
École nationale supérieure d'arts et métiers

UEE SMILE - Innovation Mécatronique & Robotique

# Grasping Regression

*Élèves:*
RICO GUARIN Juan Esteban

*Enseignants:*
O. GIBARU


December 15, 2023

# Grasping Regression

December 15, 2023

## Sommaire

# 1 Introduction

La manipulation robotique est une compétence fondamentale pour les robots opérant dans des environnements réels, leur permettant d'effectuer des tâches telles que la manipulation d'objets, l'assemblage et le tri automatisé. La maîtrise de cette compétence est cruciale pour que les robots puissent s'adapter à des objets divers et peu familiers dans un scénario de monde ouvert. L'apprentissage profond, avec ses remarquables capacités de généralisation, apparaît comme une voie prometteuse pour doter les robots de la capacité de saisir efficacement des objets divers.

Ce rapport examine un mini-projet qui se concentre sur la création d'un modèle permettant de déterminer les paramètres de préhension d'un objet. L'objectif est de développer un réseau neuronal capable de prédire les rectangles de préhension, en tirant parti de la richesse des informations fournies par la base de données de Cornell. Cette base de données se compose de 750 images, chacune annotée avec les paramètres de préhension associés, fournissant une base complète pour l'entraînement et l'évaluation.

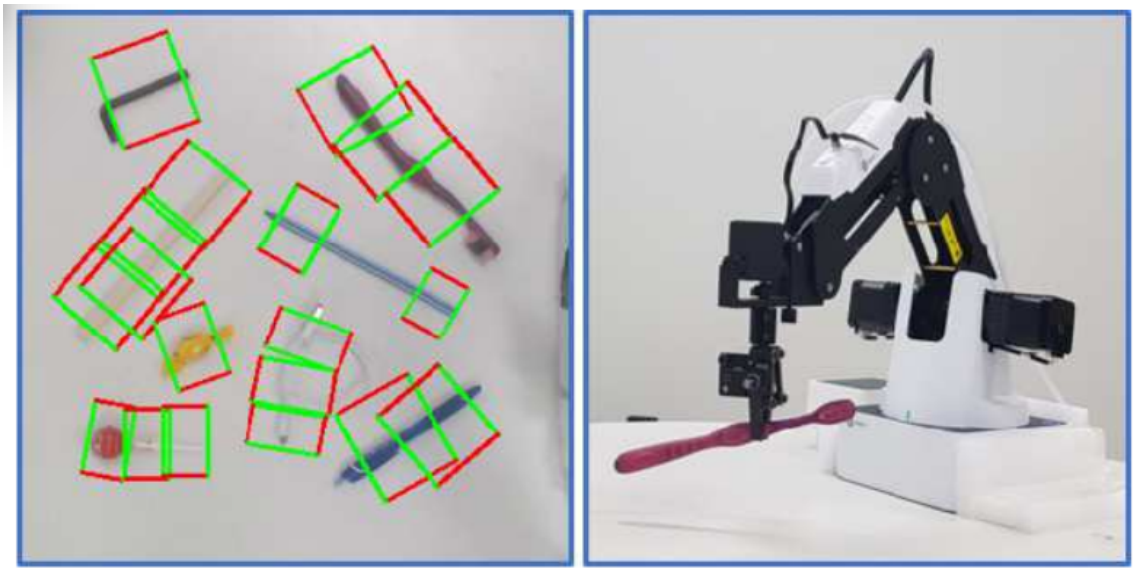

Figure 1: Préhension robotic

# 2 Description du problème : identification de la préhension

Le projet est développé dans l'environnement Colab qui fournit une plateforme interactive et polyvalente pour le développement basé sur Python. L'objectif de cette tâche est de créer un réseau neuronal pour déterminer les prises de vue possibles d'un objet à partir d'une photo d'un objet. Pour ce faire, nous utiliserons un échantillon de l'ensemble de données Cornell, qui se compose d'images montrant des objets sur un fond blanc. Tout au long du processus, nous discuterons :

- *Création d'un réseau neuronal pour la prédiction du rectangle de préhension* : il s'agit de concevoir une architecture de modèle capable de capturer efficacement les subtilités des divers objets de l'ensemble de données.

- *Évaluation des performances du réseau :* un aspect essentiel du projet consiste à évaluer les performances du réseau neuronal. Diverses mesures et techniques seront utilisées pour mesurer l'exactitude, la précision et l'exhaustivité du modèle dans la prédiction des paramètres de préhension.

- *Mise en œuvre d'une fonction de perte personnalisée* : le rapport décrit la création d'une fonction de perte personnalisée, un élément clé pour régler le réseau neuronal afin d'obtenir des performances optimales.

- *Augmentation des données pour améliorer les capacités de généralisation :* l'amélioration des capacités de généralisation du modèle par la mise en œuvre d'une classe d'augmentation des données est abordée. Cette section se concentre sur les stratégies visant à diversifier l'ensemble des données et à garantir la robustesse du modèle face aux différents scénarios du monde réel.

Dans ce travail, nous utiliserons la base de données Cornell, qui contient 750 images blanches. Chaque image est associée à une liste de préhensions possibles pour capturer l'objet. Ces solutions de préhension sont données comme suit :

$$[(x_1, y_1, \theta_1, \omega_1, h_1), (x_2, y_2, \theta_2, \omega_2, h_2), (x_2, y_2, \theta_2, \omega_2, h_2)] \tag{1}$$

Comme on peut le voir, nous utiliserons 5 paramètres de préhension, que l'on peut voir graphiquement dans la Figure 2.

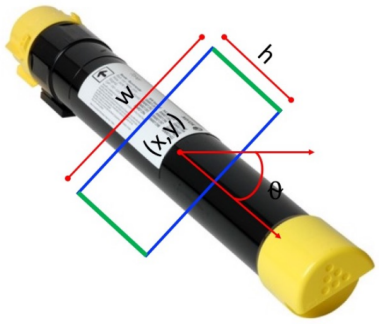


Figure 2: *Grasping* paramètres

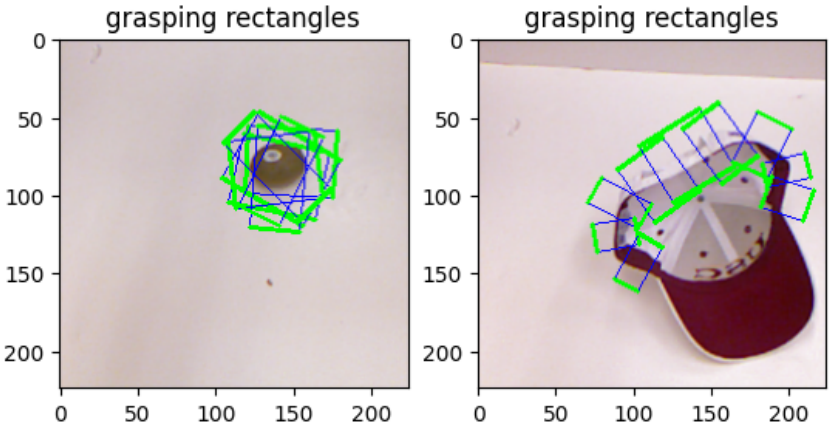


Figure 3: Exemple de données - Visualisation avec fonction *vizualise*

# 3 Preparation environnement

Ce bloc de code dans Google Colab installe des bibliothèques essentielles telles que TensorFlow, OpenCV et Matplotlib. Il importe ensuite les modules nécessaires à la manipulation des données et à la vision par ordinateur. Enfin, il monte Google Drive pour faciliter l'accès aux fichiers stockés dans le cloud pendant le développement des applications d'apprentissage profond.

Listing 1: Bibliothèques utilisées

```
!pip install tensorflow tensorflow-gpu opencv-python matplotlib
!pip list
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import os
import cv2
from google.colab import files, drive
drive.mount('/content/drive/')
```

Ces données sont ensuite stockées dans des tableaux, afin d'être traitées dans les fonctions précédentes. Il convient de noter la taille de nos données (entrée et sortie) :

```
Input dimensions:  (750, 224, 224, 3)
Output dimensions:  (750,)
```

Figure 4: Taille Input et Output

# 4 Modèle 1 - Première approche

Dans ce cas, un CNN sera utilisé pour la tâche en question. Les réseaux neuronaux convolutifs (CNN) sont utilisés dans les tâches de robotique de préhension en raison de leur capacité à apprendre des hiérarchies de caractéristiques visuelles, ce qui permet au réseau d'identifier des modèles clés dans les images sensorielles, tels que la position et la forme des objets. La structure des couches convolutives facilite l'extraction automatique des caractéristiques pertinentes pour la tâche de préhension, améliorant ainsi l'adaptabilité et l'efficacité du système dans des environnements dynamiques.

Pour les modèles, on a cherché l'état de l'art des différentes architectures et modèles qui sont couramment utilisés dans les problèmes de préhension robotique. Dans cette optique, le modèle présenté est dérivé de l'architecture proposée par Krizhevsky et al. pour les tâches de reconnaissance d'images. Cette architecture comporte 5 couches convolutives suivies de 3 couches denses entièrement connectées. Les couches convolutives sont suivies d'une couche de normalisation et de maxpooling.

Le modèle présenté dans cette première partie est un dérivé de ce modèle. Il comporte 4 couches de convolution, suivies d'une normalisation et d'un maxpooling. En outre, le premier noyau (5x5) permet une première exécution sur l'image avec suffisamment d'informations, puis les noyaux sont réglés sur 3, plus les strides, qui sont également réglés sur 2, ce qui rend le modèle suffisamment complexe pour la tâche à accomplir. Ensuite, les neurones denses contiennent un grand nombre de filtres pour capturer toutes les données importantes qui caractérisent nos images. Dans ces derniers neurones, les neurones Dropout sont positionnés, afin de minimiser l'overfitting dans notre formation, ainsi que pour aider un peu dans la simplification de notre modèle.

## 4.1 Description -Justification de couches

Dans cette partie, on donnera une brève explication des couches utilisées et de la raison pour laquelle on les utilise dans le modèle.

⇒ **Conv2D()**: La couche Conv2D (convolution bidimensionnelle) est utilisée dans un CNN en raison de sa capacité à extraire des images des caractéristiques spatiales pertinentes. La convolution permet de détecter des motifs locaux, tels que les bords et les textures, dans différentes régions de l'image.

⇒ **MaxPooling2D()**: MaxPooling2D réduit la dimensionnalité spatiale des caractéristiques extraites par les couches de convolution, en préservant les caractéristiques les plus importantes. Cela permet de diminuer le nombre de paramètres dans le réseau et de réduire le coût de calcul.

⇒ **Flatten()**: La couche Flatten() est utilisée dans un CNN pour transformer les activations bidimensionnelles obtenues à partir des couches de convolution et de regroupement en un vecteur unidimensionnel. Cette opération est essentielle pour relier les couches convolutives aux couches entièrement connectées, ce qui permet au réseau neuronal d'interpréter et de traiter les informations globales de l'image.

⇒ **Dense()**: Les couches denses sont utilisées pour effectuer des opérations de connexion complète entre les caractéristiques extraites par les couches convolutives et les classes de sortie. Ces couches sont essentielles pour l'apprentissage de représentations plus abstraites et plus complexes des caractéristiques des images.

⇒ **Dropout()**: La couche Dropout() permet d'atténuer l'adaptation excessive pendant l'apprentissage du modèle. L'ajustement excessif se produit lorsque le réseau apprend trop bien les détails spécifiques des données d'apprentissage et a du mal à se généraliser à de nouvelles données. (Overfitting)

## 4.2 Paramètres (Entre autres)

⇒ **Nadam**: L'optimiseur Nadam est utilisé dans ce modèle. Il combine les avantages de NAG et d'Adam pour offrir un équilibre entre une convergence rapide et la capacité de s'adapter à la variabilité des taux d'apprentissage.

⇒ **'accuracy'**: Dans un premier temps, nous considérerons la précision comme l'un des moyens d'évaluer la performance de notre modèle. En cours de route, nous verrons qu'elle n'est pas suffisante.

⇒ **Paramètres calculés**: Le nombre de paramètres calculés pour notre modèle est assez important, mais il est cohérent avec le type de tâche que nous avons. Ces paramètres sont visibles dans la Figure 5.

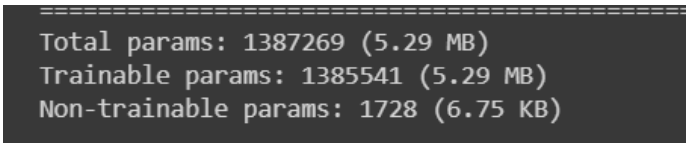


Figure 5: Nombre de paramètres entraîne

L'architecture est présentée à la Figure 6 et le code utilisé se trouve dans les annexes. Il constitue la base des modèles suivants.

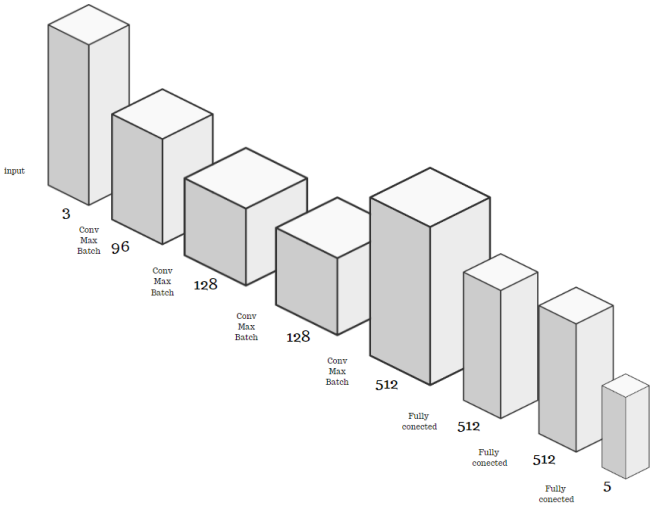


Figure 6: Représentation architecture

## 4.3 Training

Il faut maintenant passer à la partie formation de notre modèle. Pour ce faire, nous avons traité les listes $y_{train}$ et $y_{test}$ dans un format acceptable, puis nous avons lancé la formation. On a utilisé ici un *batch_size* : 50 et 120 *epochs*, ce qui permet un entraînement rapide et est suffisant pour un bon entraînement en termes de nombre d'exécutions par itération.

Listing 2: Training

```
## Training
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, ...
    Dropout
from keras import optimizers
import keras
print("Modelo: ", model0)
history = model0.fit(x_train,y_train, ...
    batch_size=50,epochs=120,verbose=1,validation_data=(x_test,y_test) ...
    )
```

## 4.4 Performance

Pour observer le fonctionnement du modèle, nous avons tracé un graphique de la "accurancy", une mesure qui nous permet de quantifier nos performances. Dans ce cas, un maximum de 82,5 % a été atteint. Cependant, pour notre tâche, il est nécessaire d'utiliser l'IOU (Intersection over Union) qui nous permet de bien comparer nos prédictions. Dans la figure, vous pouvez voir la performance en fonction de la "accurancy".

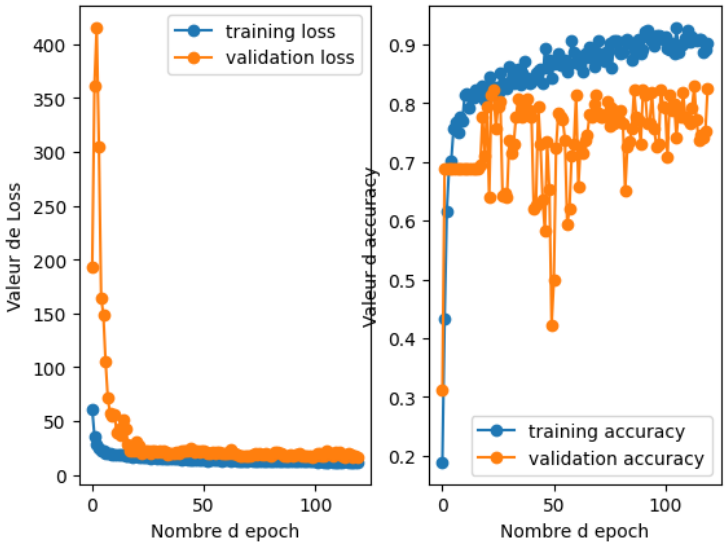


Figure 7: Performance: Accurancy

```
9/9 [==============================] - 0s 12ms/step
> 82.510
```

Figure 8: Performance: Accurancy - Pourcentage

La fonction de performance permet de calculer et de comparer les rectangles prédits. Elle est appelée à partir de la fonction suivante, qui permet de déterminer une bonne performance dans notre tâche de préhension.

Listing 3: Performance test

```
def test_performance(y_pred, y_test):
  pred=0
  for i in range(len(y_pred)):
    for j in range(len(y_test[i])):
      if performance(y_pred[i], y_test[i][j]):
        pred+=1
        break
result=pred/len(y_pred)
return(result)
```

Avec cette fonction, nous obtenons une performance de 63,1 %, ce qui est faible mais prévisible. On dispose de très peu de données et d'un modèle avec de nombreux paramètres que l'on peut qualifier de complexes. C'est pourquoi il faut chercher d'autres moyens d'améliorer les performances.

```
9/9 [==============================] - 0s 10ms/step
Performance of my trained model : 63.1%
```

Figure 9: Performance: Fonction propre - Pourcentage

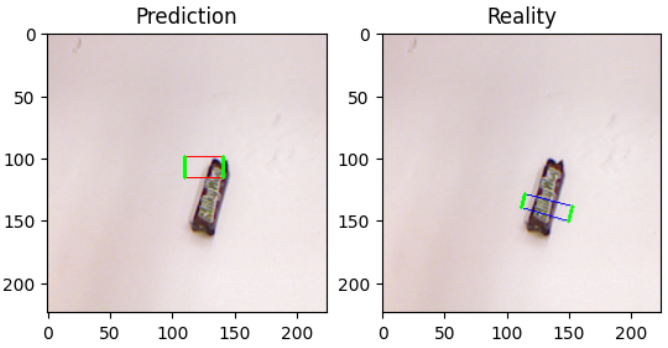


Figure 10: Prédiction avec réalité

# 5 Modèle 2 - Correction et "Normalisation" de Données

## 5.1 Nouveau format Y

Lors de la correction des données, Y est reformaté pour avoir une structure uniforme (nombre d'images, 40, 5), ce qui assure la cohérence et facilite l'apprentissage du modèle. Chaque image comporte désormais 40 rectangles de préhension uniformes représentés par cinq paramètres $(x, y, \theta, \omega, h)$. Pour assurer l'uniformité, des paramètres de préhension fictifs ([1000, 1000, 1000, 1000, 1000, 1000, 1000]) sont ajoutés à Y, agissant comme des marqueurs sans impact significatif sur la fonction de perte. Cette stratégie garantit que toutes les images ont exactement 40 rectangles de préhension possibles dans leur représentation, ce qui simplifie le traitement et l'optimisation du modèle pendant l'apprentissage.

Listing 4: Code *reshape* Y

```
def preprocess_data_min_mse(Y):
    num_grasps = 40
    default_grasp = [1000, 1000, 1000, 1000, 1000]
    # Fill with this array and have a "normalization of our data"
    filled_Y = []
    for image_grasps in Y:
        filled_Y.extend(image_grasps)
        num_missing_grasps = num_grasps - len(image_grasps)
        filled_Y.extend([default_grasp] * num_missing_grasps)
    reshaped_Y = np.array(filled_Y).reshape((len(Y), num_grasps, ...
        -1)) # Transform to numpy
    return reshaped_Y
```

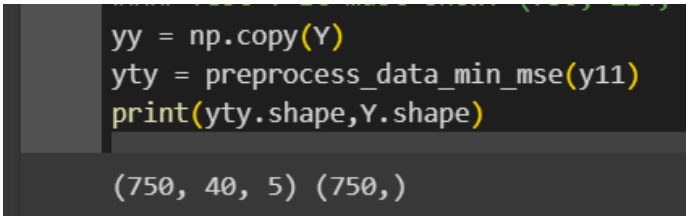


Figure 11: Nouveau taille de Y

## 5.2 *Loss Function*

La deuxième correction à apporter consiste à modifier la fonction de perte afin de prendre en compte toutes les solutions possibles. Dans cet ajustement, la fonction compare la prédiction à toutes les solutions connues et renvoie l'erreur relative à celle qui est la plus proche. En comparant la prédiction avec plusieurs solutions, la capacité du modèle à apprendre et à généraliser des modèles complexes liés à la tâche spécifique de préhension robotique est améliorée.

Listing 5: Function Loss

```
import tensorflow as tf
import tensorflow.keras.backend as K

def myFunction(y_true, y_pred):
    y_pred_temp = K.repeat(y_pred, K.shape(y_true)[1])
    res = K.mean(K.square(y_true - y_pred_temp), axis=[1,2])
    return res
```

## 5.3 Performance

Dans ce deuxième modèle, la même architecture que le modèle 1 a été réutilisée, seuls certains numéros de filtre des étapes de convolution ont été changés afin d'être frugal et de rechercher un modèle plus simple mais avec de grandes performances. Ce deuxième modèle permet d'obtenir une bonne réponse, avec une amélioration remarquable de 85,3 %.

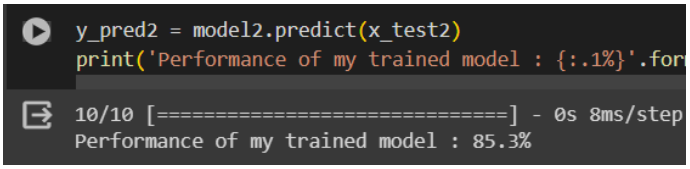


Figure 12: Performance Modèle 2

Ce modèle a une bonne réponse, mais nous pouvons aller plus loin. C'est pourquoi nous utiliserons une autre technique dans ce mini-projet pour améliorer encore les performances.

# 6 Modèle 3 - *Best* modèle

La technique d'augmentation des données sera mise en œuvre pour améliorer les performances du modèle, compte tenu de la quantité limitée de données (750 images). Un générateur de données (DataGenerator) sera créé pour effectuer des transformations géométriques aléatoires sur les images et leurs étiquettes pendant l'entraînement. En particulier, un changement de luminosité et un changement d'orientation seront appliqués pour simuler des conditions d'éclairage variables dans des environnements réels et différentes rotations de l'aéronef. La fonction de changement de luminosité utilisera OpenCV, convertissant les images RVB en HSV, modifiant le canal de luminosité et revenant à RVB. Le retournement se fera via np.fliplr ou np.flipud, une fonction créée pour cette action. Cette approche assure la variabilité sans affecter les coordonnées de la poignée, améliorant la robustesse du modèle à diverses conditions d'éclairage et d'orientation.

## 6.1 Modification de la luminosité

Cette fonction reçoit une image RVB en entrée. Elle convertit l'image au format HSV pour manipuler le canal de luminosité. Elle ajuste ensuite de manière aléatoire le facteur de luminosité, en évitant la saturation, et renvoie l'image avec la luminosité modifiée dans son format RVB d'origine.

Listing 6: Function Loss

```
import cv2
def change_luminosity(img):
    imhsv = cv2.cvtColor(img,cv2.COLOR_RGB2HSV)
    factor = np.random.uniform(0.1,1.9)
    mask = imhsv[:,:,2] * factor > 255
    v_channel = np.where(mask, 255, imhsv[:,:,2] * factor)
    imhsv[:,:,2] = v_channel
    return cv2.cvtColor(imhsv,cv2.COLOR_HSV2RGB)
```

## 6.2 *Flip*

Cette fonction prend en entrée une image et une liste de coordonnées d'adhérence. Elle choisit aléatoirement entre trois types de retournements (haut et bas, gauche et droite, ou pas de retournement). Ensuite, l'inversion correspondante est effectuée sur l'image et les coordonnées de préhension sont ajustées en conséquence, reflétant ainsi les modifications spatiales. La fonction renvoie l'image modifiée et les nouvelles coordonnées de préhension.

Listing 7: Function flip

```
def flip(image, y_list):
  img = np.copy(image)
  flipT= np.random.choice(3)

```

```
### For flip up and down

  if flipT==0:
    img = np.flipud(img)
    y_list_2=[]
    box_n=[]
    for box in y_list:
        for edges in grasp_to_bbox(box):
          box_n.append([edges[0], 224 - edges[1]])
    box_n = np.array(box_n).reshape(-1,8)
    [y_list_2.append(bboxes_to_grasps(flip_box)) for flip_box in ...
       box_n]
    y_list_2 = np.array(y_list_2,dtype=np.float64).reshape(-1,5)
### For flip right left
  elif flipT==1:
    img = np.fliplr(img)
    y_list_2=[]
    box_n=[]
    for box in y_list:
        for edges in grasp_to_bbox(box):
          box_n.append([224 - edges[0], edges[1]])
    box_n = np.array(box_n).reshape(-1,8)
    [y_list_2.append(bboxes_to_grasps(flip_box)) for flip_box in ...
       box_n]
    y_list_2 = np.array(y_list_2,dtype=np.float64).reshape(-1,5)
### Leave image like it comes to world :)
  else:
      y_list_2 = y_list
  return img, y_list_2
```

## 6.3 Performance

En réponse aux données limitées dont nous disposons, une fonction sera créée pour nous permettre d'avoir des données différentes. Ici, nous avons construit une classe DataGenerator qui tire parti des fonctions d'augmentation des données, telles que le flip et le changement de luminosité, pour générer des lots de données de manière dynamique pendant l'entraînement du modèle. Cette approche garantit que le modèle est entraîné avec une variété d'exemples transformés afin d'améliorer sa généralisation.

Nous utilisons cette dernière fonction avec la même architecture que celle utilisée dans le modèle 2, ce qui nous permet d'obtenir de bien meilleures performances que les 2 précédents. Dans ce travail, il s'agit du maximum atteint avec ces valeurs et peut être considéré comme une valeur exceptionnelle compte tenu de la complexité de la cible.

```
y_pred3 = model_3.predict(x_test3)
print(y_pred3.shape)
print('Performance of my trained model : {:.1%}'.format(test_performance(y_pred3, y_test3)))
```

```
5/5 [==============================] - 0s 9ms/step
(150, 5)
Performance of my trained model : 97.3%
```

Figure 13: Performance Modèle 3

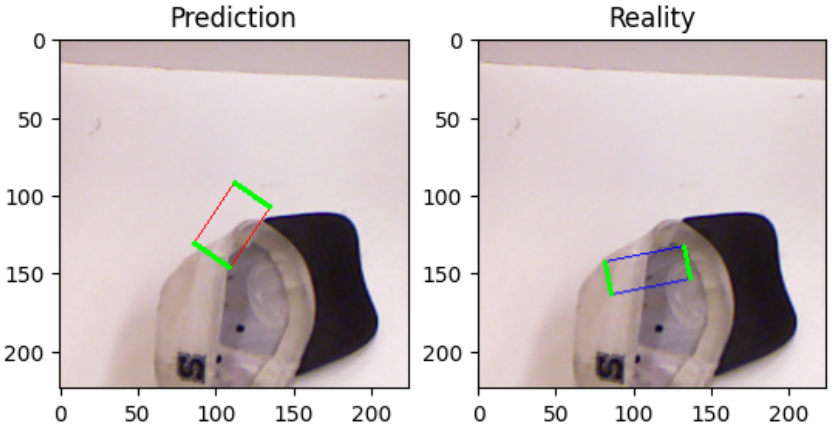


Figure 14: Prediction - exemple

# 7 Conclusions

En résumé, le rapport présente une approche robuste pour développer des modèles de prédiction de la préhension d'objets grâce à l'apprentissage profond, en mettant en évidence des stratégies efficaces d'augmentation des données, telles que le décalage et le retournement de la luminosité, qui améliorent de manière significative les performances du modèle. L'importance de la généralisation est soulignée, mettant en évidence l'adaptabilité du modèle à diverses conditions. Malgré des résultats remarquables, la nécessité d'explorer davantage la complexité du problème et d'envisager des approches supplémentaires pour améliorer encore les capacités du modèle est reconnue. Ce travail constitue une base solide pour les recherches futures dans le domaine de la manipulation robotique.

# 8 Reference


- Redmon, J., R., Angelova, A., A. (s. d.). Real-Time Grasp Detection Using Convolutional Neural Networks. https://arxiv.org/abs/1412.3128.4
- Caldera, S., C., Rassau, A., R., Chai, D., C. (2018, 3 septembre). Review of Deep Learning Methods in Robotic Grasp Detection. https://www.mdpi.com/2414-4088/2/3/57.

# 9 Annexes

Listing 8: Modèle 1

```
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from tensorflow.keras.layers import BatchNormalization

def model_1(input_shape):
  model = Sequential()
  model.add(Conv2D(96, (5,5), 2, activation='relu', ...
      input_shape=(224, 224, 3)))
  model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
  model.add(BatchNormalization())

  model.add(Conv2D(128, (3,3), 2, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
  model.add(BatchNormalization())

  model.add(Conv2D(128, (3,3), 1, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2,2), strides=2))
  model.add(BatchNormalization())

  model.add(Conv2D(512, (3,3), 2, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2,2), strides=2))
  model.add(BatchNormalization())

  model.add(Flatten())
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(5, activation='relu'))
  model.compile(loss='mean_absolute_error',
                optimizer='Nadam',
                metrics=['accuracy'])
  model.summary()
  return model

input_shape = (224, 224, 3)
model0 = model_1(input_shape)
```

Link: `https://drive.google.com/file/d/1lvkUlgy2BG6GcBGEPeFZiXPivu4e9AxL/view?usp=sharing`