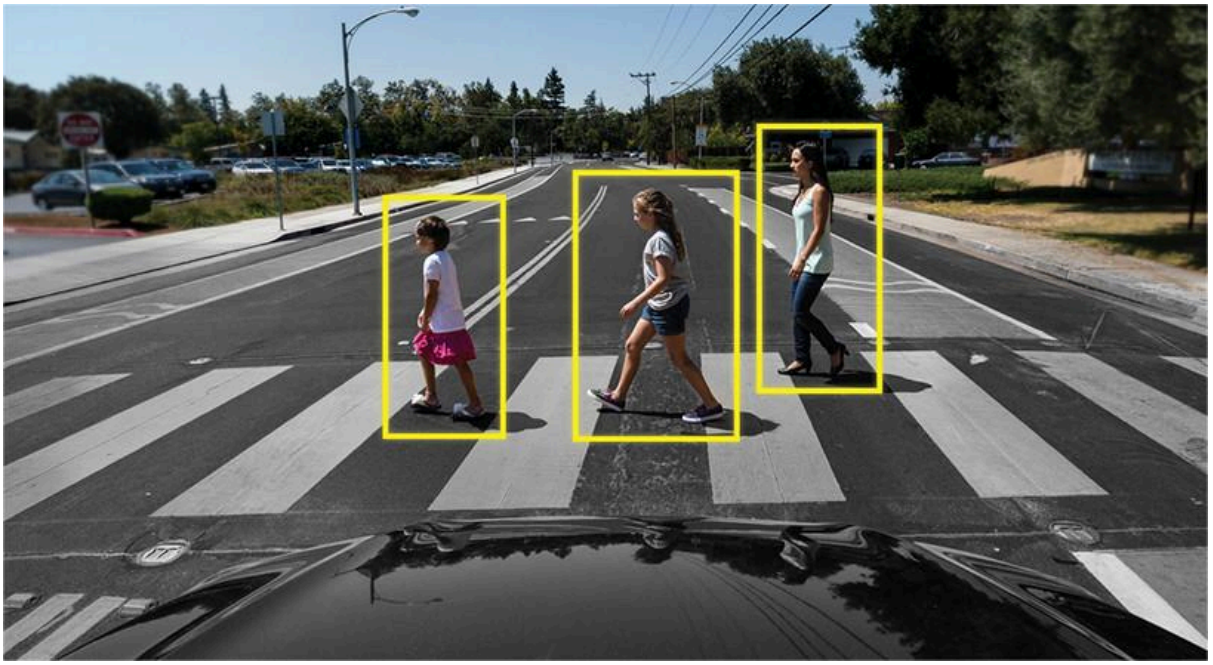
Bureau d’études - Détection des piétons
Compte-rendu



Elizaveta KOMPANIETS
Elève Ingénieure de deuxième année
Filière Énergie, Promo 2025

Tuteurs: Ranta RADU, Steven LE CAM, Didier MAQUIN, Maya KALLAS-EDELSON

Sommaire

Introduction

Ce bureau d'études vise le développement d'une méthode de détection de piétons à partir d'images fixes, pouvant à terme faire partie d'un système de vision embarquée sur véhicule autonome.

Nous travaillerons avec le logiciel MATLAB et sur une base de données mise à disposition par Dalal et Triggs, auteurs de la publication intitulée *Histograms of Oriented Gradients for Human Detection*, datant de 2005. Cette base contient plusieurs milliers d'images avec présence (image positive) ou absence (image négative) d'individus/piétons. Une partie de la base des images positives a été traitée et normalisée pour fournir des ensembles d'apprentissage et de test sur les images positives. Ces images, de taille 64 par 128 pixels, contiennent des individus en leur centre, permettant un apprentissage efficace sur des silhouettes variées. Nous utiliserons également la partie de la base des images non normalisées.

Nous allons donc réaliser une extraction de caractéristiques des images fournies par la méthode des HOG (Histograms of Oriented Gradients). Ensuite, nous réduisons ces caractéristiques par l'Analyse des Composantes Principales (ACP) afin d'améliorer l'efficacité du traitement ultérieur. Une fois les données extraites, nous évaluerons les performances pour la classification en piéton/non-piéton des différentes approches étudiées pendant le bloc d'études B8-10 "Intelligence artificielle, analyse de données et méthodes d'apprentissage" à l'ENSEM, à savoir: les approches par modélisation probabiliste des données (méthode de Parzen), les SVM linéaires ou non linéaires, et les réseaux de neurones. Enfin, nous testerons ces modèles sur les images réelles, non normalisées.

Partie 1: Extraction de caractéristiques de l'image

Dans cette partie, nous cherchons à coder une fonction qui réalise la technique d'extraction de caractéristiques HOG. Cette technique se base sur la décomposition d'une image en segments (appelés cellules) et la compilation d'histogrammes de gradients orientés pour ces cellules. Elle permet de capturer les formes locales et les contours en analysant les orientations et les normes des gradients de luminance à travers l'image. C'est pour cette raison que les caractéristiques HOG sont utiles pour la reconnaissance de formes et la détection d'objets dans les images, notamment des piétons.

Pour chaque cellule, nous allons calculer le gradient pour chaque pixel, ainsi que sa norme et son orientation, et nous rangeons ces gradients dans des histogrammes selon leur orientation. Ces histogrammes sont ensuite normalisés pour améliorer la robustesse de la méthode face aux variations d'éclairage et de contraste. Les histogrammes de toutes les cellules sont combinés pour former le vecteur de caractéristiques HOG.

a. Application des HOG sur une image

Nous commençons par coder l'algorithme des HOG sur Matlab que nous testerons sur une des images avec un piéton de la base de données. Nous allons ensuite utiliser ce script pour écrire une fonction réalisant cette technique pour l'appliquer à toutes les images. Tout d'abord, nous chargeons l'image choisie sur Matlab à l'aide de la fonction imread et la convertissons en noir et blanc à l'aide de la fonction *rgb2gray*.

Figure 1: affichage de l'image originale et l'image en noir et blanc

Afin de réaliser les HOG, nous devons effectuer un filtrage passe-haut de l'image, en calculant le gradient de chaque pixel sur l'ensemble de l'image par convolution avec le noyau de filtrage [-1 0 1]. Pour préparer le calcul de la norme du gradient par la suite, nous allons filtrer l'image de deux façons différentes : pour la première, nous procédons ligne par ligne, et pour la deuxième, colonne par colonne. Pour cela, nous utilisons la fonction *imfilter* avec le paramètre *'conv'* pour effectuer la convolution entre l'image et le filtre. Nous utilisons également la fonction *double* pour obtenir une précision double des valeurs du gradient. Nous obtenons le résultat suivant:

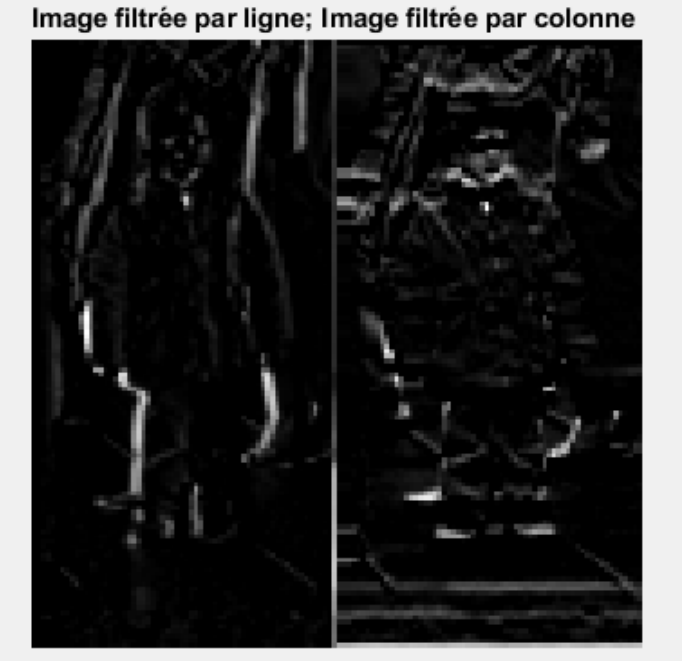


Figure 2: affichage de l'image filtrée par ligne et par colonne

Nous remarquons que le résultat du filtrage n'est pas le même selon la façon dont cela a été réalisé. Nous pourrions même dire que les deux images obtenues sont 'complémentaires' l'une de l'autre, et que leur somme aurait donné le contour de l'image.

Une fois que nous avons calculé le gradient en chaque point de l'image, nous procédons au calcul de la norme et de l'orientation de chacun de ces gradients. Pour cela, nous calculons la norme en prenant en paramètres les deux résultats des deux filtrages réalisés auparavant. De même, pour le calcul de l'orientation, nous utilisons la fonction *atan* sur le quotient des gradients des deux filtrages. Nous visualisons ensuite ces résultats.

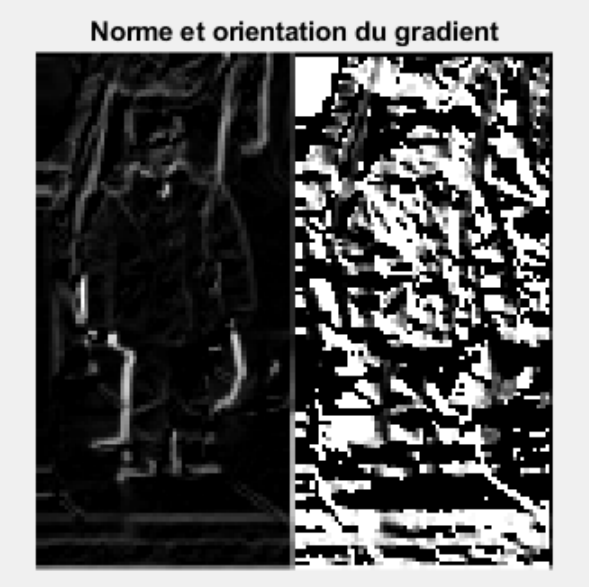


Figure 3: affichage de la norme et de l'orientation du gradient

Nous pouvons enfin réaliser l'histogramme de gradients orientés de l'image choisie. Pour effectuer ce calcul, nous extrayons d'abord les dimensions de l'image grâce à la fonction size et nous définissons la taille de chaque cellule HOG que nous fixons à 8, ainsi que le nombre de colonnes dans l'histogramme que nous fixons à 9.

L'image est découpée en cellules de 8x8 pixels grâce à une boucle qui la parcourt. À l'intérieur de chaque cellule, nous calculons la norme et l'orientation du gradient pour chaque pixel, que nous stockons ensuite dans deux vecteurs respectifs. Pour chaque orientation possible, délimitée par des intervalles spécifiques en radians, nous identifions les pixels dont l'orientation tombe dans cet intervalle. Nous sommons ensuite les normes des gradients correspondants pour obtenir la valeur de chaque colonne de l'histogramme.

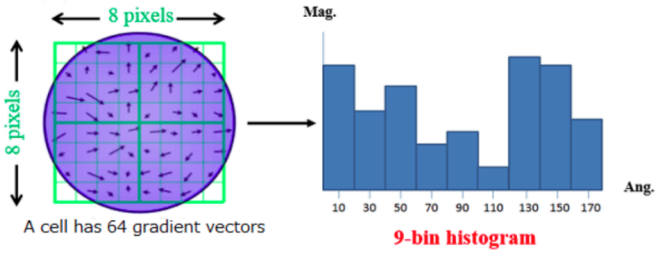


Figure 4: visualisation du calcul des HOG pour une cellule

Ce processus est répété pour toutes les orientations définies, permettant ainsi de remplir l'histogramme pour chaque cellule. Une fois toutes les cellules traitées, nous concaténons leurs histogrammes pour former l'histogramme HOG final de l'image, sous forme de vecteur, que nous utiliserons par la suite. Nous pouvons visualiser ces HOG:

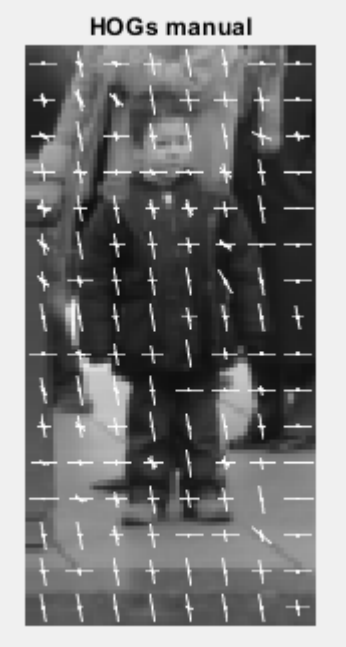


Figure 5: visualisation des HOG de l'image

Chaque segment blanc sur l'image indique la orientation prédominante du gradient dans une cellule de l'image. Nous observons un grand nombre de lignes verticales dans les zones correspondant aux contours du corps de la personne, ce qui indique les variations de l'intensité de l'image décrivant la silhouette du piéton.

b. Application de la fonction des HOG sur l'ensemble des images

En se basant sur les étapes de l'application des HOG sur une image, nous voudrions maintenant appliquer cette méthode sur l'ensemble d'images d'entraînement et de test.

Pour cela, nous créons la fonction *hogfeatures* qui réalise le calcul des HOG qui prend en argument l'image en noir en blanc *Im_g*, le filtre du gradient *filt*, le nombre de cases de l'histogramme *N* et la taille de la cellule *T* et qui retourne les caractéristiques extraites de l'image *Im_g*. Avec cette fonction, nous pouvons réaliser les HOG plus efficacement sur l'ensemble d'images dont nous disposons.

Dans un premier temps, nous définissons des paramètres d'entrée pour la fonction *hogfeatures* - nous choisissons le même filtre du gradient, le même nombre de cases de l'histogramme et la même taille de la cellule HOG que dans la partie 1.a, et nous fixons le nombre de caractéristiques voulues à 1152.

Nous récupérons ensuite la base de données d'apprentissage - pour cela, nous utilisons la fonction *dir* qui charge les données sur Matlab et qui prend en argument le chemin d'accès de ces données. Nous récupérons et nous appliquons l'algorithme des HOG séparément sur les images positives et les images négatives.

Pour chacun des types d'images nous créons ensuite des tableaux de caractéristiques que nous remplissons par des caractéristiques issues de la fonction *hogfeatures.* Ensuite nous parcourons

les images chargées une par une à l'aide de la fonction *imread* qui prend en argument le chemin spécifié par la combinaison du dossier et du nom de fichier et séparé par le séparateur du chemin du fichier filesep. Nous n'oublions pas de passer en double précision à l'aide de la fonction *double* et de normaliser normaliser les valeurs des pixels pour qu'elles soient comprises entre 0 et 1 en les divisant par 255. Nous convertisseur ensuite l'image en noir et blanc et nous lui applquons enfin la fonction *hogfeatures* pour stocker ensuite ses caractéristiques dans le tableau de caracéristiques crée auparavant.

Nous créons ensuite un unique tableau des caractéristiques (pour les images positives et négatives) en concaténant les deux tableaux d'avant, et nous créons également le vecteur des labels dont les composantes valent 1 pour les images avec piétons et 0 pour les images sans piétons. Ce vecteur nous permettra de réaliser un apprentissage supervisé dans le cas des données d'apprentissage et de vérifier la performance de nos modèles dans le cas des données de test.

Nous procéderons de la même manière pour la récupération des caractéristiques des images de test.

Partie 2: Application de l'Analyse en Composantes Principales (ACP)

Dans cette partie, nous continuons de préparer les données pour l'apprentissage et pour le test des différents modèles en leur appliquant l'analyse en composantes principales (ACP). Cette analyse nous permet de simplifier les caractéristiques HOG pour garder uniquement les composantes les plus significatives, réduisant ainsi la dimension des données et facilitant leur traitement dans la suite de l'étude.

L'algorithme de l'ACP est composé de plusieures étapes:

- Centralisation et normalisation des données
- Calcul de la matrice de corrélation des données et de ses vecteurs propres (composantes principales)
- Obtention des nouvelles données par multiplication de la matrice des données centrées et normalisées par la matrice des composantes principales.

Nous avons déjà coder cet algorithme lors du TD d'Introduction à la reconnaissance des formes, nous l'appliquons donc à des caractéristiques HOG des images d'entraînement.

Nous cherchons à déterminer le nombre de composantes principales que nous allons retenir pour la suite de notre étude, autrement dit la dimension de l'espace ACP. Pour cela, nous étudions les valeurs propres de la matrice de covariance en les visualisant avec la fonction *plot* de Matlab. Nous obtenons le résultat suivant:

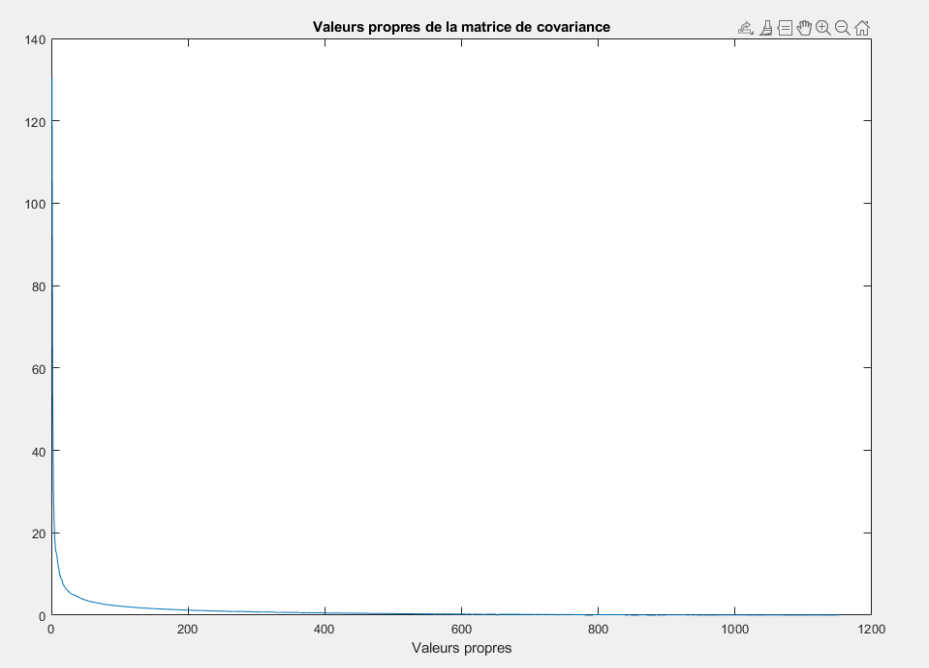


Figure 6: tracé des valeurs propres de la matrice de covariance

Nous disposons de plusieurs critères pour déterminer la dimension de l'espace ACP:

- D'après la règle du coude, la dimension de l'espace ACP est autour de 30.
- Pour évaluer le critère de Kaiser, nous cherchons le nombre de valeurs propres supérieures à 1. Nous en trouvons environ 241.
- Évaluons maintenant le critère de la variance expliquée: d'après la figure 6, nous supposons que la somme de toutes les valeurs propres de la matrice de covariance vaut environ 150. Avec ceci, la variance expliquée cumulée à environ 90% correspond à environ 30 valeurs propres.

Après avoir évaluer ces différents critères, nous décidons de retenir pour notre étude 30 valeurs propres, et donc **nous choisissons l'espace de l'ACP de dimension 30**, car ceci correspond approximativement à un nombre de valeurs propres donné par le critère de coude et de la variance expliquée cumulée.

Visualisons les caractéristiques des données d'apprentissage traitées par l'ACP sur le plan de trois premières composantes principales:

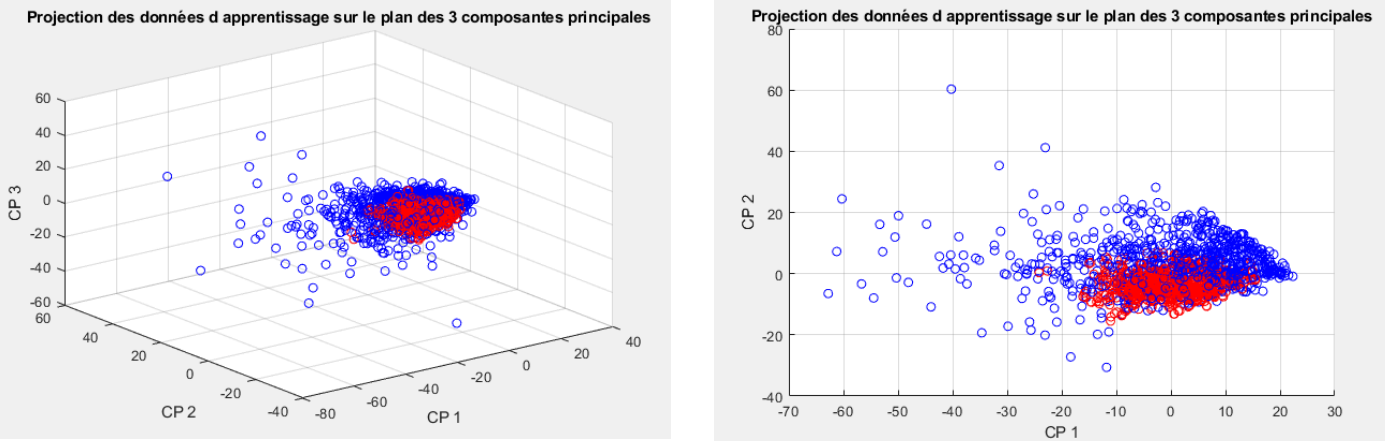


Figure 7: visualisation des données d'apprentissage sur un plan 3D et 2D des composantes principales

Nous distinguons bien deux types de données - les données positives en rouge et les données négatives en bleu. Contrairement aux données positives qui sont regroupées, les données négatives semblent d'être plus dispersées sur le plan des composantes principales, ce qui peut indiquer une plus grande variabilité ou une moindre cohérence interne de ces données. Cela pourrait indiquer que les images sans piétons sont plus variées et que donc leurs caractéristiques sont influencées par un plus grand nombre de facteurs.

Nous appliquerons la même ACP sur les données de test afin de les préparer à la phase de classification.

Partie 3: Apprentissage et classification des données de test par la méthode de Parzen

a. Présentation de la méthode

Dans cette partie, nous voudrions utiliser la méthode de Parzen pour classifier les images de test. L'idée de cette méthode est d'estimer la densité de probabilité en un point donné en examinant combien d'autres points de données se trouvent à proximité, dans un certain 'volume' autour de ce point, défini par un noyau dite de Parzen.

b. Phase d'entraînement

Pour la phase d'apprentissage, nous commençons par séparer la matrice des caractéristiques HOG des images d'entraînement en deux matrices comprenant les données d'entraînement correspondant à chacune des classes.

Nous définissons aussi l'écart-type du noyau de Parzen que nous utiliserons dans le calcul des densités de probabilité de vraisemblances à chacune des classes. Il s'agit d'une variable que nous allons modifier pour trouver celle qui permet d'avoir un modèle le plus performant.

c. Phase de test (Classification)

Dans la phase de classification des données de test, nous allons calculer les probabilités d'appartenance de chaque image de test à chacune des classes. Cette étape est réalisée par la fonction *gaussParzen,* codée lors du TD Reconnaissance des formes, qui prend en argument les données de test, les données d'apprentissage correspondante à une des classes et l'écart-type de Parzen et qui donne en sortie la probabilité d'appartenance à cette classe . Nous allons stocker ensuite ces probabilités dans une matrice. Nous allons ensuite chercher le maximum des deux probabilités d'appartenance pour chaque image - avec ceci, nous pouvons connaître la classe qui a été attribuée à cette image.

d. Phase de validation

Dans la phase de validation, nous voudrions savoir si notre classifieur est performant ou pas. Pour cela, nous calculons le pourcentage d'erreurs commises lors de la classification des données de test.

Voici l'évolution du pourcentage d'erreur en fonction de l'écart-type de Parzen choisie lors de la phase de classification:

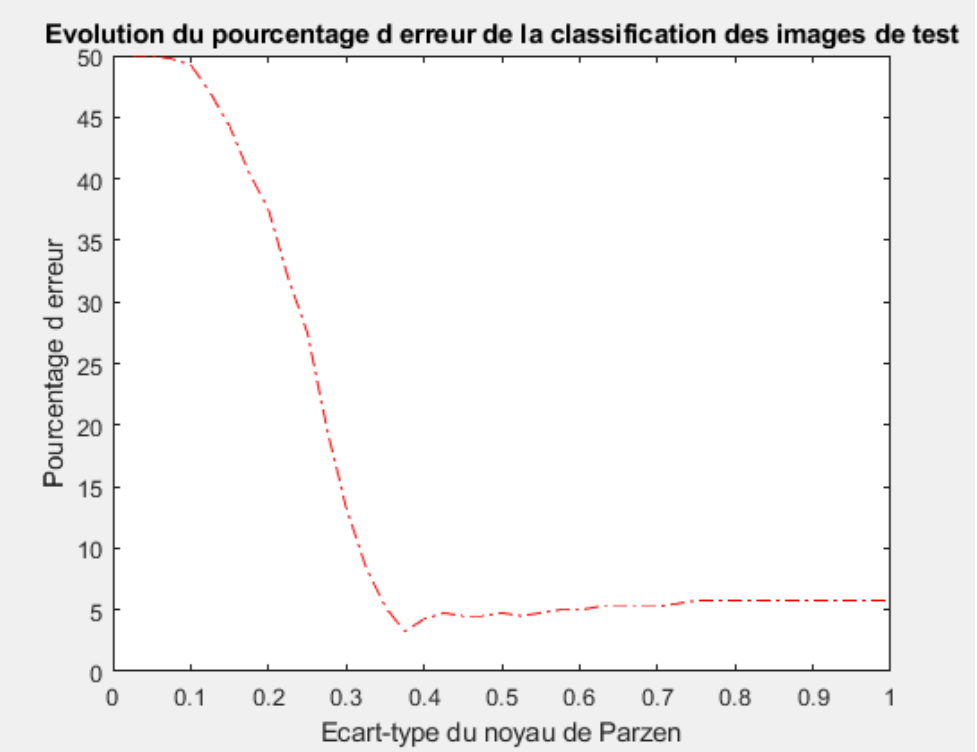


Figure 8: tracé de l'évolution du pourcentage d'erreur en fonction de l'écart-type du noyau de Parzen

Nous remarquons que le pourcentage d'erreur sur la classification des images de test décroît significativement à mesure que l'écart-type du noyau de Parzen augmente, jusqu'à atteindre une valeur stable autour de 0.37. Ce comportement suggère que l'augmentation de l'écart-type du noyau de Parzen jusqu'à ce point permet une meilleure estimation de la densité de probabilité, résultant en une réduction des erreurs de classification. Cependant, au-delà de cette valeur optimale, l'erreur de classification ne diminue plus et se stabilise, ce qui indique qu'un noyau de Parzen plus grand n'apportent pas d'amélioration significative.

Pour déterminer la valeur exacte de l'écart-type optimal , nous utilisons la fonction *min* pour récupérer la valeur et l'indice de l'erreur minimale et ensuite nous récupérons l'écart-type correspondant à ce pourcentage d'erreur minimal qui sera donc l'écart-type optimale. Nous obtenons le résultat suivant:

```
err_minimale =          sig_optimal =

    3.2500                  0.3750
```

Figure 9: valeurs du taux d'erreur minimale et de l'écart-type optimal

Nous fixons donc l'écart-type à 0.375. Nous définissons maintenant la matrice de confusion, qui a pour diagonale le nombre des images positives et négatives bien classées (vrais positifs et vrais négatifs) et hors diagonales les nombres d'images mal classées (faux positifs et faux négatifs). Nous obtenons donc la matrice suivante:

```
conf =

   193     6
     7   194
```

Figure 10: matrice de confusion

Nous remarquons que le modèle a mal classé 7 images sans piétons et 6 images avec piétons.

Partie 4: Apprentissage et classification des données de test par le SVM à noyau gaussien

a. Présentation de la méthode

Dans cette partie, nous voudrions utiliser le séparateur à vaste marge à noyau gaussien pour classifier les images de test. L'idée de cette approche est de trouver un hyperplan qui sépare au mieux les différentes classes dans un espace avec la plus grande marge possible. Le noyau gaussien nous permet de gérer les cas où les frontières de séparation sont non linéaires (comme dans le cas des images avec/sans piétons) en adaptant la forme de l'hyperplan aux caractéristiques plus complexes des données.

b. Phase d'entraînement

Pour la phase d'apprentissage, nous allons utiliser la fonction *fitcsvm* qui permet d'entraîner un modèle SVM sur les données d'apprentissage. Dans notre cas, cette fonction prend en argument la matrice des données réduites par l'ACP et le vecteur de labels de ces données; nous précisions que le noyau à utiliser et le noyau gaussien (*"KernelFunction", "rbf"*) et que les paramètres de modèle doivent être déterminés automatiquement par Matlab afin d'améliorer la performance du modèle *("OptimizeHyperparameter","auto")*.

c. Phase de test (Classification)

Afin de classer les images de test avec le SVM défini dans la sous-partie d'avant, nous utilisons la fonction *predict* qui prend donc en argument le modèle SVM ainsi que les données de test et qui retourne les labels prédites pour ces données.

d. Phase de validation

Dans cette sous-partie, nous voulons savoir si le modèle établi est performant ou pas. Nous voulons donc savoir le taux d'erreur sur la prédiction des classes des données de test. Nous comptons donc le nombre d'images mal classées et nous les divisons par le nombre total d'images de test. Nous obtenons le résultat suivant:

```
err =

    0.7500
```

Figure 11: taux d'erreur de la classification par le modèle SVM

La classification des images de test par le modèle SVM à seulement 0.75% d'erreurs, ce qui est un résultat …. .

Etudions la matrice de confusion de cette classification:

```
conf =

   200     3
     0   197
```

Figure 12 : matrice de confusion de la classification par le modèle SVM

Le modèle a donc bien classées toutes les images de test sans piétons et a mal classées seulement 3 images avec piétons.

Partie 5: Apprentissage et classification des données de test par un réseau de neurones de type feedforward

a. Introduction

Dans cette partie, nous voudrions réaliser la classification des images de test en utilisant un réseau de neurones de type feedforward. Ce type de réseau de neurones fonctionne en prenant les données d'entrée, qui dans ce cas sont les caractéristiques des images, et en les passant à travers plusieurs couches cachées. Chaque couche applique une transformation linéaire suivie d'une fonction d'activation non linéaire, ce qui permet au réseau d'apprendre des représentations des données. Lors de l'entraînement, le réseau ajuste les poids des connexions entre les neurones pour minimiser l'erreur entre les prédictions du réseau et les labels réels des données d'entraînement, et une fois entraîné, il peut prendre des caractéristiques des images de tests à travers les couches du réseau, et produire des prédictions de classe pour chaque image.

Nous allons appliquer le réseau de neurones sur les caractéristiques brutes des HOG et ensuite sur les caractéristiques réduites avec l'ACP et nous comparerons la performance et le temps de calcul pour les deux types de données d'entrée.

Nous commençons par créer un réseau de neurones feedforward à en utilisant la fonction *feedforwardnet* de Matlab. Pour notre étude, nous allons définir quatres réseaux avec un nombre de couches et de tailles différentes. A l'issue de la phase d'apprentissage, nous choisirons le réseau le plus performant. Nous allons donc prendre:

- un réseau composé d'une seule couche de taille 3
- un deuxième réseau composé d'une couche de taille 6
- un troisième réseau composé de deux couches de taille 3 et 6 respectivement
- un quatrième réseau composé de trois couches de taille 3, 6 et 9 respectivement

Pour préparer l'apprentissage du réseau, nous divisons les données d'entrée en un ensemble d'apprentissage et un ensemble de validation. Nous choisissons les proportions suivantes: 80%, 20%.

b. Apprentissage du réseau sur les caractéristiques des HOG

1. Phase d'entraînement du réseau

Dans un premier temps, nous entraînons nos réseaux de neurones sur les caractéristiques HOG brutes, avant la réduction par l'ACP. Nous réalisons donc l'apprentissage de nos réseaux sur ces données et nous évaluons leurs performances à l'aide de la fonction *perform*. Visualisons ces performances:

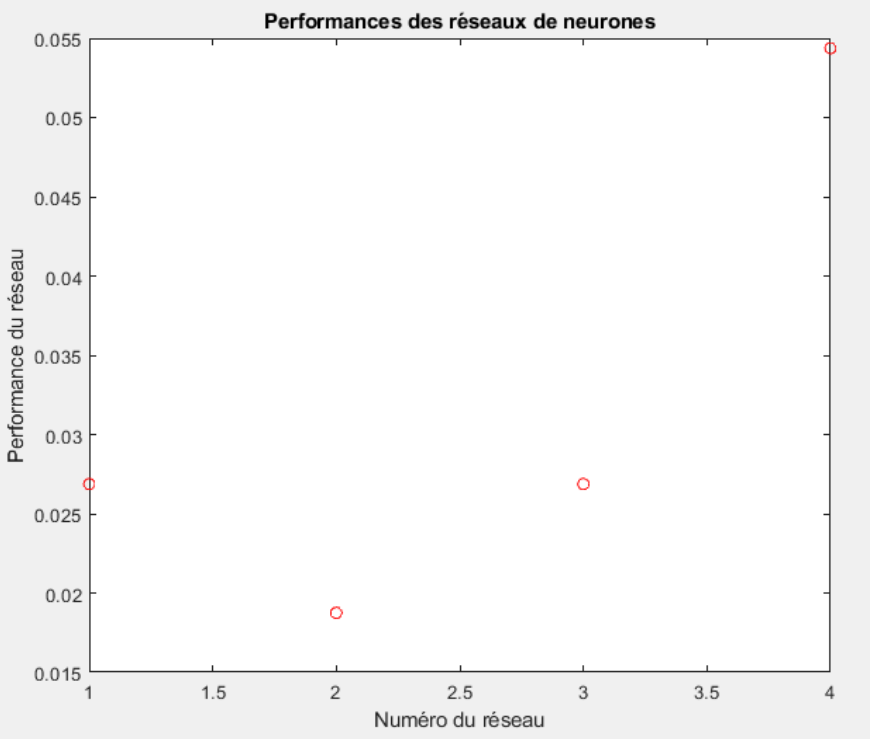


Figure 13: évaluation des performances pour chaque réseau entraîné sur les données HOG

Nous constatons donc que le réseau 2 est le plus performant des quatres car possède la plus petite performance qui vaut 0.0187. D'après la figure 13, nous pouvons aussi en déduire que dans le cas de cette étude, l'augmentation de taille des couches améliore la performance du réseau et, au contraire, l'augmentation du nombre des couches diminue sa performance. Cette baisse de performance dans le cas du réseau 3 peut être due au phénomène de surapprentissage quand le modèle devient trop spécifique aux données d'apprentissage et se généralise mal aux données de validation.

Nous choisissons donc **le réseau 2** pour la classification sur les caractéristiques HOG.

2. Phase de test (Classification)

Pour la phase de test, nous appliquons le réseau choisi sur les caractéristiques HOG des images de test. Nous obtenons en sortie un vecteur de labels prédites par le réseau. Cependant, ce vecteur est composé des nombres réels proches de 0 et 1 et non des entiers 0 et 1 comme voulu. Pour y remédier, nous 'filtrons' ce vecteur en mettant 0 si la composante est inférieure à 0.5 et en mettant 1 dans le cas contraire.

3. Phase de validation de la classification

Dans la phase de validation , nous voulons déterminer le taux d'erreur de classification. Pour cela, nous calculons le nombre d'images mal classées en comparant des vrais labels des images de test et les labels prédites par le réseau et nous divisons cette somme par le nombre total d'images de test. Nous obtenons le résultat suivant:

```
err =

    4.7500
```

Figure 14: taux d'erreurs sur la classification selon les caractéristiques HOG

Le réseau a donc mal classé 4,75% des images de test. Visualisons la matrice de confusion de cette classification:

```
conf =

   189     8
    11   192
```

Figure 15: matrice de confusion

Nous remarquons donc que le réseau a plus de faux positifs et que de faux négatifs, ce qui veut dire que le réseau classe mieux les images avec des piétons que de sans piétons.

c. Apprentissage du réseau sur les caractéristiques réduites par l'ACP

1. Phase d'entraînement du réseau

Dans cette sous-partie, nous réalisons l'apprentissage de nos réseaux de neurones sur les caractéristiques HOG réduites par l'ACP. Comme dans 5.b.1), nous appliquons nos réseaux sur cette base d'apprentissage et nous évaluons leurs performances pour déterminer le réseau le plus performant. Nous obtenons le résultat suivant:

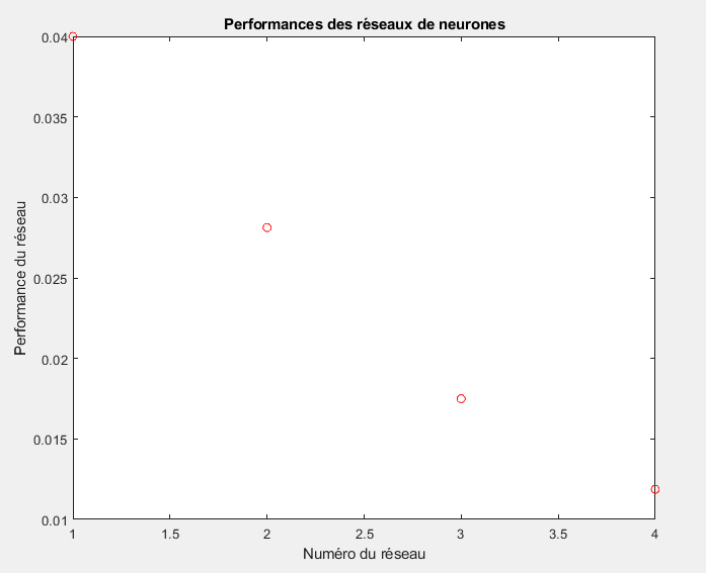


Figure 16: évaluation des performances pour chaque réseau entraîné sur les données ACP

Nous constatons donc que contrairement au cas où les réseaux sont entraînés sur les données HOG (cf figure 13), le réseau entraîné sur les données ACP avec la meilleure performance est le réseau 4, avec la performance qui vaut 0.011. Nous pouvons aussi en déduire que dans le cas d'apprentissage sur les données réduites, l'augmentation du nombre de couches, comme la taille des couches, semble améliorer la performance du modèle. Ceci est dû à la capacité des réseaux plus profonds et plus larges de capturer des abstractions complexes et des nuances subtiles dans les données. L'ACP, en réduisant la dimension des données, permet au réseau de concentrer son apprentissage sur les aspects les plus significatifs des données, ce qui améliore l'efficacité de l'apprentissage et réduit potentiellement le phénomène de sur-apprentissage.

Nous choisissons donc **le réseau 4** pour la classification sur les caractéristiques HOG réduites par l'ACP.

2. Phase de test (Classification)

Nous procédons de la même manière que dans la partie 5.a.2. en appliquant le réseau entraîné sur les données de test réduites par l'ACP et en normalisant le vecteur des labels prédites.

3. Phase de validation de la classification

Nous voudrions savoir le taux d'erreurs de classification avec ce modèle. Nous obtenons le résultat suivant:

```
err =

    3.5000
```

Figure 17: taux d'erreurs sur la classification selon les caractéristiques ACP

Nous remarquons que entraîné sur les données ACP, le réseau prédit mal seulement 3,50% des images. C'est donc 1,25% de plus d'images bien classées comparé au modèle entraîné sur les caractéristiques HOG brutes. Visualisons sa matrice de confusion:

```
conf =

   193     7
     7   193
```

Figure 18: matrice de confusion

Maintenant, le modèle classe particulièrement mieux les images sans piétons, mais il y a également une légère amélioration de la classification des images avec piétons.

Partie 6: Apprentissage et classification des données de test par un réseau de neurones convolutif

a. Introduction

Dans cette partie, nous utiliserons le réseau de neurones convolutif directement sur les images, sans passer par les caractéristiques HOG. Ce type de réseau est spécialement conçu pour traiter des données structurées en forme de grille, comme les images. Son fonctionnement repose sur plusieurs types de couches, comme les couches de convolution, les couches de pooling, les couches de normalisation et etc.

Pour optimiser le codage de cette partie, nous commençons par effacer toutes les variables stockées suite à l'exécution des codes pour les parties précédentes, car elles ne nous seront pas nécessaires dans cette sous-partie.

Nous chargeons séparément les bases de données des images d'entraînement et de test sur Matlab à l'aide de la fonction *imageDatastore* qui prend en argument le chemin d'accès des dossiers avec des images, et qui inclut des sous-dossiers qui séparent les données positives et négatives et qui prend comme l'étiquette des données les noms des sous-dossiers dans lesquelles elles sont stockées.

Nous définissons ensuite l'architecture de notre réseau de neurones convolutif. Pour notre étude, nous allons définir cinq réseaux avec des architectures différentes. A l'issue de la phase d'apprentissage, nous choisirons le réseau le plus performant que nous utiliserons dans la classification. Nous allons donc prendre:

- un réseau convolutif proposé par le sujet du TD 2 "Réseaux de neurones"
- un deuxième réseau dans laquelle nous ajoutons un ensemble de couches [convolution (filtre de taille 16), normalisation, ReLU, Max-pooling]
- un troisième réseau qui reprend le réseau 2 et dans laquelle nous ajoutons un ensemble de couches [convolution (filtre de taille 32), normalisation, ReLU, Max-pooling]
- un quatrième réseau qui reprend le réseau 3 dans laquelle nous ajoutons un ensemble de couches [convolution (filtre de taille 64), normalisation, ReLU, Max-pooling]
- un cinquième réseau qui reprend le réseau 4 dans laquelle nous ajoutons un ensemble de couches [convolution (filtre de taille 64), normalisation, ReLU, Max-pooling]

Pour toutes les architectures, nous indiquons dans la couche *imageInputLayer* la taille des images d'apprentissage qui est 128x64x3 pixels Nous indiquons également dans la couche *fullyConnectedLayer* le nombre de classes en sortie qui vaut 2.

Après avoir défini les réseaux, nous précisons les options d'apprentissage avec la fonction *trainingOptions*. Nous utilisons la descente de gradient stochastique avec inertie (SGDM) avec un taux d'apprentissage initial de 0,01, en fixant le nombre maximum d'époques à 6, nous mélangeons les données à chaque époque, calculons la précision sur les données de validation à intervalles réguliers de fréquence 10, en désactivant la sortie de la fenêtre de commande et en activant le tracé de la progression de l'entraînement.

Pour préparer l'application du réseau, nous divisons les données d'apprentissage en: ensemble d'apprentissage (*trainData*) et ensemble de validation (*valData*). Pour cela, nous utilisons la fonction *splitEachLabel* dans laquelle nous indiquons les proportions des images attribuées à chaque ensemble. Pour notre étude, nous choisissons les proportions suivantes: 80%, 20%.

b. Phase d'entraînement du réseau convolutif

Nous passons enfin en phase d'entraînement de nos réseaux convolutifs sur les images de test à l'aide de la fonction *trainNetwork* qui prend en argument les données d'apprentissage, l'architecture du réseau et les options d'apprentissage. Nous calculons ensuite le précision de chacun des cinq réseaux en appliquant la fonction *classify* sur les données de validation et ensuite nous calculons la précision globale de chaque réseau en divisant le nombre d'images bien classées sur le nombre d'images total (nous aurions pu récupérer cette précision appelle *'Validation accuracy'* dans la fenêtre de progression de l'entraînement de chaque réseau). Nous visualisons ensuite les précisions pour chaque réseau:

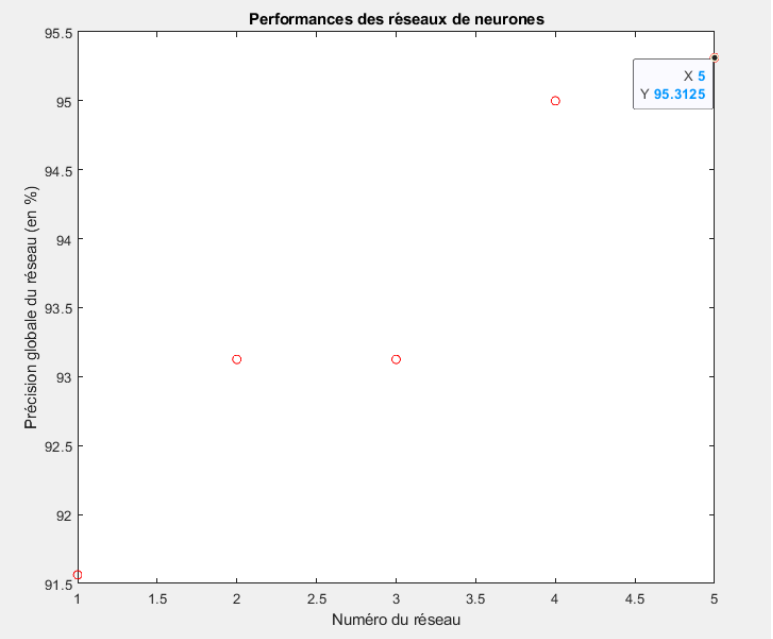


Figure 17: évaluation des performances pour chaque réseau convolutif

Nous remarquons que le réseau 1 a une très faible précision comparé aux autres, ce qui suggère qu' une seule couche de convolution n'est pas suffisante pour avoir une précision de validation satisfaisante. Cependant, le réseau avec la plus grande précision (plus de 95%) est le réseau 5, donc le réseau avec le plus grand nombre de couches. Nous pouvons donc en déduire que l'augmentation du nombre de couches, et en particulier de l'ensemble de couches [convolution, normalisation, ReLU, Max-pooling] augmente la performance du réseau.

Pour le reste de la partie, nous ne considérons que le **réseau 5**.

c. Phase de test (Classification)

Nous utilisons le réseau choisi dans la classification des images de test. Pour cela, nous utilisons la fonction *classify* qui prend en argument le réseau choisi et les données de test.

d. Phase de validation de la classification

Nous voulons connaître le taux d'erreur sur la classification faite par le réseau de neurones convolutif. Nous obtenons le résultat suivant:

```
err =

    1.7500
```

Figure 19: taux d'erreurs sur la classification par un réseau de neurones convolutif

Ce taux d'erreurs indique que le modèle est particulièrement performant car ne classe mal que 1,75% d'images de tests sans passer par les caractéristiques HOG. Visualisons maintenant la matrice de confusion de cette classification:

```
conf =

   193     0
     7   200
```

Figure 20: matrice de confusion

Nous pouvons constater que le réseau convolutif a bien classé tous les images avec piétons et mal classé seulement 7 images sans piétons.

Partie 7: Comparaison des performances et des temps de calcul pour les différentes modèles

Dans cette partie, nous allons comparer les performances des modèles établies avec les différentes méthodes des parties précédentes.
Pour cela, nous allons utiliser les critères de performance suivants:

- Précision globale ou accuracy: taux d'images de test classées dans les bonnes catégories.
- Temps de calcul: temps pris par le modelé pour la préparation des données, l'apprentissage du modèle et la classification des données de test.
- Sensitivité: taux d'images positives (avec piétons) bien classées parmi les images positives.
- Précision: taux d'images positives (avec piétons) bien classées parmi les images classées positives.
- F-mesure: mesure combinée de la précision et de la sensibilité, utilisée pour évaluer l'équilibre entre ces deux métriques
- Spécificité: taux d'images négatives (sans piétons) bien classées parmi les images négatives.

Ces critères de performances pour les différentes modèles sont résumés dans le tableau ci-dessus:

Modèle établi par...		Critères de performance du modèle					
		Accuracy (en %)	Temps de calcul (en s)	Sensibilité (en %)	Précision (en %)	F-mesure (en %)	Spécificité (en %)
Méthode de Parzen		96,75	24,49	97	96,52	96,76	96,5
SVM à noyau gaussien		99,25	52,33	98,5	100	99,25	100
Réseau de neurones	Feedforward sur les HOG	95,25	342,77	96	94,58	95,29	94,5
	Feedforward sur les ACP	96,5	24,58	96,5	96,5	96,5	96,5
	Convolutif sur les images	98,25	46,69	100	96,62	98,28	96,5

Étudions ces résultats plus en détails dans les sous-parties suivantes.

a. Accuracy

D'après le tableau ci-dessus, nous pouvons constater que tous les modèles ont une précision globale de plus de 95%. Cette performance élevée indique que chaque modèle, malgré ses différences en termes de méthodologies, est bien adapté à la détection des piétons sur les images.

Nous pouvons constater que le modèle établi par le **SVM à noyau gaussien** présente la meilleure précision globale, de 99,25%. Cependant, le **réseau de neurones convolutif** a également une haute précision globale, de 98,25%. Ce sont donc les deux modèles qui classent le mieux des images dans leurs classes respectives.

Etudions maintenant la précision globale du **réseau de neurones feedforward** sur les données HOG brutes et celle réduites par l'ACP. Nous remarquons qu'après la réduction de la dimension des caractéristiques par l'ACP ce réseau a une plus haute précision globale (96,50%) que le réseau entraîné et appliquée sur les données HOG brutes (95,25%). Ceci montre que la réduction de la dimension des caractéristiques par l'ACP permet d'améliorer les performances du réseau de neurones car en réduisant la complexité des données d'entrée, l'ACP élimine le bruit et caractéristiques non pertinentes, ce qui facilite l'apprentissage du modèle.

b. Temps de calcul

D'après le tableau, nous pouvons constater que le temps de calcul est très différent selon les modèles établies. Par exemple, le **réseau de neurones feedforward sur les HOG** a le temps de calcul le plus élevé (342,77 s). Ceci est dû au très grand nombre de caractéristiques (1152) qui sont pris en entrée du réseau, ce qui le rend moins pratique. Cependant, les méthodes les plus rapides sont celle de **Parzen** (24,49 s) et du **réseau de neurones feedforward sur les ACP** (24,58 s). Ce sont donc les modèles les plus efficaces en termes de calcul.

c. Sensibilité, Précision et F-mesure

De même que pour la précision globale, tous les modèles ont une haute sensibilité et une haute précision (plus de 96% pour la sensibilité et 94% pour la précision). Ceci indique que les modèles sont non seulement capables de détecter correctement les images positives (images avec piétons) avec une faible proportion de faux négatifs, mais aussi de prédire avec une grande exactitude les images positives parmi celles qu'ils ont classées comme telles, réduisant ainsi les faux positifs. De même, la moyenne harmonique de la précision et de sensibilité (F-mesure) dépasse 95% pour tous les modèles, ce qui signifie qu'ils ont un bon équilibre entre les faux positifs et les faux négatifs.

En termes de sensibilité, le meilleur résultat appartient au **réseau de neurones convolutif**, qui a pu bien détecter 100% d'images avec piétons. Ce résultat est particulièrement important car le coût de faux négatifs (images avec piétons mais classées comme celles sans piétons) est important dans le contexte de détection des piétons par les véhicules autonomes, car manquer la présence d'un piéton peut entraîner des accidents graves, mettant en danger la vie des piétons et des passagers. Comme dans le cas de la précision globale, le deuxième meilleur résultat appartient au **SVM à noyau gaussien** qui a également une excellente sensibilité (98,5%), et il est suivi par le modèle entraîné par la **méthode de Parzen** (97%).

Cependant, le **SVM à noyau gaussien** a la meilleure précision parmi tous les modèles qui est de 100%. Ceci indique que chaque prédiction de présence de piétons par ce modèle est correcte, sans aucun faux positif, ce qui pourrait également être bénéfique pour le système de détection des piétons par des véhicules autonomes. Tous les autres modèles, sauf le réseau de neurones entraînés sur les caractéristiques HOG, ont une très haute précision qui dépasse 96%.

En termes d'équilibre entre la classification de faux positifs et de faux négatifs, la meilleure F-mesure appartient au **SVM à noyau gaussien,** avec 99,25%, et au r**éseau de neurones**

convolutif, avec 98,28%, ce qui en fait les modèles les plus performants en termes de qualité de prédiction

d. Spécificité

D'après le tableau, nous pouvons constater que tous les modèles ont une haute spécificité de plus de 94%, ce qui signifie que ces modèles sont très efficaces pour classer correctement les images sans piétons, sans générer de faux positifs.

Le meilleur résultat appartient au **SVM à noyau gaussien**, qui a la spécificité de 100%, ce qui veut dire que chaque fois que le **SVM à noyau gaussien** prédit qu'une image ne contient pas de piétons, cette prédiction est toujours correcte. Tous les autres modèles, sauf le réseau de neurones entraînés sur les caractéristiques HOG, ont une très haute spécificité qui est de 96,5%.

e. Conclusion

D'après la comparaison des différents critères de performance, nous pouvons conclure que le **SVM à noyau gaussien** se démarque par ses excellentes performances, comme haute précision, sensibilité, F-mesure et spécificité, avec un temps de calcul raisonnable. Cependant, le **réseau de neurones convolutif** est aussi très performant, particulièrement en termes de sensibilité et de F-mesure, tout en ayant un temps de calcul assez bas. Le modèle le moins performants parmi ceux étudiés est **le réseau de neurones feedforward qui utilise les caractéristiques HOG brutes**, car non seulement il a les performances les plus basses (qui restent toujours hautes dans l'absolu), mais aussi le temps de calcul qui est extrêmement grand comparé à ceux des autres modèles.

Conclusion

Durant ce bureau d'études, nous avons développé et évalué plusieurs modèles pour la détection de piétons à partir d'images fixes et normalisées. Le processus a commencé par l'extraction des caractéristiques des images à l'aide de la méthode des HOG (Histograms of Oriented Gradients), suivie de la réduction de la dimension des données par l'Analyse en Composantes Principales (ACP). Nous avons ensuite appliqué différents modèles de classification, notamment la méthode de Parzen, les SVM à noyau gaussien et les réseaux de neurones feedforward et convolutifs.

Les résultats ont montré que tous les modèles avaient une précision globale supérieure à 95%, avec le SVM à noyau gaussien et le réseau de neurones convolutif qui se distinguait par leurs performances exceptionnelles. Le SVM à noyau gaussien a atteint une précision et une spécificité parfaites de 100%, tandis que le réseau de neurones convolutif a montré une sensibilité de 100%, ce qui est particulièrement important pour ne pas manquer les piétons. Le réseau feedforward entraîné sur les données HOG réduites par ACP ont montré une amélioration notable des performances par rapport aux données HOG brutes, soulignant l'importance de la réduction de la dimension des données. En termes de temps de calcul, les méthodes de Parzen et le réseau feedforward sur les ACP se sont avérées les plus efficaces. Le réseau feedforward sur les HOG brutes, bien que précis, a montré un temps de calcul trop élevé pour être pratique pour les applications en temps réel.

Pour l'avenir, il serait pertinent de tester ces modèles sur des images non normalisées afin d'évaluer leur robustesse et leur performance dans des conditions plus variées et réalistes. Cette étape a été prévue dans le cadre de ce bureau d'études mais je n'ai malheureusement pas eu le temps de le réaliser suite au manque du temps.