




# Parcours Ingénieur Machine Learning Projet 08:

Participez à une compétition Kaggle !

E. Vezzoli

27/11/2019

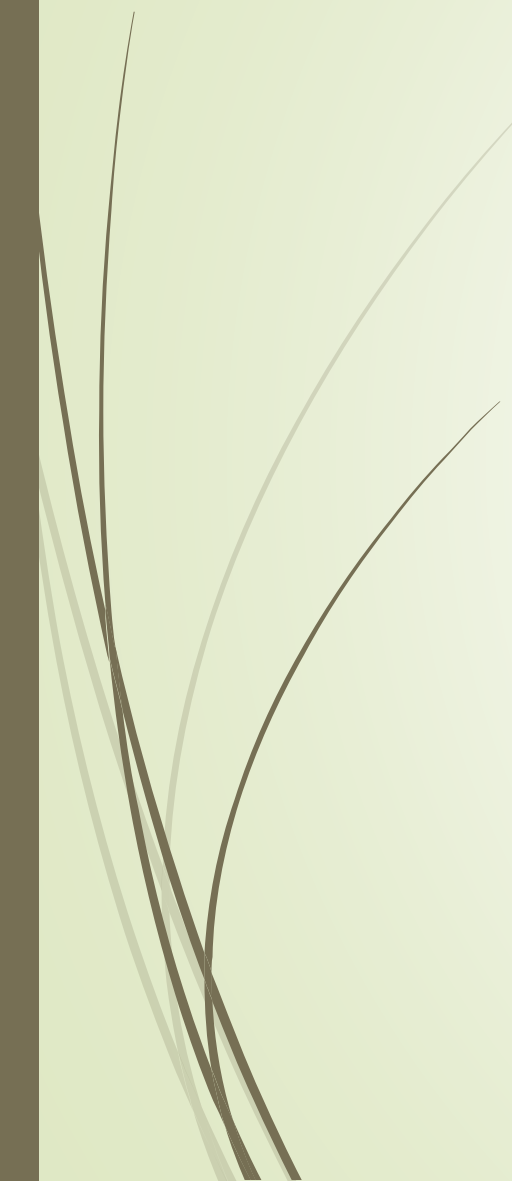
# Projet 08:



Peking University/Baidu -  
Autonomous Driving : Can you  
predict vehicle angle in different  
settings ?



# SOMMAIRE

- 
- 1) Problématique
  - 2) Données
  - 3) Analyse exploratoire
  - 4) Préparation des données
  - 5) Modélisation
  - 6) Evaluation
  - 7) Déploiement de la solution
  - 8) Conclusion



# Problématique

# Présentation de Kaggle

- Création : 2010
- Créateur : Anthony Goldbloom
- Principe :
  - Site de competition de data science
  - Site de cours en ligne
  - Forum de discussion theme data
  - Réseau social
- Autres :
  - Différents niveaux de novice à master



# Problématique

- Détecter les différentes positions des voitures:
  - Orientation / Distance
- Difficultés:
  - Image en entrée
  - Plusieurs voitures par images
  - 6 caractéristiques par voitures à prédire



# Environnement de travail

➤ Pour ce projet, il a fallu mettre en place un environnement de travail:

➤ Google collab:

- Gratuit
- CPU / GPU / TPU
- Connexion à google drive





Données

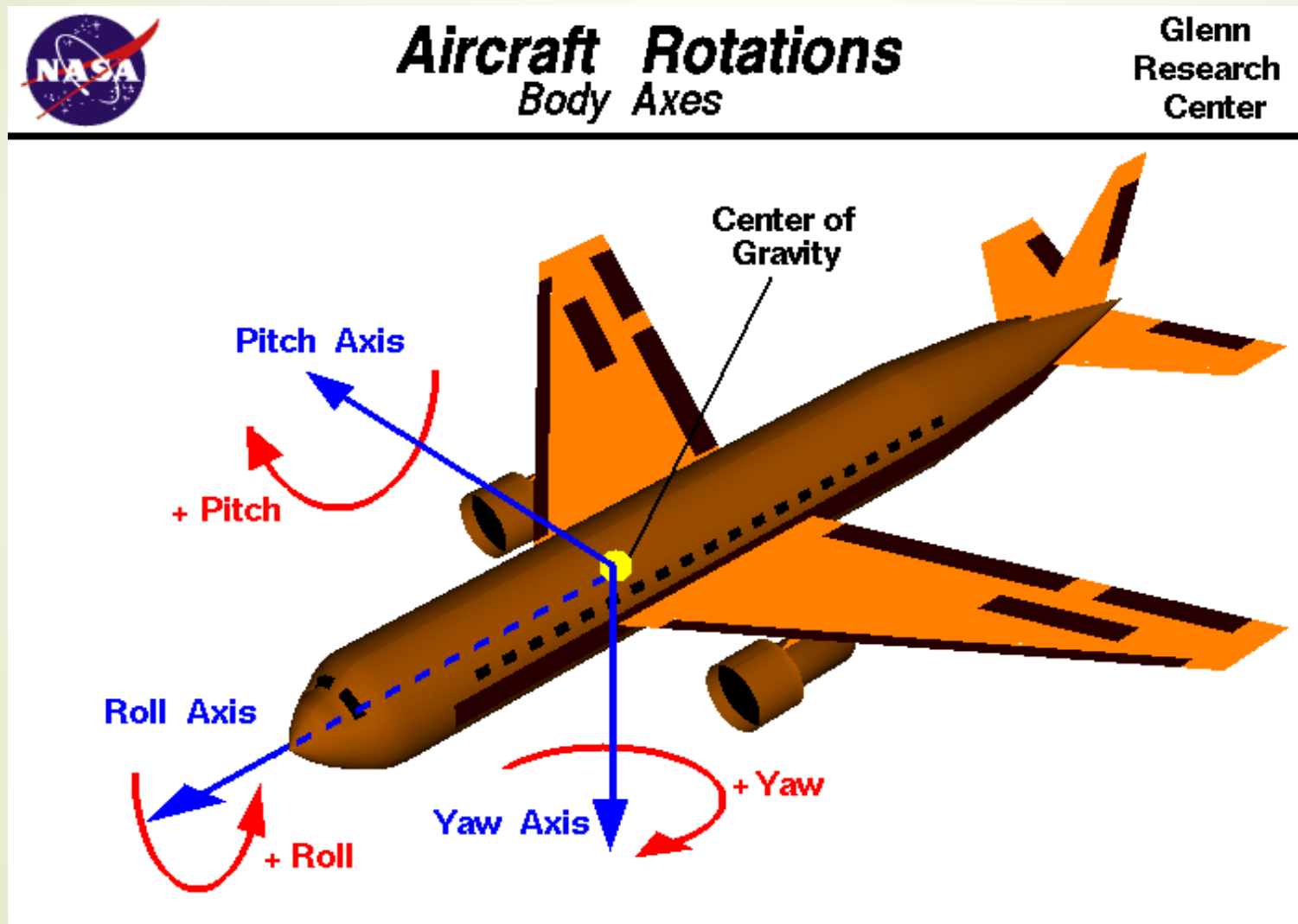


# Données d'entraînement

- Pour l'entraînement du modèle
  - Modèles véhicules .json
  - Données cibles pour l'entraînement train.csv
    - Car\_model
    - Yaw
    - Pitch
    - Roll
    - X
    - Y
    - Z
  - images d'entraînement
  - masks d'entraînement

train.csv (2.63 MB)		
	▲ ImageId ▼	▲ PredictionString ▼
	4262 unique values	4262 unique values
1	ID_8a6e65317	16 0.254839 -2.57534 -3.10256 7.96539 3.20066 11.0225 56 0.181647 -1.46947 -3.12159 9.60332 4.66632 19.339 70 0.163072 -1.56865 -3.11754 10.39 11.2219 59.7825 70 0.141942 -3.1395 3.11969 -9.59236 5.13...

# Yaw, Pitch, Roll



# Données d'évaluation

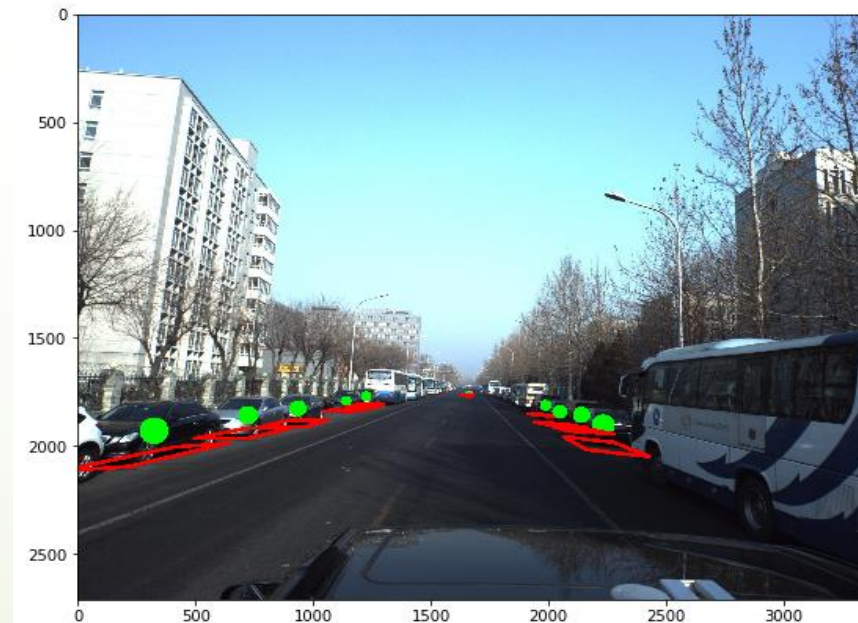
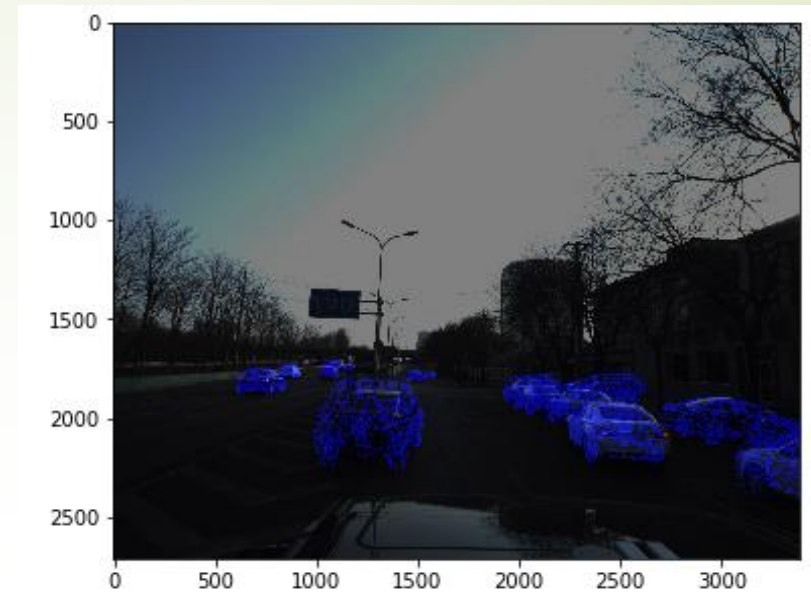
- Pour l'évaluation du modèle
  - Modèles véhicules .json
  - images d'évaluation
  - masks d'évaluation
  - Fichier exemple de soumission

train.csv (2.63 MB)		
	A ImageId	A PredictionString
	4262 unique values	4262 unique values
1	ID_8a6e65317	16 0.254839 -2.57534 -3.10256 7.96539 3.20066 11.0225 56 0.181647 -1.46947 -3.12159 9.60332 4.66632 19.339 70 0.163072 -1.56865 -3.11754 10.39 11.2219 59.7825 70 0.141942 -3.1395 3.11969 -9.59236 5.13...

# Resources interessantes

Pour la représentation de voiture en 3Dimension sur  
l'image : <https://www.kaggle.com/ebouteillon/augmented-reality>

Pour l'observation des centres des véhicules sur l'image :  
<https://www.kaggle.com/hocop1/centernet-baseline>

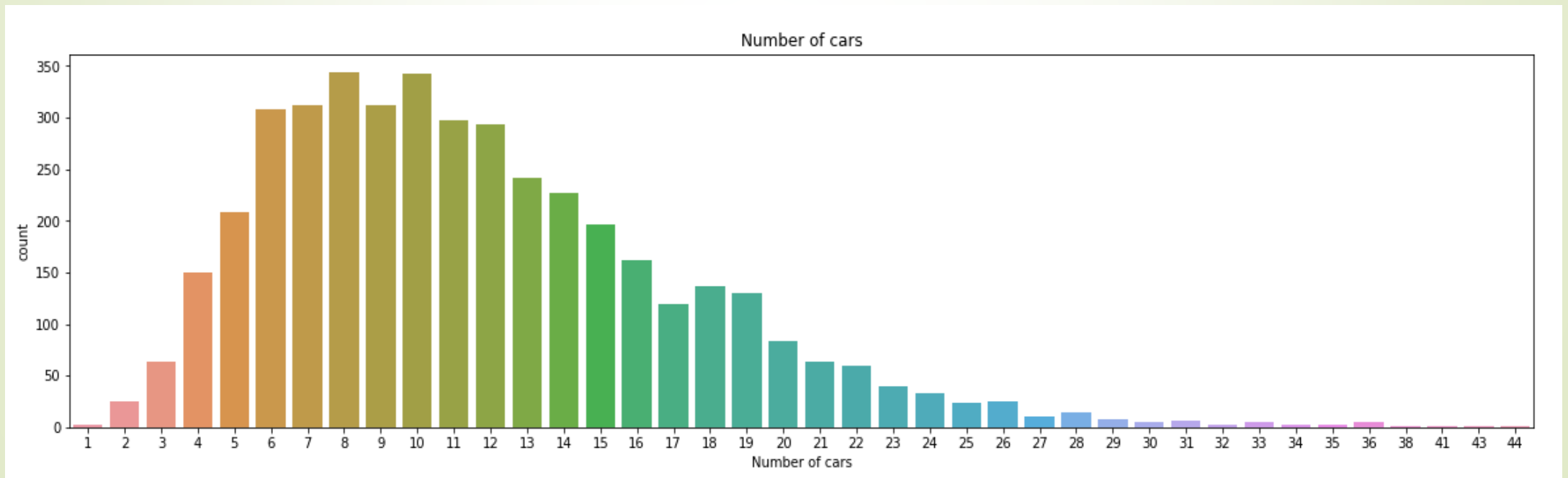




# Analyse exploratoire

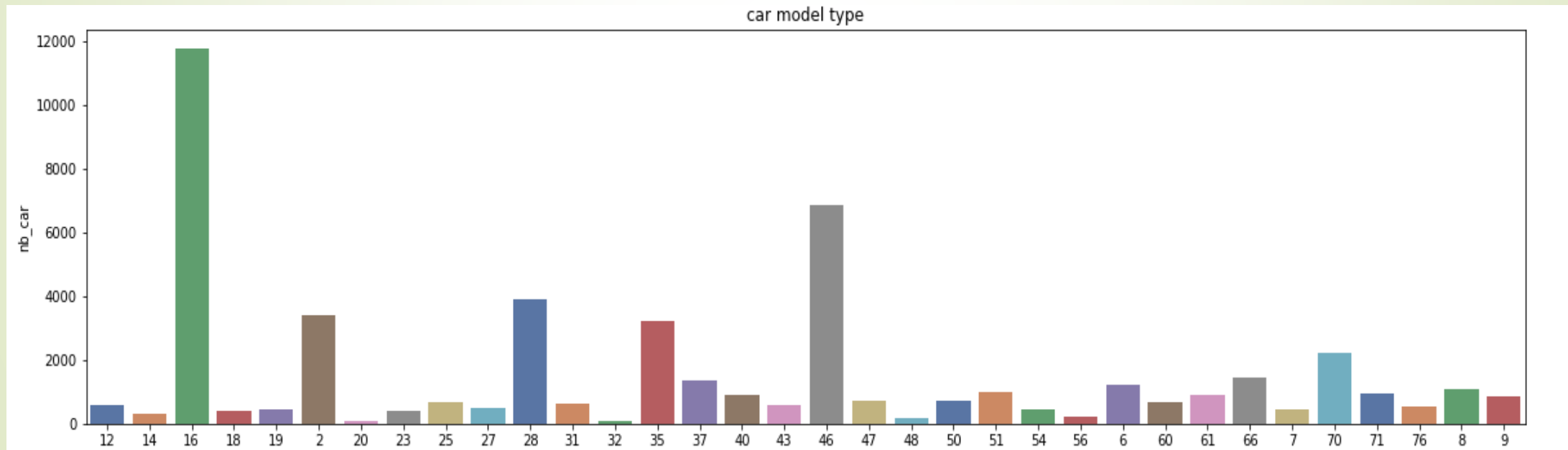
# Analyse exploratoire

- On observe la distribution du nombre de voitures



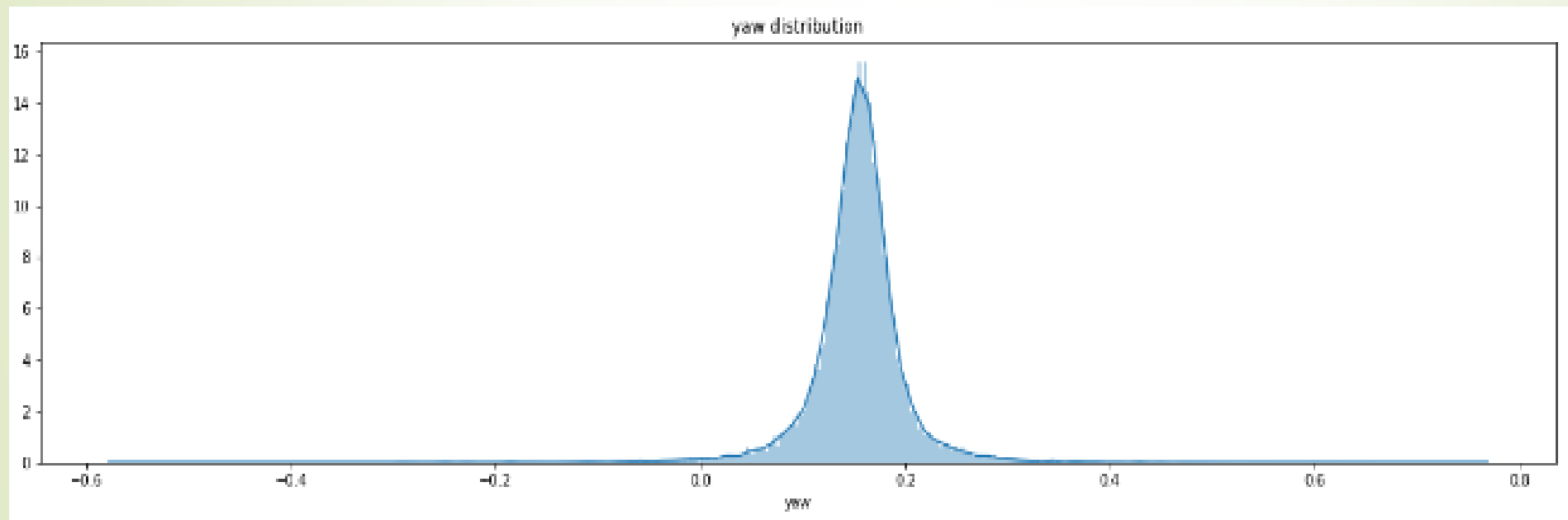
# Analyse exploratoire

- On observe la distribution des types de voitures



# Analyse exploratoire

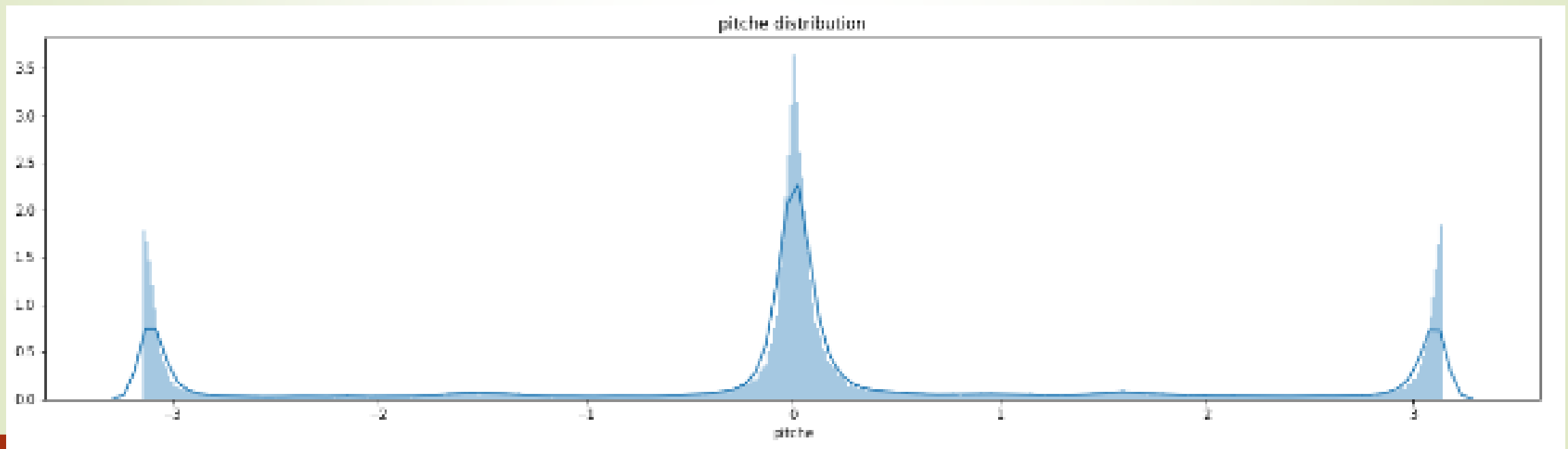
- ➡ On observe la distribution des yaw





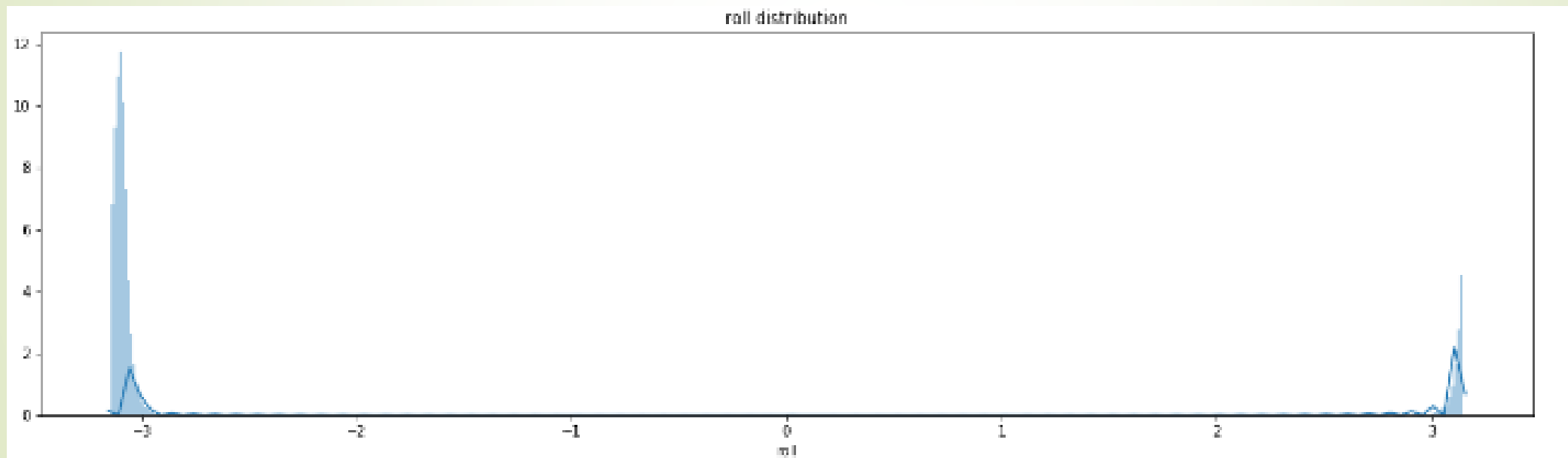
# Analyse exploratoire

- ➡ On observe la distribution des pitch



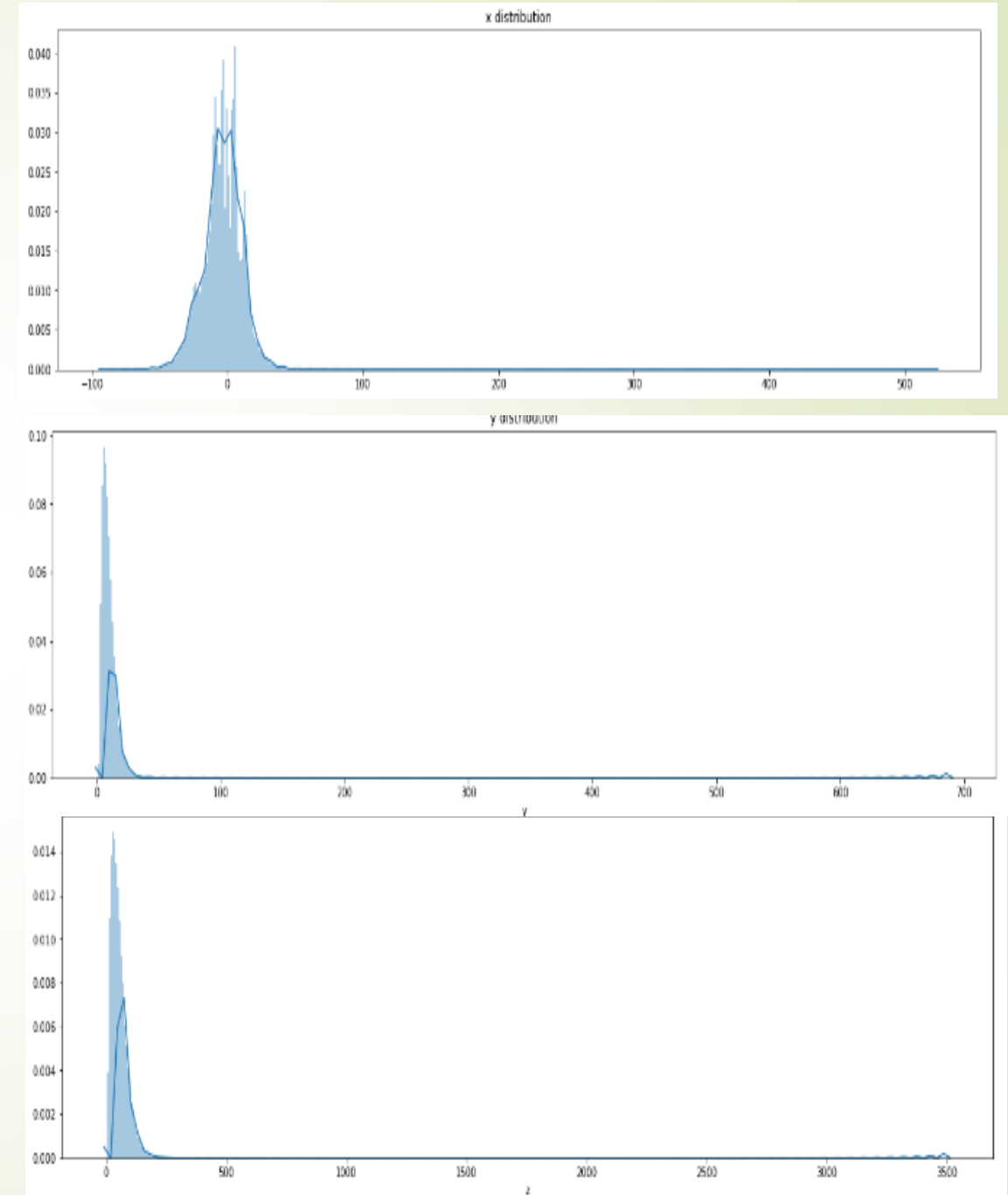
# Analyse exploratoire

- ➡ On observe la distribution des roll



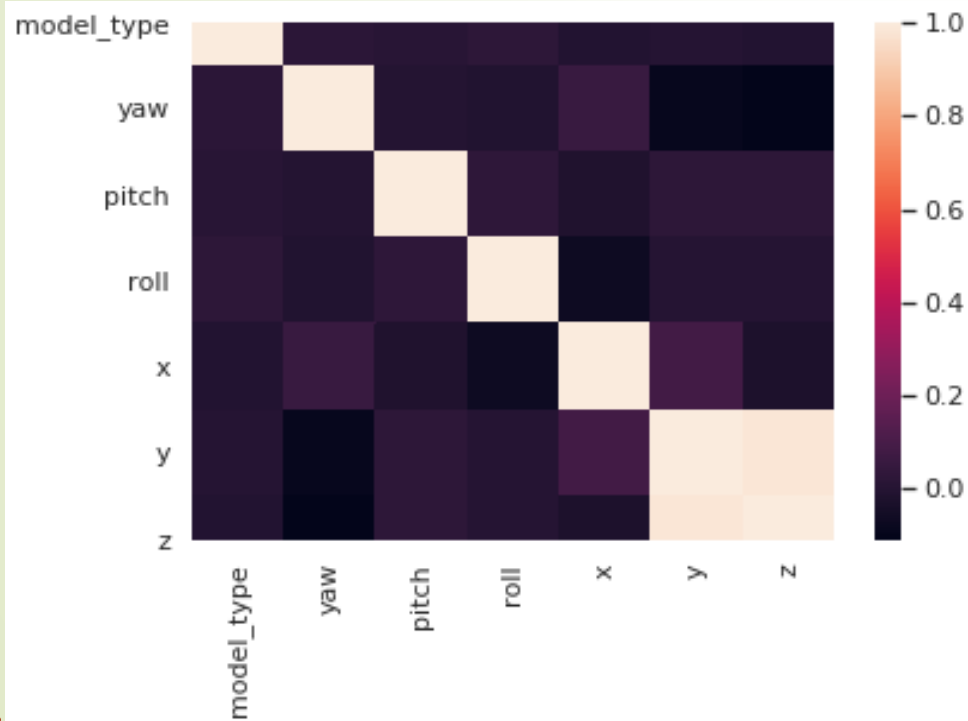
# Analyse exploratoire

- ➡ On observe la distribution des  $x$ ,  $y$  et  $z$

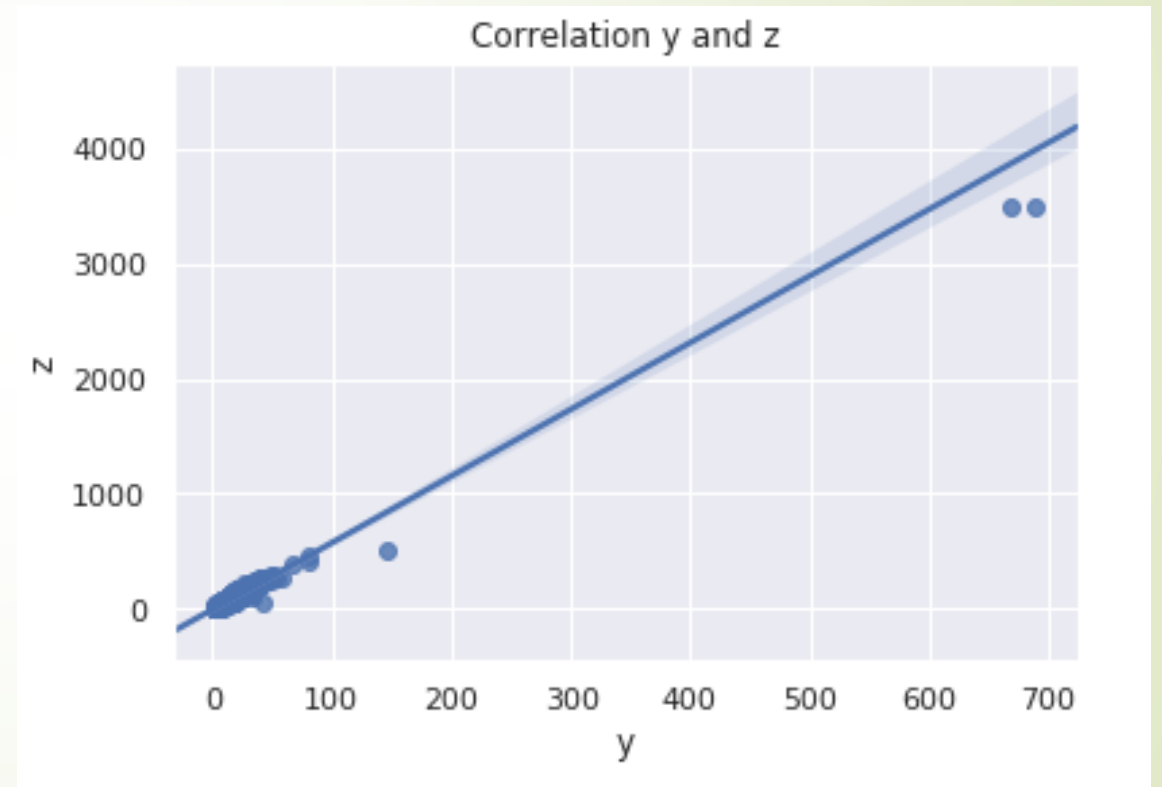


# Corrélation

- On étudie les corrélations



- Il y a une corrélation linéaire entre y et z
- Il n'y a peut être pas assez d'image en côte





# Préparation des données

# Redimensionnement des images

- Il est important de ne pas déformer les images
- Réduire la taille des images nous permettra d'entraîner le modèle plus rapidement

```
1 def image_format(i, border, size):  
2     image1 = Image.open(train['train_img'][i])  
3     image1 = ImageOps.expand(image1, border=border, fill=0)  
4     x = np.array(image1).shape[0]  
5     y = np.array(image1).shape[1]  
6     image1 = ImageOps.fit(image1, (max(x, y), max(x, y)), 2, 0.0, (0.5, 0.5))  
7     image1 = image1.resize((size, size), resample=0)  
8     return image1  
9  
10
```

# Séparation train test val

```
liste_index_train, liste_index_test, liste_index_val = prepare_data(alpha=0.2, nb_pics=0.2, size=512, border=0)
```

- On sépare le dataset en trois
  - Train (pour l'entraînement)
  - Test (pour valider l'entraînement)
  - Val (pour valider les performances du modèle)
- Méthode:
  - Création d'une fonction qui déplacent les données aléatoirement dans les dossiers



# Enregistrement des jeux dans des dataframes

- On crée trois dataframes contenant:
  - Les chemins d'accès aux fichiers train test et val
  - Les valeurs cibles standardisé entre -1 et 1 (par souci d'efficacité pour les modèles)

```
1 x_train = train['train_img'][liste_index_train]
2 x_test = train['train_img'][liste_index_test]
3 x_val = train['train_img'][liste_index_val]
4 y_train = liste['nb_car'][liste_index_train]/44
5 y_test = liste['nb_car'][liste_index_test]/44
6 y_val = liste['nb_car'][liste_index_val]/44
7
8 # Pour l'entrainement
9 new_train = pd.DataFrame({"x":x_train})
10 new_train = new_train.join(y_train)
11
12 new_test = pd.DataFrame({"x":x_test})
13 new_test = new_test.join(y_test)
14
15 # Pour la validation
16 new_val = pd.DataFrame({"x":x_val})
17 new_val = new_val.join(y_val)
18 # On corrige les chemins d'accès
19 new_train['x'] = new_train['x'].str.replace('train_images', 'Train')
20 new_test['x'] = new_test['x'].str.replace('train_images', 'Test')
21 new_val['x'] = new_val['x'].str.replace('train_images', 'Val')
```



# Création des fonctions d'évaluations

- On crée une fonction de perte se basant sur la somme des erreurs absolues
  - Permet d'entraîner le modèle efficacement

```
1 def sum_absolute_error(y_true, y_pred):  
2     if not K.is_tensor(y_pred):  
3         y_pred = K.constant(y_pred)  
4     y_true = K.cast(y_true, y_pred.dtype)  
5     return K.sum(K.abs(y_pred - y_true), axis=-1)
```



# Modélisation

# Hypothèse de modèle

- CNN + Régression
- Capsules networks (Problème de ressources (RAM))
- Center\_Resnet (Pas assez de ressources (RAM))

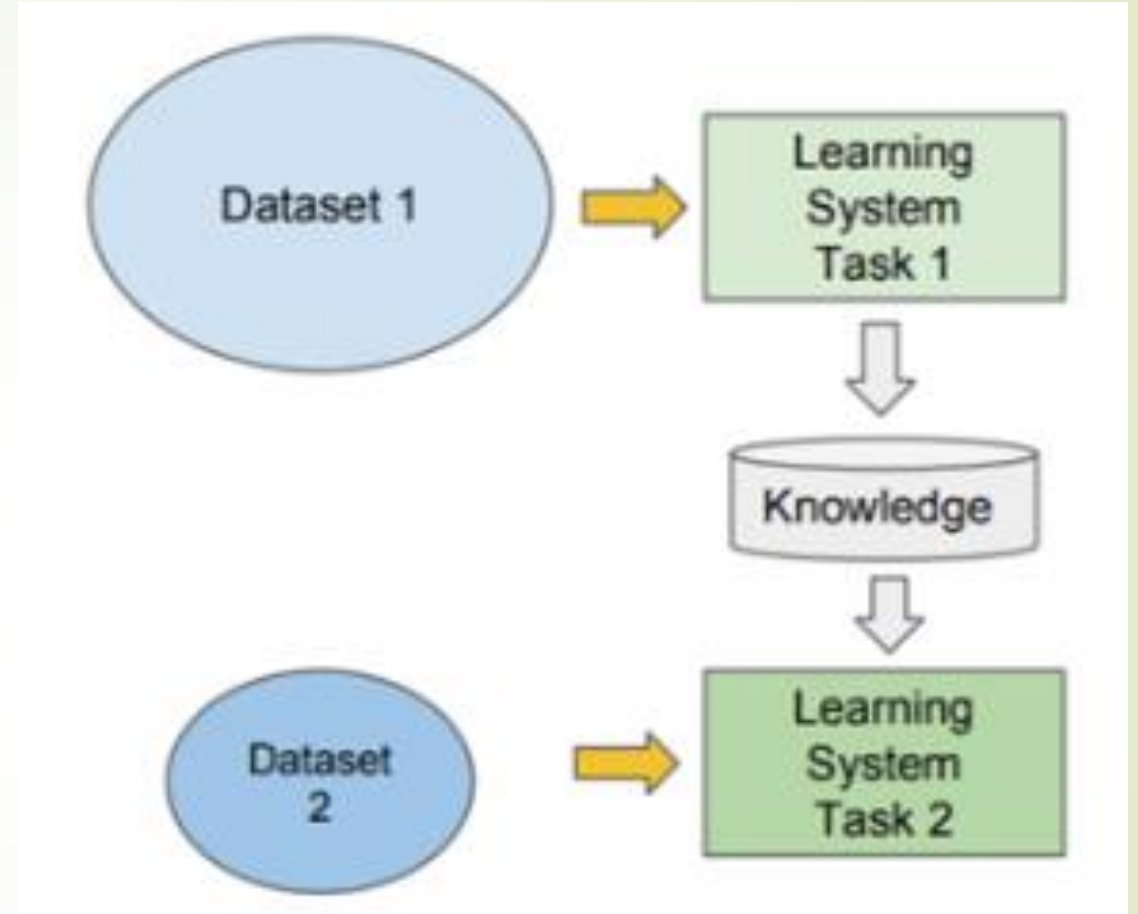
# CNN + Régression

- On va créer plusieurs modèles 1 pour chaque variable cible
- On utilisera un reseau de neurones à convolution pour détecter les features
- On utilisera un reseau de neurons entièrement connecté pour effectuer une regression
- On comparera les performances des modèles sur une simple variable (par souci de temps)



# Présentation de inceptionresnetv2 (TRANSFER)

- L'approche par transfer learning permet d'utiliser une base déjà entraîné sur une grande quantité de données.
- Il ne reste qu'à l'adapter à la nouvelle base de données
- On entraîne dans ce cas les dernières couches



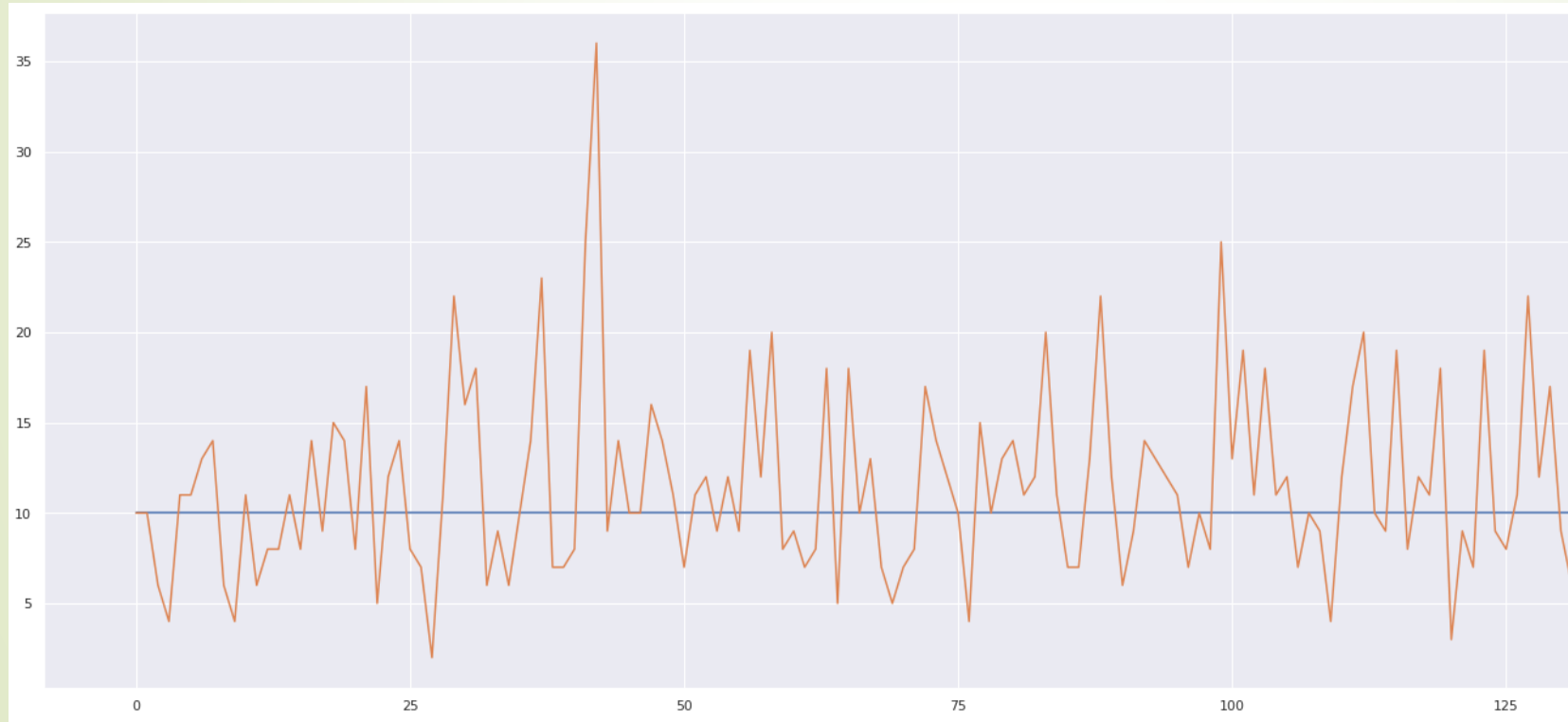
Source: [towardsdatascience.com](https://towardsdatascience.com)

# Premier modèle : inception\_resnet\_v2

Layer (type)	Output Shape	Param #
=====	=====	=====
inception_resnet_v2 (Model)	(None, 14, 14, 1536)	54336736
flatten_2 (Flatten)	(None, 301056)	0
dense_3 (Dense)	(None, 1024)	308282368
dense_4 (Dense)	(None, 1)	1025
=====	=====	=====
Total params: 362,620,129		
Trainable params: 308,283,393		
Non-trainable params: 54,336,736		

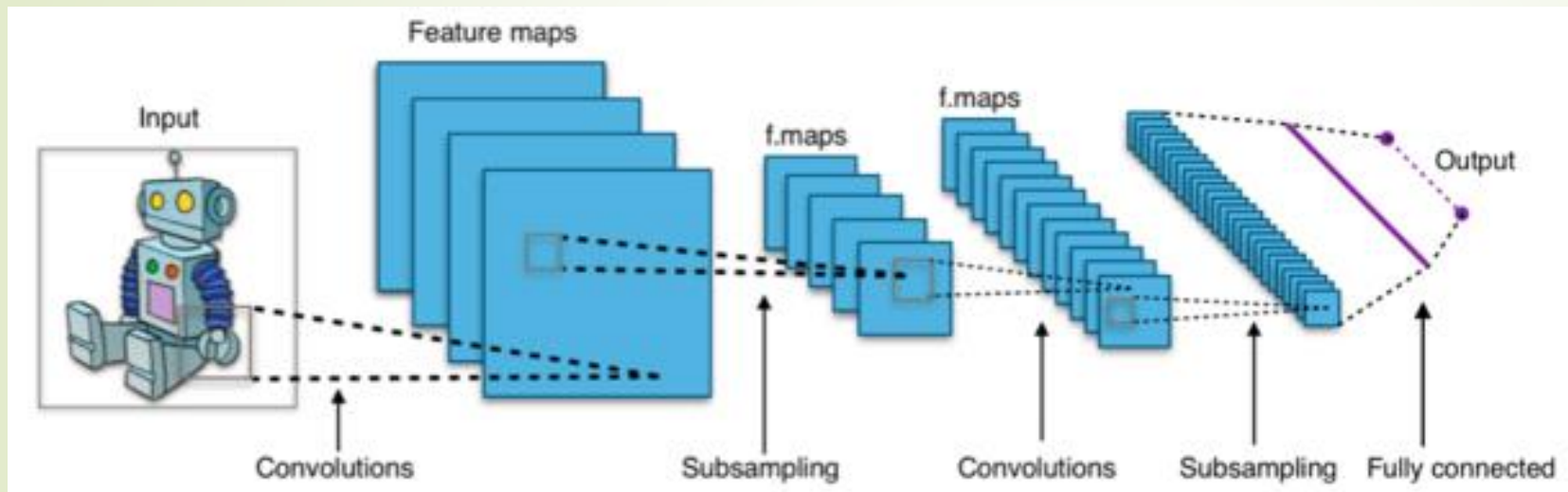
# Premier modèle : inception\_resnet\_v2

➡ On observe les valeurs prédites: résultat faible et très lourd



# Présentation CNN

- Rappel : fonctionnement d'un CNN





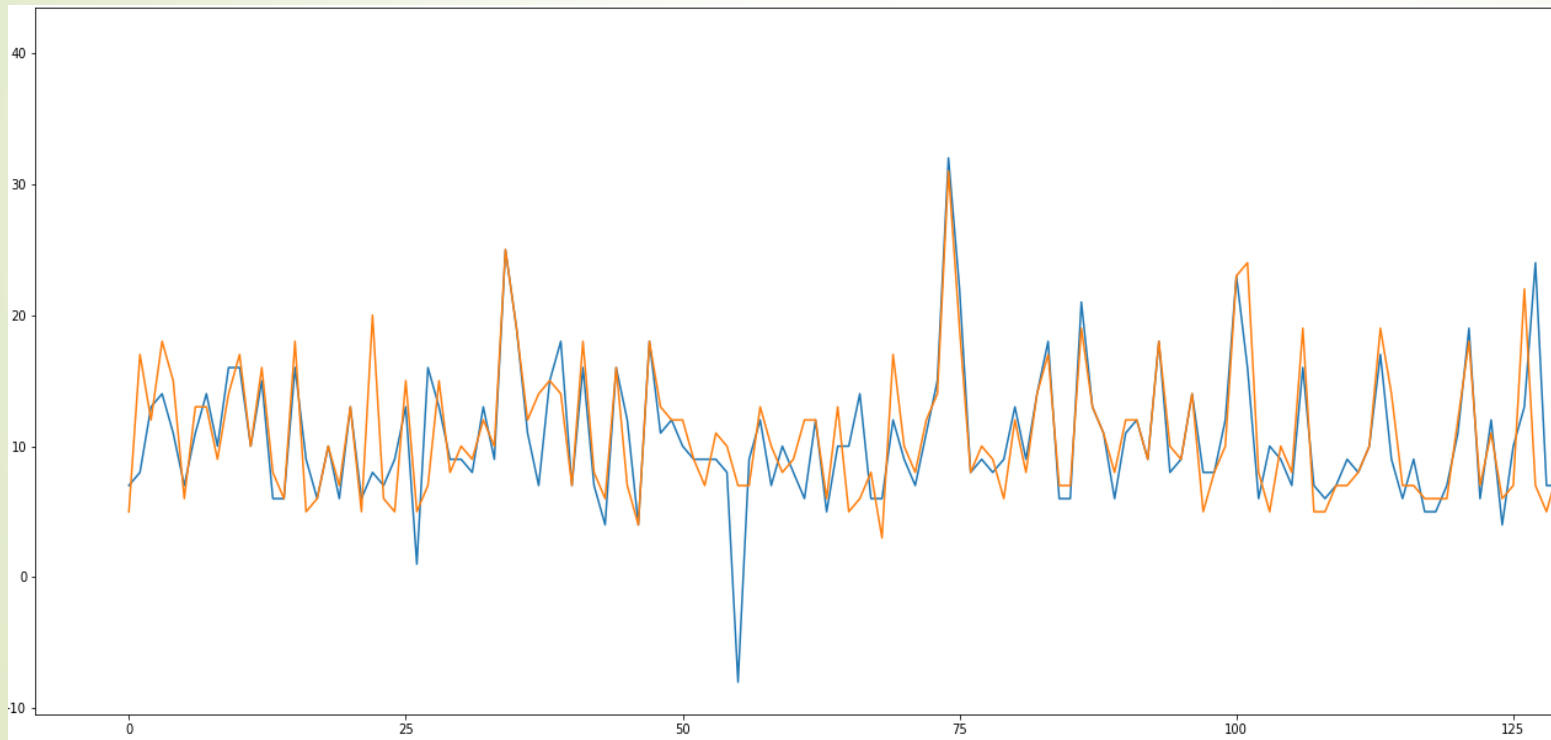
# Deuxième modèle : CNN + Régression

- 6 couches de convolution
  - Conv2d
  - Batch-normalization
  - Max\_pooling2d
- 1 flatten
- 1 dense (activation: relu)
- 1 dropout
- 1 Dense (activation: linear)

conv2d_604 (Conv2D)	(None, 254, 254, 4)	76
batch_normalization_604 (Bat	(None, 254, 254, 4)	16
max_pooling2d_604 (MaxPoolin	(None, 127, 127, 4)	0
flatten_87 (Flatten)	(None, 64)	0
dense_173 (Dense)	(None, 1024)	66560
dropout_87 (Dropout)	(None, 1024)	0
dense_174 (Dense)	(None, 1)	1025
=====		
Total params: 72,417		
Trainable params: 72,301		
Non-trainable params: 116		

# Deuxième modèle : CNN + Régression

- On observe les valeurs prédites: les predictions sur la validation suivent les valeurs cibles



# Deuxième modèle : CNN + Régression

- On cherche à optimiser les performances

Paramètres	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
CNN filter	2	4	4	1	4	2
Kernel size	(3, 3)	(3, 3)	(3, 3)	(1, 1)	(1, 1)	(1, 1)
Unit	1024	1024	1025	1024	1024	1024
Nb dense	2	1	1	1	1	1
Dropout	0.4	0.2	0.5	0.5	0.5	0.5
Mean absolute error	3.0106	3.08	2.74705	2.7	2.08	2.07

# Comparaison des performances

- Comparons les performances des modèles

	CNN+Regression	Inception_resnetV2
Time/step	300ms/step	863ms/step
Nb paramètre	72417	308283393
Taille des poids	348ko	3.65 Go
Mean absolute error	2.07	3.82

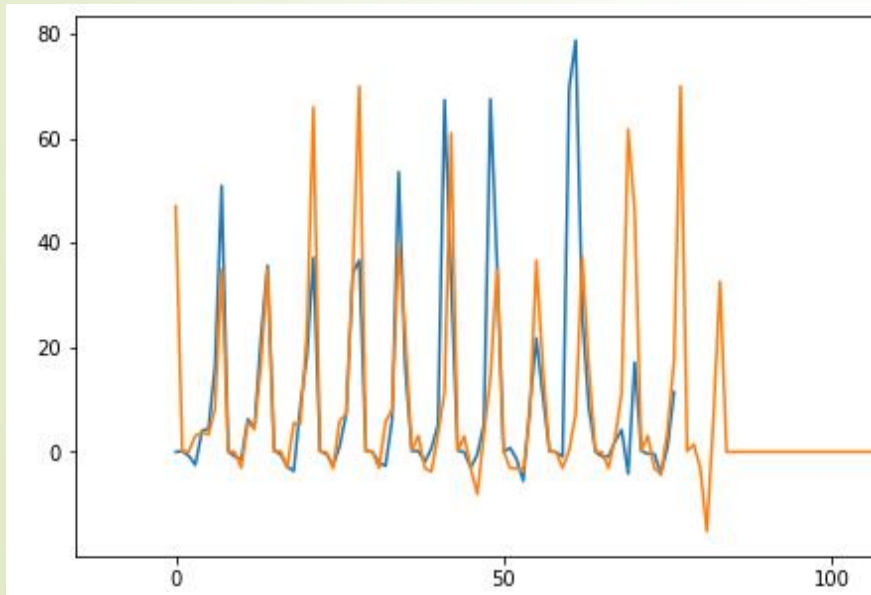
- Le meilleur modèle est le CNN+ Regression
- On l'applique sur toutes les variables cibles (toujours par souci de temps)



# Evaluation

# Evaluation complémentaire

- On charge l'ensemble des modèles sur les images de validation
- On stocke ces valeurs dans un dataframe
- On corrige les données impossibles
- On observe les predictions en bleu et les données réelles en orange



```
1 model_name = 'nb_car.h5'
2 first_model.load_weights('/content/' + model_name)
3 # import val
4 y_hat_total = []
5 for i in list(range(len(val_set.filepaths))):
6
7     test_image = Image.open(val_set.filepaths[i])
8     test_image_full = np.array(test_image)
9     test_image = test_image.resize((512, 512), resample=0)
10    test_image = np.array(test_image)/255
11    test_image = np.expand_dims(test_image, axis=0)
12    result = int(first_model.predict(test_image)*44)
13    y_hat = []
14    for i in list(range(1,141)):
15        if i <= (7*result-1):
16            model_name = 'model' + str(i) + '.h5'
17            train_model.load_weights('/content/' + model_name)
18            result2 = train_model.predict(test_image)
19            y_hat.append(result2[0][0])
20        else:
21            y_hat.append(None)
22    y_hat_total.append(y_hat)
```

```
result = pd.DataFrame(y_hat_total, index=list(new_val.index), columns=list(range(1,141)))
for i in list(range(141,308)):
    result[i] = None
result[0] = 0
result = result[list(range(0,308))].astype('float32')
for i in list(range(0,308)):
    for j in list(result.index):
        if result.loc[j, i] is not None:
            if result.loc[j, i] > 1:
                result.loc[j, i] = 1
            elif result.loc[j, i] < -1:
                result.loc[j, i] = -1
result = result*normalize_max
```



# Evaluation visuelle

- On observe que quelques voitures sont correctement placées





Déploiement de la solution



# Déploiement

- On applique les algorithmes sur le jeu de test du dataset
- On Transforme les résultats pour obtenir le format demandé
- On exporte le fichier en csv de soumission et on le poste sur Kaggle





# Conclusion



# Conclusions

- Le modèle trouvé est très faible
- Solution : chercher un autre modèle plus performant
- Augmenter les ressources disponibles en ligne (Passer sur une solution AWS/ IBM Google payante)

<https://www.kaggle.com/evezoli/simple-eda-nb-car-find?scriptVersionId=24158145>



Merci de votre attention