B);
            window_orientation = Or((i-1)*B+1:i*B, (j-1)*B+1:j*
               B);
            % Init histogram pour la fenetere courante
            hist = zeros(1, nbins);
            % Calcul histogram pour la fenetre courante
            for bin = 1:nbins
                % Trouvez les pixel qui tombe dans la bin
                   courante en
                bin_mask = (window_orientation >= bin_edges(bin
                   )) & (window_orientation < bin_edges(bin+1)
                   );
                % Somme des magnitudes
                hist(bin) = sum(window_norme(bin_mask));
            end
            % Normaliser l'histogram
            hist_norm = norm(hist);
            if hist_norm > 0
                hist = hist / hist_norm;
            end
            % Ajouter l'histogram au HOG feature vector
            hogs = [hogs, hist];
        end
    end
end
```

Pour visualiser le résultat de l'extraction des caractéristiques HOG sur les images `person.png` et `gantrycrane.png`, nous avons défini une structure de visualisation particulière. Cette structure est inspirée de la documentation de la méthode `extractHOGFeatures` de MATLAB.

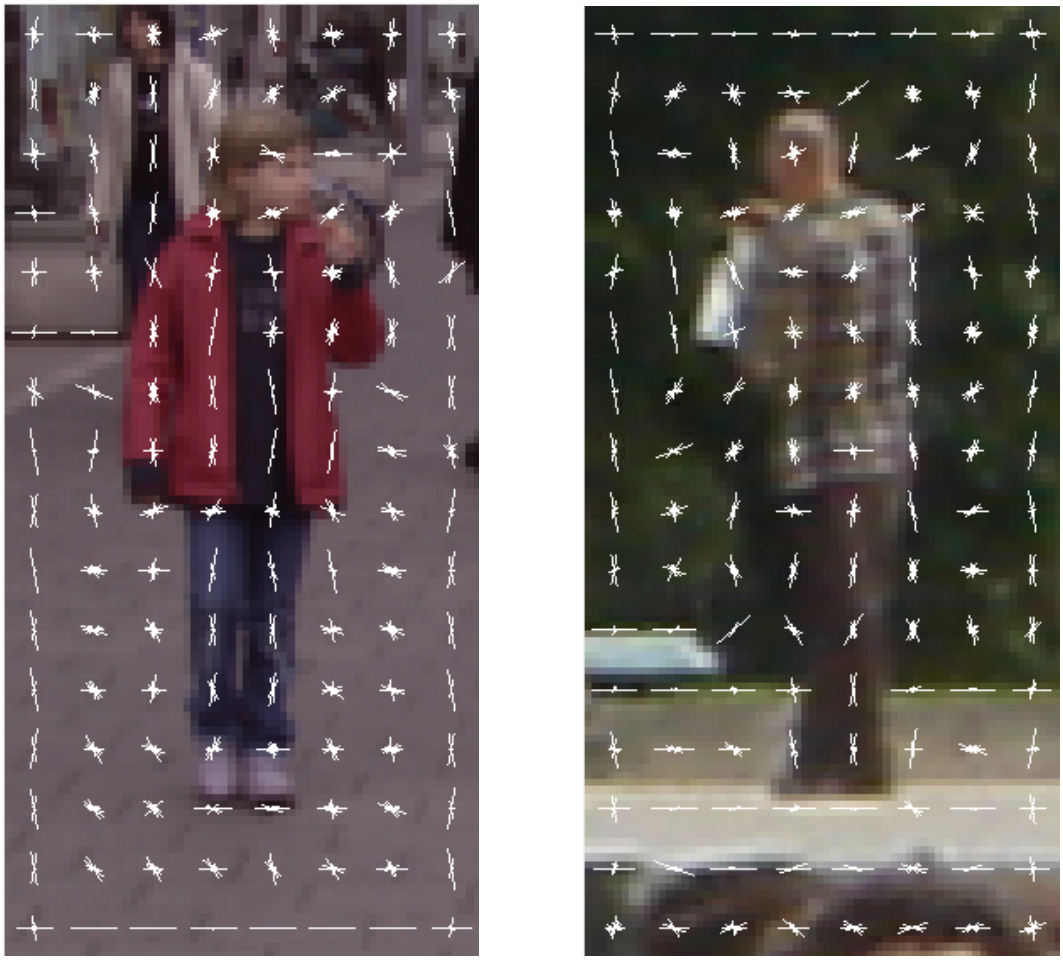
```
% Parametres d'extraction et de visualisation
[H,W]=size(ing); hCell=8; wCell=8;
nbhCell=H/hCell; nbwCell=W/wCell;
nbBins=9;

% Parametres pour la visualisation
param.ImageSize=[H W];
param.WindowSize=[H W];
param.CellSize=[hCell wCell];
param.BlockSize=[1 1];
param.BlockOverlap=[0 0];
param.NumBins=9;
param.UseSignedOrientation=0;

% Extraction
hogfeat=hogfeatures(double(ing),[1 0 -1],hCell,nbBins);

% Visualisation
visu=vision.internal.hog.Visualization(hogfeat, param);
figure; imshow(uint8(image));
hold on;
plot(visu)
title('HOGs manual')
pause(0.1)
```

On visualise le résultat sur les deux images `person.png` et `B10_crop001119a.png`.



(a) Image `person.png` (b) Image `B10_crop001119a.png`

Figure 5: Visualisation des composantes de l'histogramme de gradient dans chaque bloc

Intérpreation : On observe que la variation des directions du gradient est très variable dans les zones de fond de l'image. En revanche, au niveau des contours de la personne, cette variation est persistante et bien définie, en particulier au niveau des pieds et autour de la tête de la personne. Cette caractéristique est cruciale pour la détection des silhouettes des personnes. Cela permet de différencier efficacement les personnes du fond, facilitant ainsi leur détection.

2.3 Extraction des caractéristiques de la base de données

Pour extraire les caractéristiques de la base de données, qui est constituée de deux répertoires : `Pos/` contenant des images de personnes et `Neg/` contenant des images sans personnes, nous commençons par déterminer le nombre d'images dans chaque répertoire, stockés dans les variables N_p et N_g.

```
filePatternPos = fullfile(imagefilesPos, '*.png');
filePatternNeg = fullfile(imagefilesNeg, '*.png');

pngFilesP = dir(filePatternPos)
pngFilesN = dir(filePatternNeg)

Np = length(pngFilesP);
Ng = length(pngFilesN);
```

Ensuite, nous itérons à travers chaque répertoire pour calculer les hogfeatures de chaque image et les concaténer à la matrice d'apprentissage des images positives (`train_matrix_pos`).

```
M = nbBins * H/hCell*W/wCell;

% Initialisation de la matrice des features positives
train_matrix_pos = zeros(Np, M);

% Lecture des images positives et creation de la
   train_matrix_pos
for k = 1:Np
  baseFileName = pngFilesP(k).name;
  fullFileName = fullfile(imagefilesPos, baseFileName);
  fprintf(1, 'Now reading %s\n', fullFileName);
  imageArray = imread(fullFileName);
  ing = rgb2gray(imageArray);
  hogfeat  = hogfeatures(double(ing),[1 0 -1],hCell,nbBins);
  train_matrix_pos(k,:) = hogfeat';
end
```

Cette même approche est effectuée pour les répertoires contenant les images négatives.

Enfin, nous créons la matrice d'apprentissage (`train_matrix`), qui est la concaténation des matrices `train_matrix_pos` et `train_matrix_neg`, et nous définissons les étiquettes (`labels`) pour chaque donnée : 1 si elle est positive et 0 si elle est négative.

```
train_matrix = [train_matrix_pos ; train_matrix_neg];
labels =[ones(Np,1);zeros(Ng, 1 )];
```

2.4 Analyse en Composantes Principales

Après avoir prétraité les données, nous procédons à l'analyse en composantes principales (ACP) pour réduire la dimensionnalité et extraire les caractéristiques les plus significatives. Voici les étapes de l'ACP et de la visualisation des caractéristiques :

1. **Centrage des données :** Nous centrons les données en soustrayant la moyenne de chaque colonne, ce qui les aligne autour de l'origine.
2. **Normalisation des données :** Les données centrées sont normalisées en divisant chaque valeur par l'écart type de sa colonne respective. Cela garantit que toutes les variables ont une variance unitaire.
3. **Calcul de la matrice de covariance :** La matrice de covariance est calculée pour quantifier les relations entre les différentes caractéristiques des données. Elle est de taille 1152×1152.
4. **Calcul des vecteurs propres et valeurs propres :** Les valeurs propres indiquent l'importance de chaque nouvelle dimension dans la représentation des données, tandis que les vecteurs propres déterminent les directions dans lesquelles les données varient le plus.
5. **Tri descendant des valeurs propres et des vecteurs propres :** Les valeurs propres et les vecteurs propres sont triés par ordre décroissant pour identifier les composantes les plus significatives.
6. **Projection des données dans l'espace des composantes principales :** Les données sont projetées dans l'espace défini par les vecteurs propres triés, ce qui permet de réduire la dimensionnalité tout en préservant au maximum la variance des données.

Ces étapes sont implementés sur matlab avec le code ci-dessous :

```
X = train_matrix;
[p, n] = size(X);
% Centrage et normalisation des donnees (etape 1 et 2)
Xm = X - mean(X);
Xs = Xm * diag(1 ./ std(Xm));
% Calcul de la matrice de covariance (etape 3)
covV = Xs' * Xs / p;
% Calcul des vecteurs propres et valeurs propres (etape 4)
[U, D, V] = eig(covV);
% Tri descendant des valeurs propres (etape 5)
[Ds, Isort] = sort(diag(D), 'descend');
V = V(:, Isort);
% Projection des donnees dans l'espace des composantes
   principales(etape 6)
Xp = Xs * V;
```

Visualisation de l'évolution des valeurs propres : Les valeurs propres représentent l'importance de chaque composante principale.

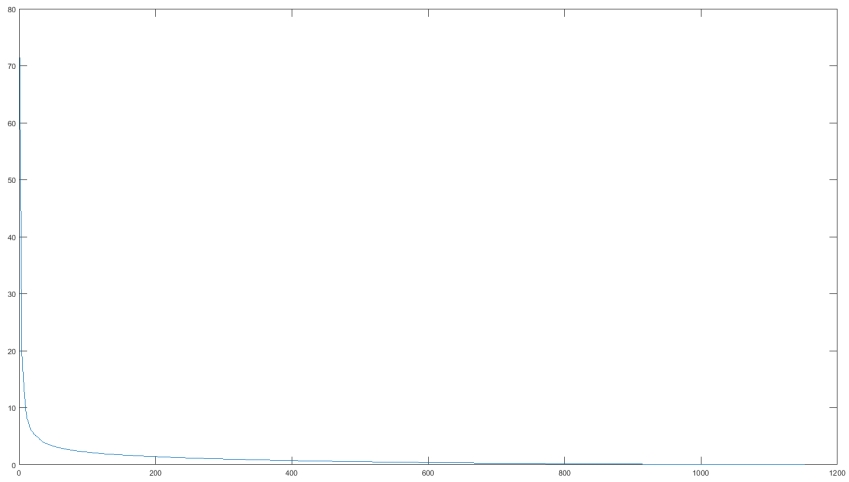


Figure 6: Evolution des valeurs propres

On visualise en 3D la dispersion des points pour les composantes 1, 2 et 3, car il est impossible d'afficher plus de dimensions.

On observe que les points sont séparés dans l'espace, ce qui signifie qu'il est possible de dessiner une ligne de décision qui sépare les deux classes.

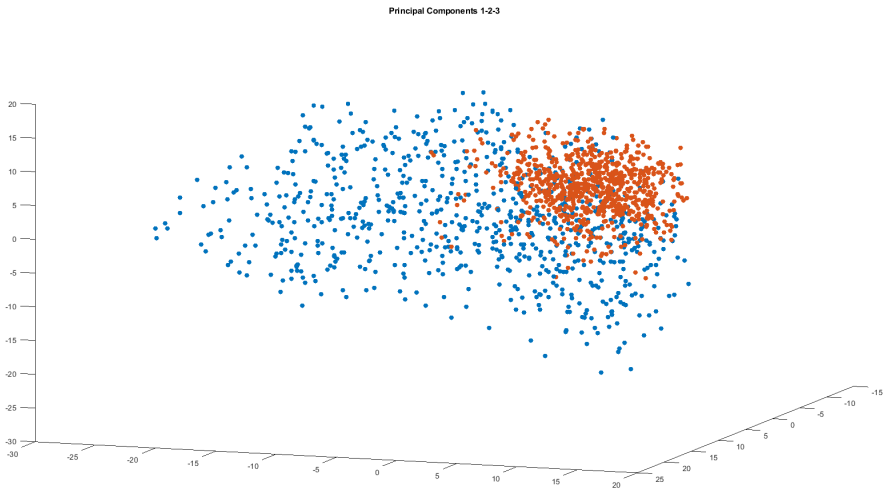


Figure 7: Visualisation de la dispersion des points pour les composantes 1, 2 et 3

2.4.1 Résultat

Pour évaluer notre modèle de classification, nous avons réduit la dimensionnalité des données à l'aide de l'analyse en composantes principales (PCA) en conservant les 309 premières directions. La sélection de ce nombre de composantes principales est basée sur les valeurs propres triées par ordre décroissant, on pourra aussi utiliser la méthode du coude.

On utilise a fonction classify cherche la courbe quadratique qui sépare les deux ensembles.

Nous obtenons les résultats suivants pour le nombre de faux positifs et le nombre de faux négatifs :

```
FP = 8
FN = 4
```

Pour finir, nous ajoutons d'autres indicateurs couramment utilisés pour évaluer la performance du modèle, à savoir l'accuracy (précision globale), la precision (précision), le recall (rappel) et le F1-score :

```
accuracy = (TP + TN) / length(labels);
precision = TP / (TP + FP);
recall = TP / (TP + FN);
f1_score = 2 * (precision * recall) / (precision + recall);
```

Les résultats obtenus sont :

```
Accuracy: 0.99
Precision: 0.99
Recall: 0.99
F1 Score: 0.99
```

Enfin, nous traçons la matrice de confusion pour visualiser les performances de classification.

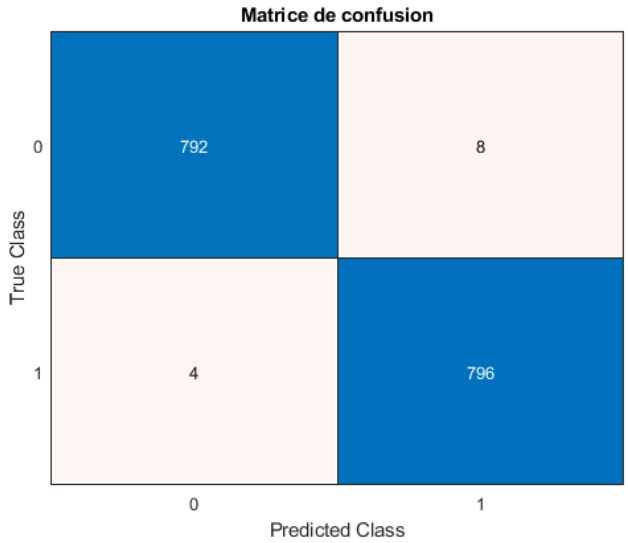


Figure 8: Matrice de confusion

Pour les résultats, bien que nous ayons obtenu de bons résultats en termes de précision et de accuracy, il est important de noter que nous avons testé notre modèle sur le même ensemble de données que celui utilisé pour l'entraînement. Cela peut entraîner un biais optimiste dans l'évaluation des performances du modèle.

3 Conclusion

Nous avons réussi, à l'aide de MATLAB et de l'algorithme des Histogrammes de gradients orientés (HOG), à construire un modèle de détection de piétons à partir de zéro. En utilisant le principe d'extraction des caractéristiques des images, nous avons pu démontrer l'efficacité de cette approche pour la tâche de classification des images. Cependant, cette approche présente des limitations. En effet, elle repose principalement sur des caractéristiques prédéfinies et ne s'adapte pas automatiquement aux variations complexes des données.

C'est pourquoi, dans les applications pratiques et à grande échelle, d'autres approches basées sur l'apprentissage automatique, telles que les réseaux de neurones et les réseaux de neurones convolutifs (CNN), sont souvent privilégiées. Ces méthodes ont prouvé qu'elles sont plus performantes dans de nombreux cas en raison de leur capacité à apprendre directement à partir des données, sans nécessiter de définition explicite des caractéristiques. Les CNN, en particulier, sont capables de capturer des motifs complexes et hiérarchiques dans les images, ce qui en fait des outils puissants pour les tâches de reconnaissance d'images et de détection d'objets.