

Projet incertitude des réseaux de neurones GIE2 2019-2020

Table des matières

I)	Remise dans le contexte par rapport au premier semestre	4
1)	Répartition du travail au premier semestre	4
2)	Résultat du premier semestre	4
3)	Objectifs de ce second semestre	4
II)	La préparation des données	5
1)	Création d'un fichier exploitable par python et donc par un programme de machine learning.	5
2)	Création des X_train et des Y_train.....	6
3)	Un petit "nettoyage des données"	6
4)	Une nécessité de data augmentation	7
5)	La transformation en coordonnées de Grasping	8
III)	Le réseau de neurones	9
IV)	Les résultats en sortie du réseau	10
1)	L'entraînement du réseau	10
2)	Résultats sans data augmentation	11
3)	Résultats avec data augmentation.....	12
V)	L'analyse des performances de nos résultats.....	14
1)	Rappel du premier semestre	14
2)	Le critère de Jacquard	14
3)	La mise en place d'une méthode d'évaluation des incertitudes sur les données de validation : Le monte Carlo Drop Out	15
VI)	Prendre une décision.....	16
VII)	Conclusion générale	18



1) Remise dans le contexte par rapport au premier semestre

1) Répartition du travail au premier semestre

Au premier semestre, nous avons travaillé de manière séparé et asynchrone sur le projet. Jean s'était chargé d'une étude précise sur les différents types de regression. Leopold s'était intéressé aux différentes méthodes de Grasping et à la base de données de Cornell. Jeffrey lui s'était intéressé aux méthodes d'évaluation des incertitudes dans les predictions d'un reseau de neurones.

2) Résultat du premier semestre

Nous étions arrivé à différentes méthodes d'évaluation des incertitudes(regression et classification) et des méthodes de Grasping et à une esquisse de programme sur la base de données de Cornell. Nous en étions arrive à une démarche pour traiter le problème des rectangles de grasping.

3) Objectifs de ce second semestre

- Nettoyer et preparer les données de la base de données de cornell
- Mettre en place un réseau convergent
- Adapter les prédictions du réseau à des coordonnées compréhensibles par le réseau
- Mettre en place des méthodes d'évaluation des incertitudes sur les prédictions du réseau

II) La préparation des données

1) Création d'un fichier exploitable par python et donc par un programme de machine learning

- **Récupération des données:** Il a fallu d'abord récupérer les fichiers dans la base de données de Cornell. Les fichiers sont regroupés comme expliqué ci-dessous en image avec son rectangle de grasping associé(Quatre coordonnées dans le plan cartésien).

The dataset file contains 5 types of files:

1. Image files

Named pcdxxxxr.png
where xxxx ranges from 0000-1034
These are the original images of the objects

2. Point cloud files

Named pcdxxxx.txt
where xxxx ranges from 0000-1034

3. Handlabeled grasping rectangles

Named pcdxxxxcpos.txt for positive rectangles
Named pcdxxxxcneg.txt for negative rectangles

4. Background images

Named pcd_b_XXXX.png

5. A mapping from each image to its background image

Named backgroundMapping.txt

















=====

2. Point cloud files

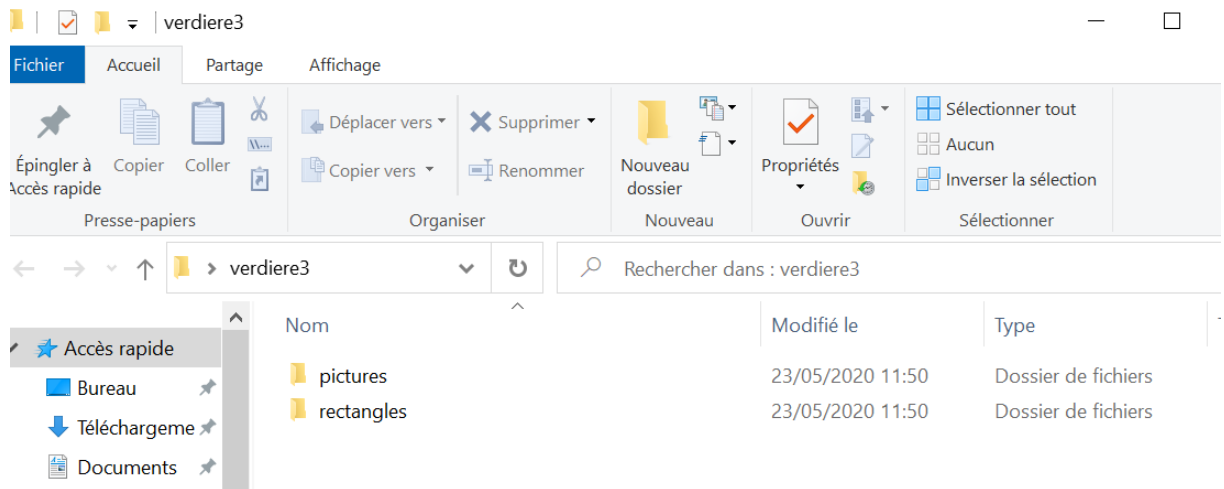
Point cloud files are in .PCD v.7 point cloud data file format

See <http://www.pointclouds.org/documentation/format7.html> for details.

- **Traitement des données récupérées:** On récupère alors les données sous forme de dix fichiers sur le bureau. Les fichiers sont alors des alternances de fichier png et texte contenant des bons et des mauvais rectangles de grasping. Il faut alors créer un fichier contenant toutes les photos et un autre avec les rectangles de grasping associé.

		 pcd050cpos	16/10/2013 21:27	Document texte
 pcd050r			16/10/2013 21:27	Fichier PNG
 pcd051cpos	 pcd051r		16/10/2013 21:27	Document texte
 pcd052cpos	 pcd052r		16/10/2013 21:27	Document texte
 pcd053cpos	 pcd053r		16/10/2013 21:27	Document texte
 pcd054cpos	 pcd054r		16/10/2013 21:27	Document texte
 pcd055cpos	 pcd055r		16/10/2013 21:27	Document texte
 pcd056cpos	 pcd056r		16/10/2013 21:27	Document texte

- **Traitement de fichiers:** Par les outils classiques des traitements de fichier sur python, on regroupe toutes les images et tous les rectangles dans un meme dossier(fichier verdier3). Cela nous permettra de créer par la suite nos X-train et Y-train.



2) Création des X_train et des Y_train

- On crée une liste d'image et de rectangle de grasping X_train et Y_train et on garde seulement 100 données de validation.

```
B+= [b]
Y1[i,k]=float(a)/13.3333333 #=640/48
Y1[i,k+1]=float(b)/7.5 #=480/64
```

3) Un petit “nettoyage des données”

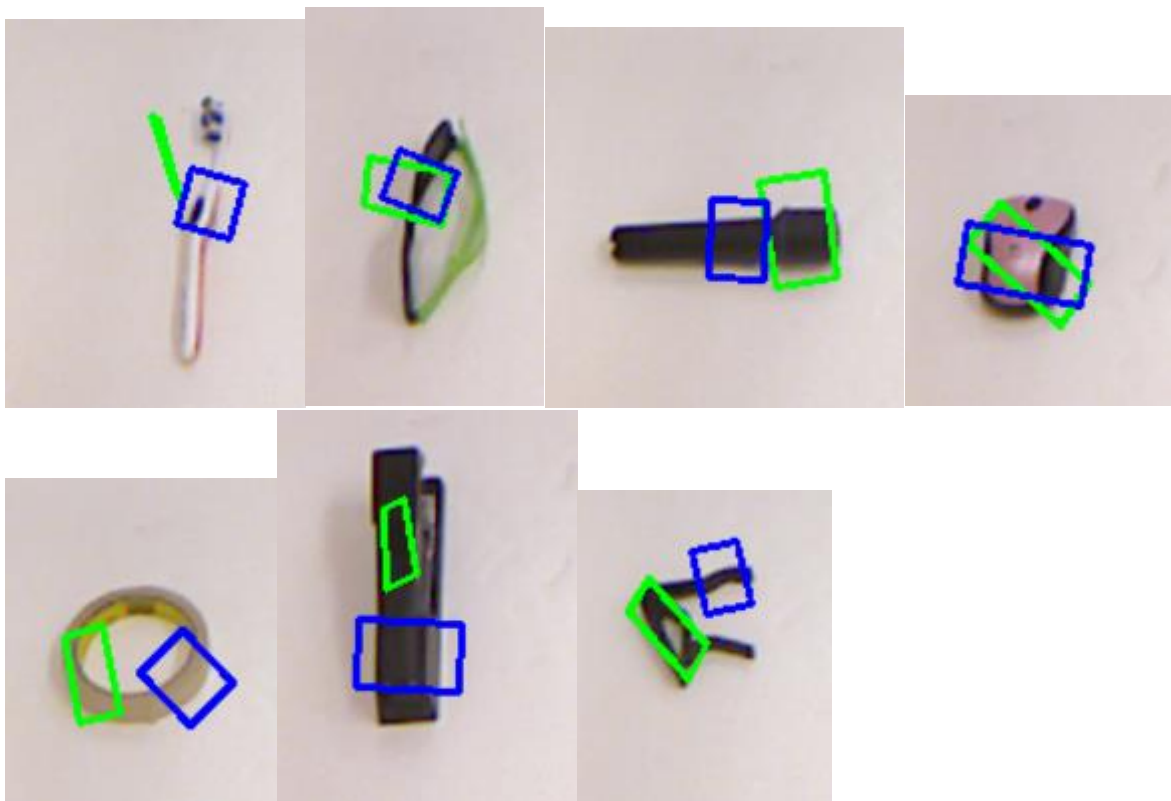
- **Eviter l'hétérogénéité des données:** On norme ensuite les pixels entre un nombre entre 0 et 1 en divisant par 255 pour éviter l'hétérogénéité des données.
- **On centre les images et les rectangles:** Il a tout d'abord fallu modifier les données brutes d'entrée. Les images que nous recevons de la base de données sont très grandes et contiennent un grand nombre de pixel. Les traiter tel quel serait bien trop lourd pour le réseau et pourrait mener à des erreurs. Nous avons donc décidé de les réduire de taille. Grace au module CV2 de python, et à l'aide de la fonction *resize*, nous avons divisé la taille des photos par 10. Nous les avons ensuite découpées de manière à ne garder que l'objet au centre, et à éliminer les bordures inutiles. Nous avons procédé pixel par pixel en vérifiant que

nous n'enlevons du contenu utile dans aucune photo. Il a ensuite fallu modifier les coordonnées des rectangles d'entraînement en conséquence.

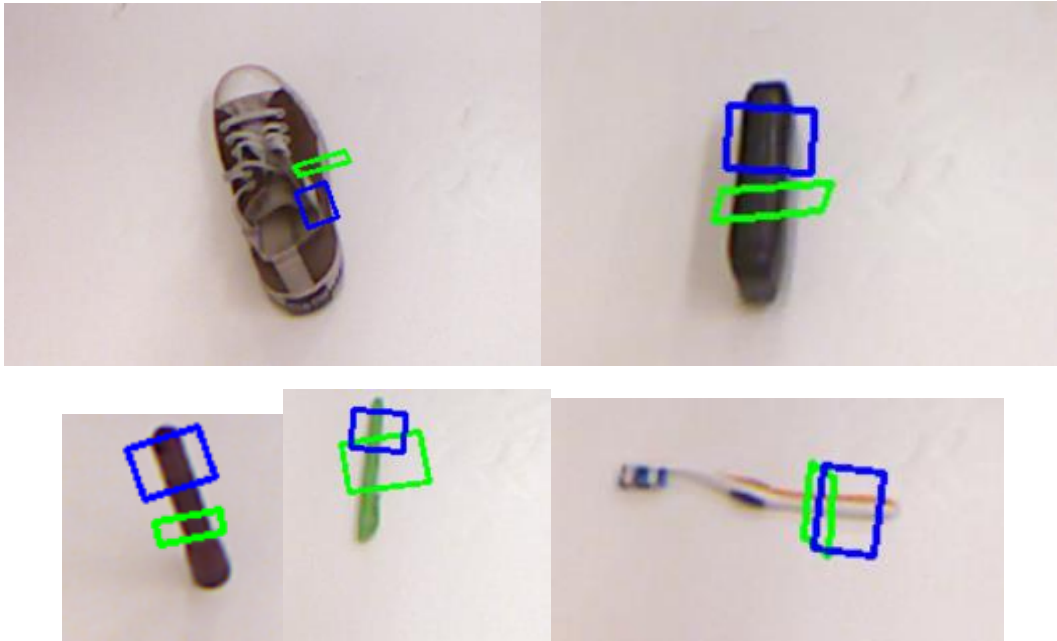
4) Une nécessité de data augmentation

On a effectué un certain nombre d'opération pour réaliser une augmentation du volume de données nécessaires à tout problème de machine learning.

On voit sur ces photos en vert les rectangles en sortie de réseau, et en bleu les rectangles que l'on souhaiterait obtenir.



On a ensuite ajouté de la data augmentation. Nous avons créé des nouvelles images qui contiennent les anciennes mais avec des symétries en plus. Ainsi, nous avons triplé nos datas en faisant deux symétries, une par rapport à l'axe x et l'autre par rapport à l'axe y. On obtient donc à la fin **2652 valeurs**, dont nous utiliserons **2552** pour l'entraînement et les **100** restantes pour le test. Nous obtenons, selon le même code couleur que précédemment, les rectangles suivants :

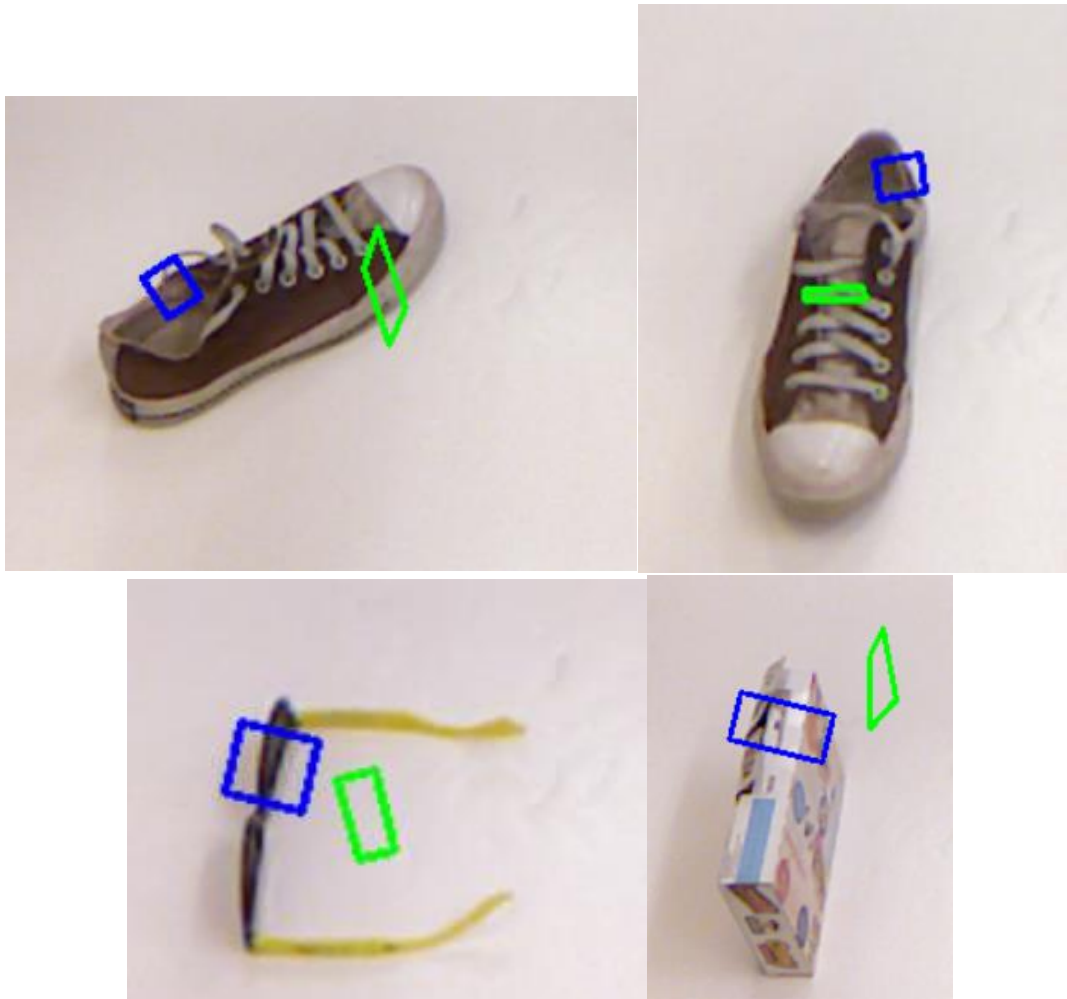


On peut voir ici que les rectangles obtenus sont très proches des rectangles bleus. Pour le cas de la chaussure, il était impossible pour le réseau, même si on l'a entraîné correctement, de deviner que le bas de la chaussure contient un trou, et que c'est donc ici qu'il faudrait prendre la chaussure.

5) La transformation en coordonnées de Grasping

Comme nous l'avons rappelé plus haut, il faut transformer les 8 coordonnées des polygones en coordonnées de grasping qui sont compréhensibles par le robot à savoir : point haut, point bas, la hauteur, la longueur et l'angle.

Cependant, il y a également des cas où le réseau a beaucoup plus de mal à déterminer le bon rectangle de grasping. Voici quelques exemples :



Encore une fois dans le cas des chaussures, on ne peut pas trop « reprocher » au réseau ses erreurs. Cependant, dans le cas de la paire de lunettes et du paquet de céréales. Dans le cas du paquet de céréales, sa couleur majoritairement blanche peut être la raison du décalage de la réponse du réseau. Dans le cas de la paire de lunettes cependant, nous n'avons trouvé aucune justification à cette réponse complètement décalée. Ainsi, comme nous n'avons pas accès au robot et que nous avons eu beaucoup de mal à spécifier les rectangles de grasping, nous avons préféré rester avec les 8 coordonnées cartésiennes.

III) Le réseau de neurones

- **Création du réseau :** Nous avons ensuite créé le réseau. Pour le traitement d'image, nous avons utilisé une architecture très commune avec des poids déjà réglés. Nous avons utilisé VGG16 qui étant déjà entraîné permet d'extraire des meilleures caractéristiques. Cela en est d'autant plus avantageux que nous sommes dans un problème de régression.

```
def build_model(mc=False):
    model=Sequential()
    model.add(VGG16(include_top=False, pooling='avg', weights='imagenet'))
    #model.add(Conv2D(2,kernel_size=(3,3),activation='relu',input_shape=(64,48,3)))
    #model.add(Conv2D(64,(3,3),activation='relu'))
    #model.add(MaxPooling2D(pool_size=(2,2)))
    #model.add(Dropout(0.25))
    model.add(Dense(1024,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(256,activation='relu'))
    model.add(Dense(128,activation='relu'))
    model.add(Dense(8))
    model.layers[0].trainable=False
    model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

- **Nécessité couche dense** : Notre réseau étant un réseau de régression et non un réseau de classification, le réseau se « termine » sur une couche dense et non sur une fonction d'activation.

IV) Les résultats en sortie du réseau

1) L'entraînement du réseau

L'entraînement du réseau: A l'aide de la fonction suivante, nous faisons ce qu'on appelle un entraînement du réseau.

```
mc_model .fit(X_train[100:],Y_train[100:],epochs=num_epochs,batch_size=100, verbose=1, callbacks=[history])
```

C'est une étape absolument centrale dans le traitement. Cette fonction va en fait attribuer les poids aux différents nœuds du réseau. C'est cela qui va déterminer quel pixel à le plus d'importance dans le résultat final, et lequel en a très peu.

Le programme tourne ensuite pendant un certain temps :

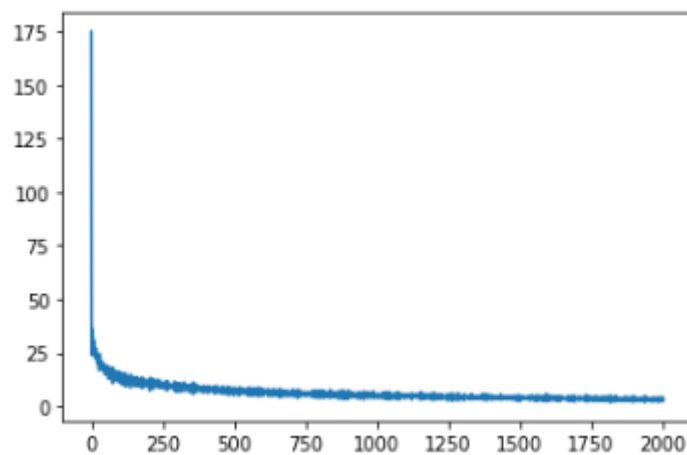
```
Epoch 1/2000
784/784 [=====] - 6s 8ms/sample - loss: 41846.4251 - mean_absolute_error: 185.7007
Epoch 2/2000
784/784 [=====] - 5s 6ms/sample - loss: 1581.4232 - mean_absolute_error: 31.3826
Epoch 3/2000
784/784 [=====] - 5s 6ms/sample - loss: 1597.5854 - mean_absolute_error: 31.6194
Epoch 4/2000
200/784 [=====>.....] - ETA: 4s - loss: 1766.4752 - mean_absolute_error: 33.6348
```

Cela peut prendre quelques heures en fonction de la complexité du réseau mais surtout en fonction du nombre d'époques et d'échantillons à traiter. La loss correspond, pour résumer, à l'erreur que le

réseau rencontre pour chaque époque. Si la loss diminue avec le nombre d'époques, on va dire que le réseau converge. Il est indispensable qu'il converge, et sa non-convergence serait une preuve d'erreur dans la conception du programme.

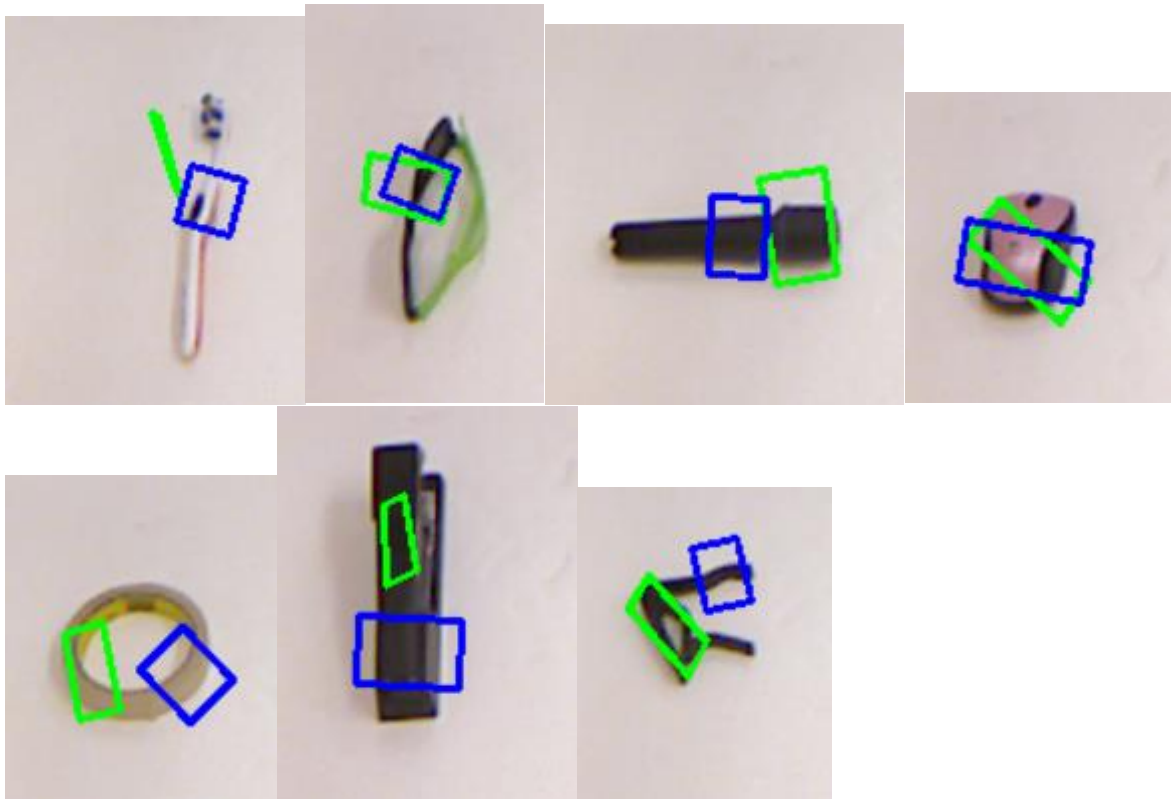
2) Résultats sans data augmentation

Comme nous vous l'avons précisé dans le rapport précédent, la data augmentation est une étape très importante pour la précision du réseau de neurone. Cela empêche aussi l'overfitting, ce qui est très courant dans ce genre de réseau et avec si peu de données d'entraînement. Sans overfitting, nous disposons de **884 valeurs**. Nous allons en utiliser **100** pour le test et **784** pour l'entraînement. Pour 2000 epochs, nous avons calculé les loss. Sur l'image ci-dessous, on voit bien que le réseau est convergent, et qu'après 2000 epochs, on arrive à un loss très faible. Il serait donc inutile de faire tourner le réseau plus longtemps. On est donc certains que sous sa forme actuelle, le réseau ne peut pas être amélioré par simple calcul additionnel.



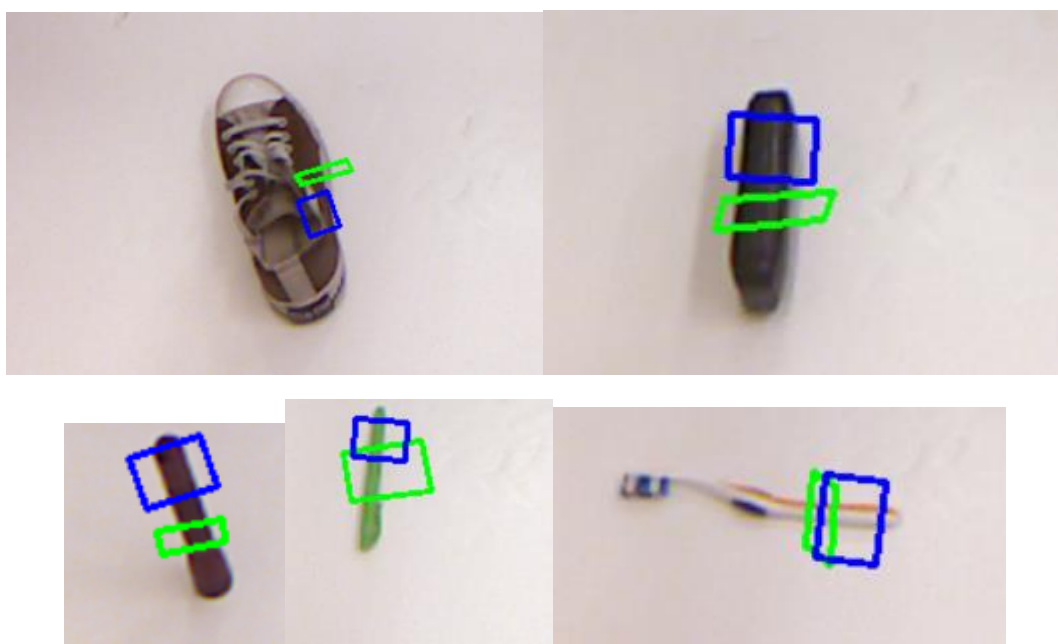
On voit donc bien ici que le réseau converge.

On voit sur ces photos en vert les rectangles en sortie de réseau, et en bleu les rectangles que l'on souhaiterait obtenir.



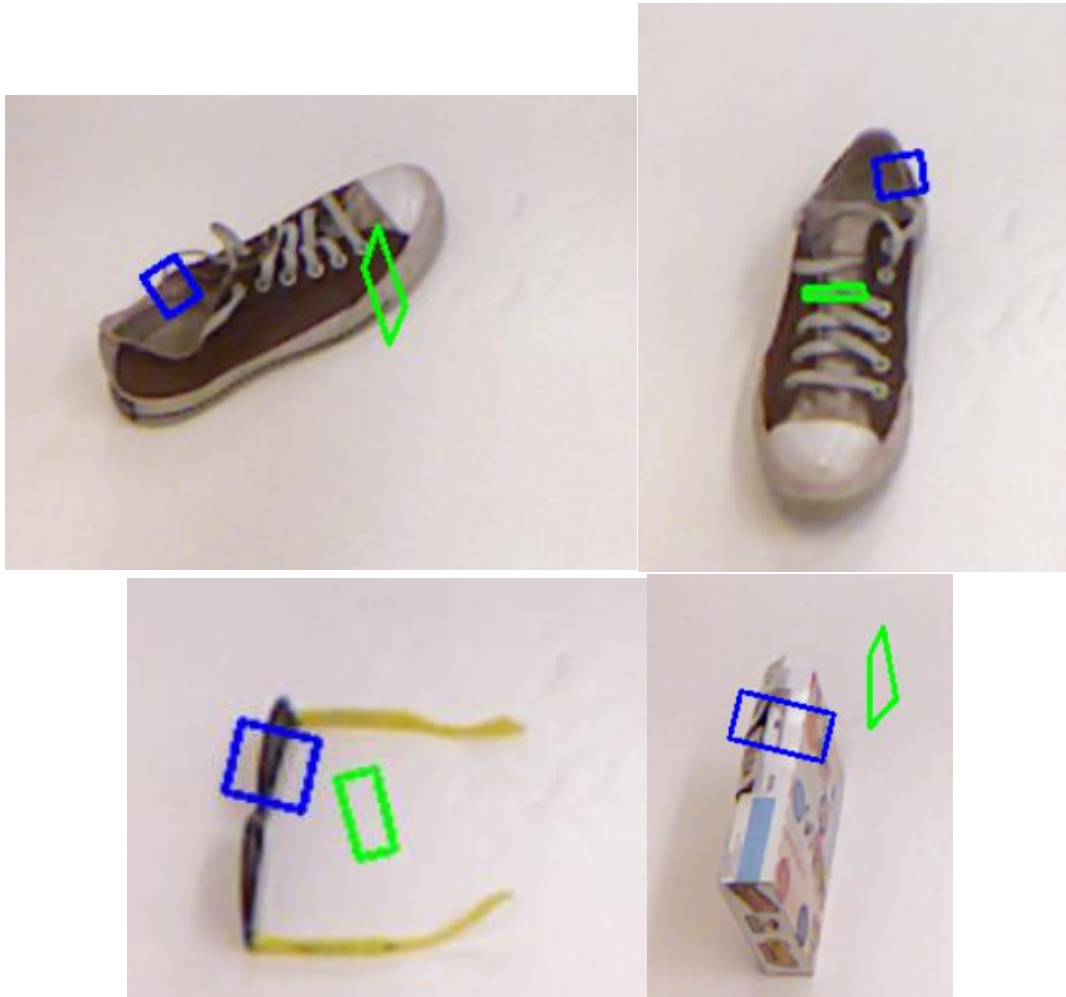
3) Résultats avec data augmentation

On a ensuite ajouté de la data augmentation. Nous avons créé des nouvelles images qui contiennent les anciennes mais avec des symétries en plus. Ainsi, nous avons triplé nos datas en faisant deux symétries, une par rapport à l'axe x et l'autre par rapport à l'axe y. On obtient donc à la fin **2652 valeurs**, dont nous utiliserons **2552** pour l'entraînement et les **100** restantes pour le test. Nous obtenons, selon le même code couleur que précédemment, les rectangles suivants :



On peut voir ici que les rectangles obtenus sont très proches des rectangles bleus. Pour le cas de la chaussure, il était impossible pour le réseau, même si on l'a entraîné correctement, de deviner que le bas de la chaussure contient un trou, et que c'est donc ici qu'il faudrait prendre la chaussure.

Cependant, il y a également des cas où le réseau a beaucoup plus de mal à déterminer le bon rectangle de grasping. Voici quelques exemples :



Encore une fois dans le cas des chaussures, on ne peut pas trop « reprocher » au réseau ses erreurs. Cependant, dans le cas de la paire de lunettes et du paquet de céréales. Dans le cas du paquet de céréales, sa couleur majoritairement blanche peut être la raison du décalage de la réponse du réseau. Dans le cas de la paire de lunettes cependant, nous n'avons trouvé aucune justification à cette réponse complètement décalée.

Nous avons ici utilisé la ligne de code suivantes afin de ne pas avoir à faire retourner le réseau à chaque petite modification. Le fichier model.h5 permet d'enregistrer les poids du réseau pour pouvoir les réutiliser plus tard. Cela s'est avéré très pratique dans le cadre de ce projet.

```
[36]: #utilise les poids d'un réseau déjà entraîné par moi  
mc_model.load_weights('model.h5')
```


V) L'analyse des performances de nos résultats

1) Rappel du premier semestre

Au premier semestre nous avons vu qu'il était possible d'évaluer les incertitudes des prédictions d'un réseau de neurones à partir de données de test. En effet, nous avons fait des tests sur deux bases de données : MNIST qui est problème de classification et Boston Housing Data Set qui est problème de regression. Nous avons appliqué la démarche d'évaluation des incertitudes utilisation l'inférence variationnelle(Monte Carlo Drop Out sur Keras) à notre étude sur la base de données de Cornell. D'autre part, nous avons utilisé la formule de Jacquard pour vérifier si nos prédictions sont bonnes ou mauvaises.

2) Le critère de Jacquard

Mise en place de l'indicateur de Jacquard : Il est évident qu'il est impossible pour un réseau de prédire « parfaitement » un rectangle de Grasping. Pour vérifier si une prédiction est bonne ou non, nous utilisons ce que l'on appelle le critère de Jacquard qui revient à calculer le rapport de l'aire de l'intersection des deux rectangles(vrais et prédits) et de l'union de l'aire des deux rectangles(vrais et prédits) comme sur la formule ci-dessous.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Nous avons pris la précaution de vérifier que les rectangles ne se « replient » pas sur eux-meme grâce à la fonction true_polygon. Dans le cas échéant, nous avons rajouter un zero pour le critère de Jacquard. Nous obtenons alors à la sortie du programme une liste avec le nombre de Jacquard pour les 100 images de validations.

```

print(1,p_pred,p_true)

if p_pred.is_valid:
    if p_pred.area + p_true.area - p_pred.intersection(p_true).area!=0:
        iou = p_pred.intersection(p_true).area / (p_pred.area + p_true.area - p_pred.intersectio
    else:
        iou=0
        s+=1
#L.append(iou)

```

Le critère de Jacquard : La littérature assure qu’une bonne prédiction est à l’origine d’un indicateur de jacard supérieur à 0,25. Néanmoins, au vu des faiblesses de notre réseau, nous nous contentons d’un indicateur différent de 0 pour asserter que la prédiction est bonne. En effet, comme vu précédemment, certains rectangles peuvent ne pas coïncider avec le vrai rectangle et que la prédiction soit tout à fait juste. Nous obtenons alors la matrice suivante :

```

[0.05473110358203798, 0.08165452180042604, 0.0883991387118623, 0.0011915573090475011, 0, 0.0, 0.0, 0.2984414887
29386, 0, 0.0, 0.0, 0.046493502052817784, 0.05451803192895302, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1
08695897487618, 0.016157781877993896, 0.07221754450555383, 0.16406597682200447, 0.05912986644707325, 0.0, 0.012
838126802217, 0.07450289292009098, 0.05548433162506585, 0.11359522985817931, 0.09366909410935438, 0.0, 0.0, 0.1
55016322894444, 0.0, 0.4586890403717131, 0.5598832187418933, 0.07264686690254897, 0.030141765335731093, 0.0, 0.
4031186382153654, 0.0, 0.0, 0.04398693802455926, 0.0, 0.0, 0.041055645905916795, 0.0011484211937396157, 0.0, 0.
0.43599454480853717, 0.0, 0.008838504810136105, 0.0, 0.0, 0.0, 0.0, 0.0, 0.15842760913655252, 0.015884001179178
7, 0.0, 0.0, 0.03144552409261193, 0.0, 0.0, 0.0, 0.13898529330509013, 0.3533646719910879, 0.7584591224058751, 0
5993353654278248, 0.2526550267739865, 0.10128862974319154, 0.1712861470384367, 0.0, 0.09801037601418879, 0.0, 0
28755531407491878, 0.25312158504118343, 0.0, 0.10329885849398064, 0.3085128569027862, 0.0, 0.2513314466063253,
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.019304171748194224, 0.0, 0.017598517722533704, 0.108388553058
58, 0.19096749865314805]

```

3) La mise en place d’une méthode d’évaluation des incertitudes sur les données de validation : Le monte Carlo Drop Out

Nous avons vu au premier semestre qu’il était possible d’utiliser le Drop Out pour évaluer des incertitudes. En effet, en laissant activer grace à la méthode ci-dessous une couche de drop out en phase de validation, on peut obtenir des masques de drop out. Ces masques de drop out qui correspondent à des tirages indépendants de réseaux(on gèle différents poids du réseau à chaque tirage indépendamment des autres) permettent d’obtenir des sorties sensiblement différentes.

```

metrics=['mae'])

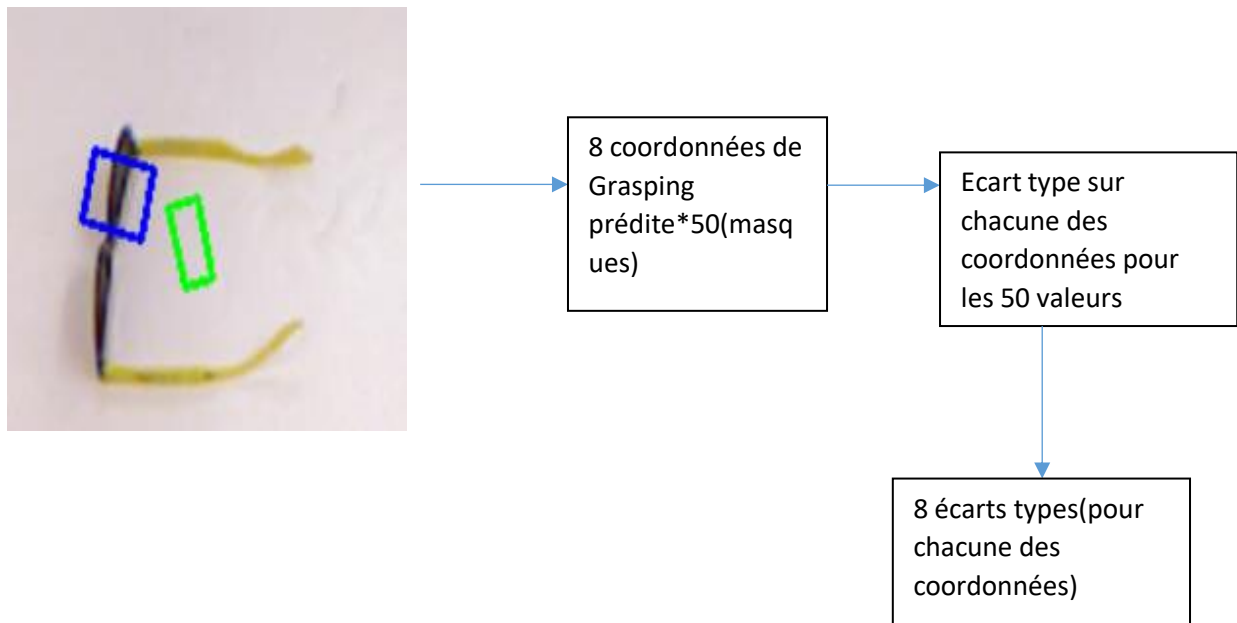
return(model)

mc_model = build_model(mc=True)
print(mc_model)

```

Si l’on choisit d’appliquer 50 masques de drop out ; on aura pour chaque entrée de validations 50 prédictions. Pour être plus précis, chaque image de validations(100 ici) aura 50 prédictions de rectangle de Grasping associé.

Notre démarche revient alors à faire l'écart-type de ces prédictions. Ainsi de faire un écart-type pour chacune des 8 coordonnées prédites. On résume le raisonnement sur le schéma suivant :



Nous avons donc 100×8 écarts-types pour nos 100 images de validation.

- **Mise en place d'un seuil sur l'écart-type :** Nous devons fixer un seuil à partir duquel nous considérons qu'une prédiction est certaine ou non. En faisant un print de la liste des écarts-types nous établissons alors un seuil sur l'écart-type.

```

50
[[2.86102295e-06 5.72204590e-06 7.62939453e-06 5.72204590e-06
 3.81469727e-06 0.00000000e+00 2.86102295e-06 5.72204590e-06]
 [4.76837158e-06 1.90734863e-06 9.53674316e-07 3.81469727e-06
 4.76837158e-06 5.72204590e-06 4.76837158e-06 1.33514404e-05]
 [0.00000000e+00 5.72204590e-06 6.67572021e-06 1.33514404e-05
 0.00000000e+00 1.33514404e-05 9.53674316e-07 7.62939453e-06]
 [5.72204590e-06 1.33514404e-05 8.58306885e-06 5.72204590e-06
 9.53674316e-06 0.00000000e+00 1.90734863e-06 1.14440918e-05]
 [1.90734863e-06 5.72204590e-06 5.72204590e-06 7.62939453e-06
 9.53674316e-06 9.53674316e-06 9.53674316e-07 7.62939453e-06]
 [9.53674316e-06 9.53674316e-06 1.90734863e-06 1.90734863e-06
 2.86102295e-06 0.00000000e+00 7.62939453e-06 1.14440918e-05]
 [3.81469727e-06 1.14440918e-05 1.90734863e-06 1.90734863e-06
 0.00000000e+00 9.53674316e-06 1.90734863e-06 3.81469727e-06]
 [5.72204590e-06 7.62939453e-06 7.62939453e-06 0.00000000e+00
 0.00000000e+00 9.53674316e-06 4.76837158e-06 7.62939453e-06]
 [1.90734863e-06 7.62939453e-06 1.90734863e-06 5.72204590e-06
 3.81469727e-06 1.90734863e-06 7.62939453e-06 9.53674316e-06]]
  
```

Nous fixons alors empiriquement un seuil à 10^{-6} .

VI) Prendre une décision

L'objectif de notre projet est donc de donner à partir d'un réseau de neurones, des coordonnées de préhension à un robot lorsqu'il doit saisir un objet : telephone, outil... Il faut être certain que la prédiction de grasping émise par notre réseau de neurones est bonne et robuste. Il faut donc

combiner le critère de Jacquard et le Monte Carlo drop out pour prendre une décision : le robot doit il agir à partir des coordonnées prédites par le réseau ou faire appel à un opérateur.

On définira alors quatre situation :

Prédiction juste et certaine : l'indicateur de Jacquard est différent de zero et les prédictions à l'issue des 500 masques ont un écart-type inférieur à 10^{-6} .

Prédiction juste et incertaine : l'indicateur de Jacquard est différent de zero et les prédictions à l'issue des 500 masques ont un écart-type supérieur à 10^{-6} .

Prédiction mauvaise et incertaine : l'indicateur de Jacquard est égal à zero et les prédictions à l'issue des 500 masques ont un écart-type supérieur à 10^{-6} .

Prédiction mauvaise et certaine : l'indicateur de Jacquard est égal à zero et les prédictions à l'issue des 500 masques ont un écart-type inférieur à 10^{-6} .

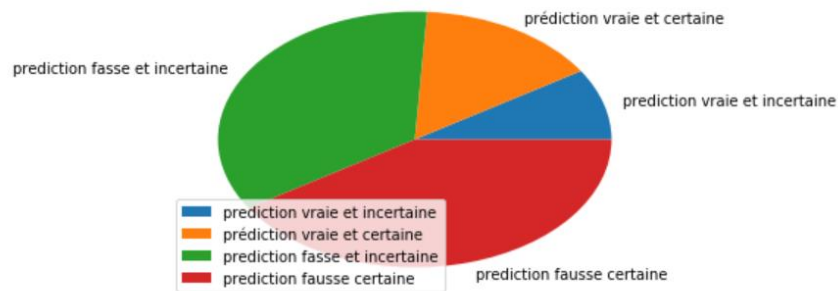
Cela se résume par le programme suivant :

```
compteur_prediction_vraie_certaine=0
compteur_prediction_fausse_certaine=0
compteur_prediction_vraie_incertaine=0
compteur_prediction_fausse_incertaine=0
#On impose un seuil sur l'écart type de  $10^{-6}$ 
for k in range(len(std_list)):
    if all(std_list[k]< $10^{-6}$  and L[k]==0:
        compteur_prediction_fausse_certaine+=1
    if all(std_list[k]> $10^{-6}$  and L[k]==0:
        compteur_prediction_fausse_incertaine+=1
    if all(std_list[k]< $10^{-6}$  and L[k]!=0:
        compteur_prediction_vraie_certaine+=1
    if all(std_list[k]> $10^{-6}$  and L[k]!=0:
        compteur_prediction_vraie_incertaine+=1

print('compteur_prediction_fausse_certaine',compteur_prediction_fausse_certaine)
print('compteur_prediction_fausse_incertaine',compteur_prediction_fausse_incertaine)
print('compteur_prediction_vraie_certaine',compteur_prediction_vraie_certaine)
print('compteur_prediction_vraie_incertaine',compteur_prediction_vraie_incertaine)
```

Nous souhaitons un maximum de prédiction juste et certaine. Néanmoins, le critère le plus important qui est tout simplement un critère de sécurité est de limiter les prédictions mauvaises et certaine. En effet, il est très problématique pour le robot, l'objet ou l'environnement environnement qu'une prédiction soit mauvaise et certaine. Si le robot passe à l'action alors que la prédiction est mauvaise, cela peut être dramatique.

Nous avons donc effectuer des tests pour 2000 epochs, 100 images de validations , 50 masques de drop out, un critère de Jacquard non nul et seuil sur l'écart-type de 10^{-6} . Nous avons la répartition suivante :



➤ Interprétation

VII) Conclusion générale

Dans la réalité nous n'aurons pas accès au critère de Jacquard. Il faudra donc se baser uniquement sur les résultats des masques de drop out. Notre conclusion est donc d'imposer un écart-type empiriquement le plus faible possible quitte à appeler plus souvent l'opérateur. On pourra réentraîner le réseau avec les cas de préhension les plus complexe.

Contact

—

Mail

jeffrey.verdiere@ensam.eu

Téléphone 0647633168

Adresse