# Participez à une compétition Kaggle !

## Projet n°8 du Parcours Ingénieur Machine learning

E. Vezzoli

11/2019

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

## Table des matières

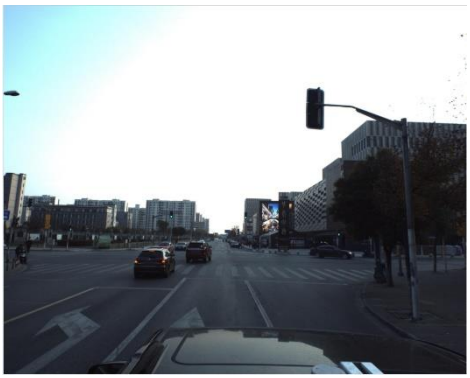
# Présentation

## Présentation de Kaggle

Kaggle est une plateforme web créé en avril 2010 par Anthony Goldbloom, organisant des compétitions en science des données. Sur ce site, des entreprises proposent des problèmes de data science et offrent des récompenses aux participants obtenant les meilleures performances. Kaggle fourni un outil de travail permettant l'utilisation de CPU et GPU pour concevoir du code et un notebook. Il est orienté pour une utilisation en communauté, avec la possibilité de participer aux discutions avec de nombreux Data Scientist et Data Analyst, de suivre des formations, de fournir des Datasets et d'évoluer graduellement dans chaque catégorie.

## Présentation de la compétition

Dans cette compétition, nous devons, à partir d'une image, détecter les voitures, leurs positions et leurs orientations dans l'espace.



Les données représentent environ 5 GB et contiennent pour la partie entrainement, différentes images, leur masks et un fichier csv recueillant l'ensembles des valeurs à prédire. Nous avons également un fichier .json pour chaque modèle de véhicule permettant de représenter la voiture sur trois dimensions et un fichier représentant les caractéristiques de la caméra.

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... |

Pour la partie test, nous n'avons que les images et leurs masks qui permet de cacher les voitures non prisent en compte dans le relevé des annotations. Un fichier sample_submission.csv est fourni comme exemple de soumission.

## Présentations des sources utiles

De nombreuses sources ont permis d'orienter mon travail vers des étapes de modélisations.

- 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

## Présentation de l'environnement de travail

Kaggle fourni un accès à un notebook utilisant de la GPU limité à 30h. Pour palier à ce problème, nous allons utiliser google colab. Malheureusement, les ressources disponibles ne permettent pas d'implémenter rapidement n'importe quel modèle, mais nous pouvons utiliser la GPU avec une limite de 12h/jours ce qui permet de faire tourner plus de modèles.

### Téléchargement des données

Après avoir charger les différentes librairies utiles pour ce projet, nous devons récupérer les données sur le site Kaggle. Pour ce faire, nous utiliserons l'API.

```
os.environ['KAGGLE_USERNAME'] = Kaggle_username
os.environ['KAGGLE_KEY'] = Kaggle_key
!kaggle competitions download -c pku-autonomous-driving # api copied from kaggle
```

Une fois télécharger, nous pouvons utiliser la librairie ZipFile pour décompresser les dossiers.

Nous créons ensuite les différentes fonctions utiles au projet.

```
# Création d'un dossier
def create_repertory(rep):
    """
    fonction de création de dossier
    """
    try:
        os.mkdir(rep)
    except:
        print('Le dossier est existant')
```

Nous pouvons ensuite, lister les fichiers des différents dossiers.

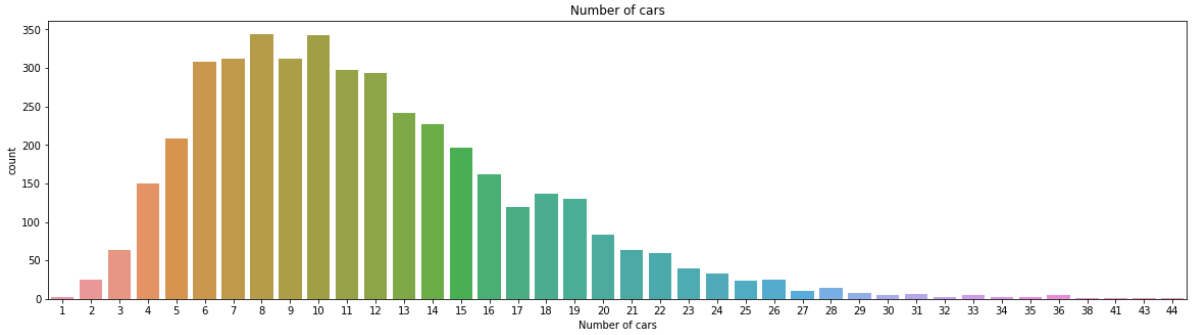
```
# On liste les json
files = glob.glob('Data/car_models_json' + "/*.json")
for file in files:
    car_models_json.append(file)
```

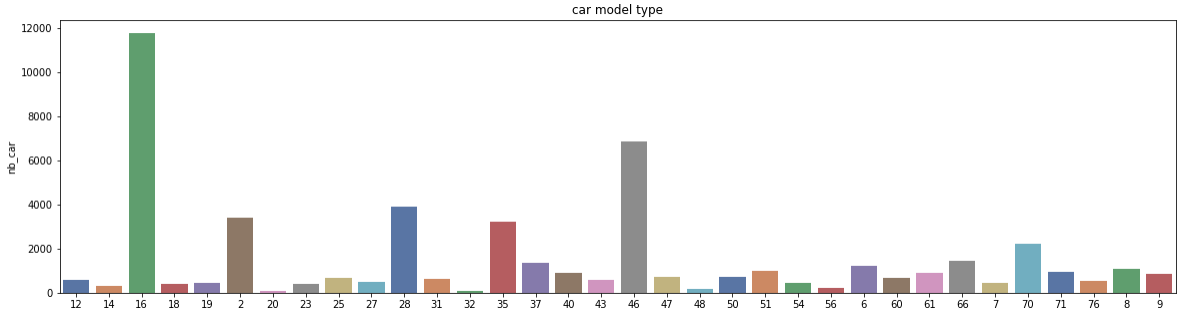
On crée le dataframe d'entrainement permettant d'obtenir pour chaque image, les informations sur son chemin d'accès et les prédictions.

| | ImageId | PredictionString | train_img | train_masks |
|---|---|---|---|---|
| 0 | ID_8a6e65317 | 16 0.254839 -2.57534 -3.10256 7.96539 3.20066 ... | Data/train_images/ID_8a6e65317.jpg | Data/train_masks/ID_8a6e65317.jpg |
| 1 | ID_337ddc495 | 66 0.163988 0.192169 -3.12112 -3.17424 6.55331... | Data/train_images/ID_337ddc495.jpg | Data/train_masks/ID_337ddc495.jpg |
| 2 | ID_a381bf4d0 | 43 0.162877 0.00519276 -3.02676 2.1876 3.53427... | Data/train_images/ID_a381bf4d0.jpg | Data/train_masks/ID_a381bf4d0.jpg |
| 3 | ID_7c4a3e0aa | 43 0.126957 -3.04442 -3.10883 -14.738 24.6389 ... | Data/train_images/ID_7c4a3e0aa.jpg | Data/train_masks/ID_7c4a3e0aa.jpg |
| 4 | ID_8b510fad6 | 37 0.16017 0.00862796 -3.0887 -3.04548 3.4977 ... | Data/train_images/ID_8b510fad6.jpg | Data/train_masks/ID_8b510fad6.jpg |

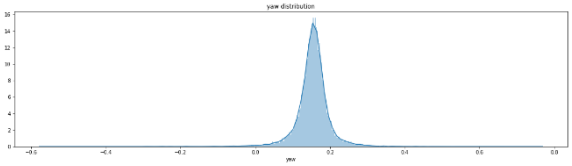
# Analyse exploratoire

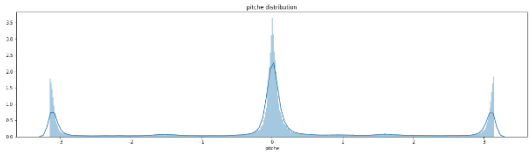
On commence par observer le nombre de voiture par image ainsi que les différents types de modèles

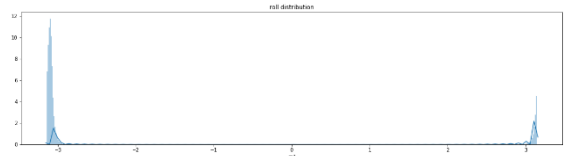


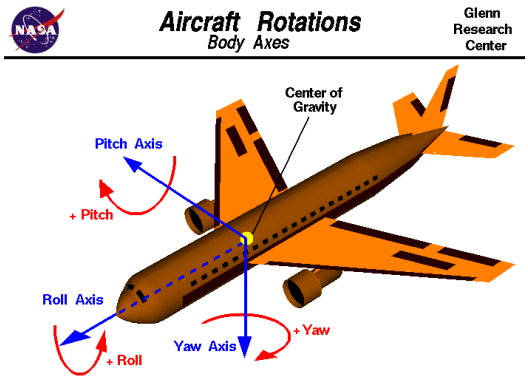




On observe également la distribution des valeurs yaw, pitch et roll correspondant aux différents axes de rotations



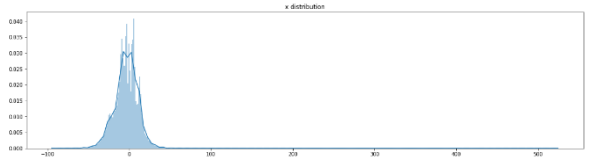


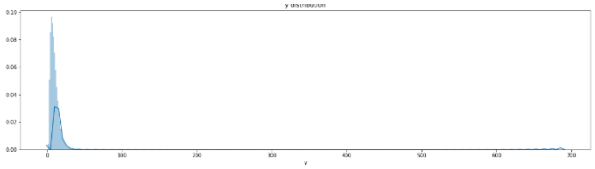


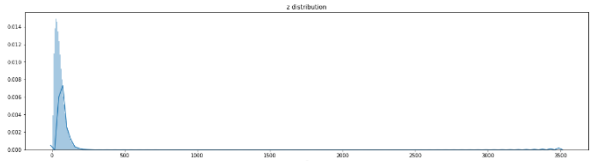



- Yaw correspond à l'axe de rotation vertical. Il indique donc le sens de circulation du véhicule
- Pitch correspond à l'inclinaison sur un axe horizontale.
- Roll correspond au troisième axe.

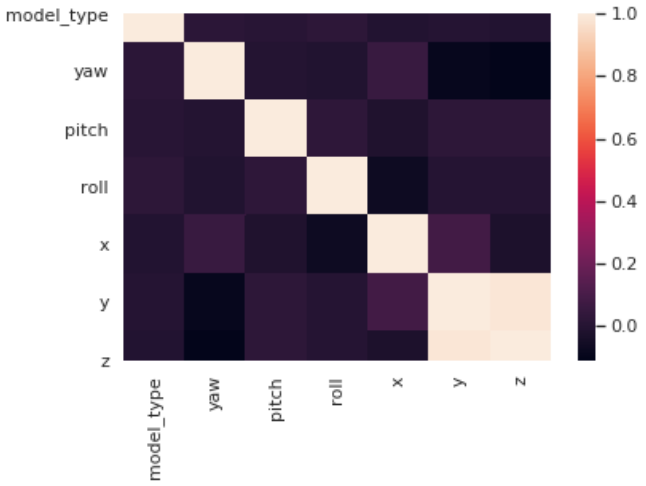
On observe également la représentation des distances x, y et z



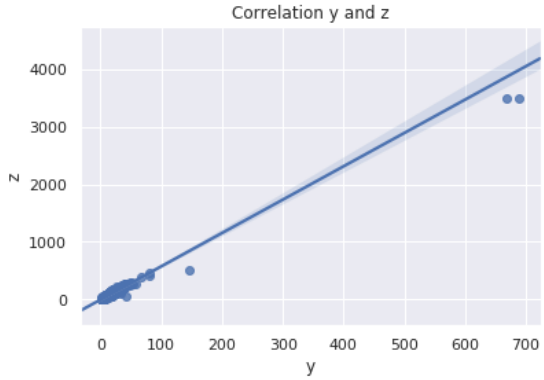




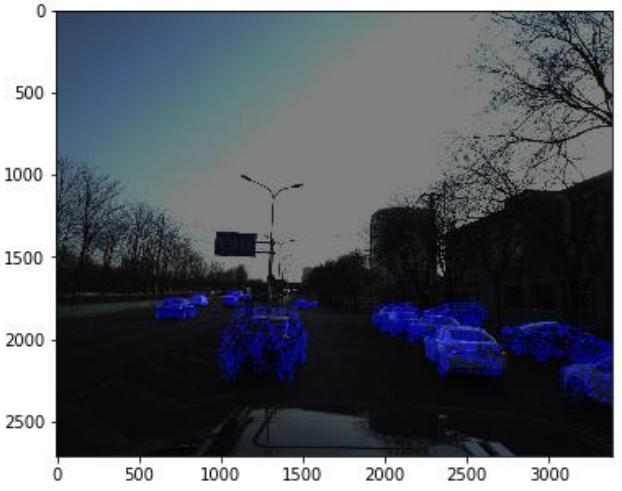
Une fois que l'on a observé la distribution de ces valeurs, on doit chercher les différentes corrélations possibles. Une façon d'observer les corrélations est d'afficher la matrice de corrélation.




Nous observons que la seule corrélation linéaire possible est la relation qui lie y et z




Observons maintenant, différentes observations liant les images et les annotations. Nous pouvons donc observer cette représentation à gauche permettant de visualiser les représentations en 3D des modèles sur les positions des véhicules ou simplement la représentation de droite permettant d'obtenir un point au centre de chaque véhicule.



Nous comprenons maintenant le lien entre les images et les annotations.

# Préparation des données

## Redimensionnement des images

Pour pouvoir importer les données dans un modèle Keras pour que l'algorithme apprenne correctement, nous devons transformer les données. Nous devons par souci de ressources disponibles réduire la taille des images sans déformations. Nous utilisons donc l'algorithme suivant pour redimensionner les images à 512 px / 512 px.

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

```

## Séparation du jeu d'entrainement de test et de validation

Nous utilisons une fonction pour créer trois dossiers contenant les images sélectionnées aléatoirement. Un dossier d'entrainement contenant 60% des données, un dossier de test contenant 20% des données et un dossier de validation contenant les 20% de données restantes.

Nous ne faisons pas de data augmentation dans ce cas précis car cela faussera les résultats. Par exemple, une rotation de l'image ou un effet miroir fausserai la correspondance entre les images et les annotations, alors que le redimensionnement sans déformation est une transformation linéaire qui permet de conserver les proportions.

On obtient ainsi 3 listes de données correspondantes aux id des images des 3 dossiers

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

On finit cette étape par créer les différents DataFrame contenant les entrées et les valeurs cibles du modèle. Par exemple pour déterminer le nombre de voiture par images, on peut utiliser ce code en prenant soin de transformer les sorties en divisant par le nombre maximal de voiture (ici 44)

```
x_train = train['train_img'][liste_index_train]
x_test = train['train_img'][liste_index_test]
x_val = train['train_img'][liste_index_val]
y_train = liste['nb_car'][liste_index_train]/44
y_test = liste['nb_car'][liste_index_test]/44
y_val = liste['nb_car'][liste_index_val]/44

# Pour l'entrainnement
new_train = pd.DataFrame({"x":x_train})
new_train = new_train.join(y_train)

new_test = pd.DataFrame({"x":x_test})
new_test = new_test.join(y_test)

# Pour la validation
new_val = pd.DataFrame({"x":x_val})
new_val = new_val.join(y_val)
# On corrige les chemins d'accès
new_train['x'] = new_train['x'].str.replace('train_images', 'Train')
new_test['x'] = new_test['x'].str.replace('train_images', 'Test')
new_val['x'] = new_val['x'].str.replace('train_images', 'Val')
```

## Créations des fonctions d'évaluations

On créer des fonctions de scores et de pertes. N'ayant pas trouvé de fonction implémentant correctement le mean average précision dans Keras, j'ai décidé d'utiliser pour mes modèles une fonction de perte correspondant à la somme des erreur absolue

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

# Modélisation

## Sélection du modèle le plus performant

Toujours dans un souci de manque de ressources, les modèles utilisant YoloV3 et CenterNet ne fonctionne pas efficacement et rapidement sur google colab. Nous allons donc implémenter pour chaque variable cible un algorithme de réseau de convolution suivit d'un réseau entièrement connecté permettant d'effectuer une régression.

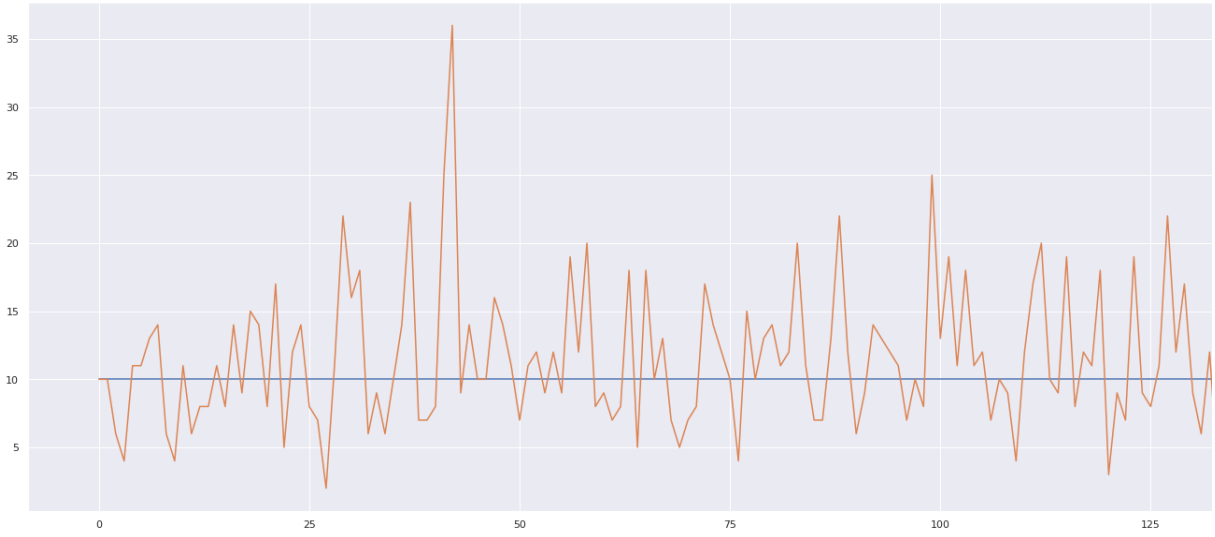
J'ai donc choisi de décomposé le problème en plusieurs problèmes plus simple.

J'ai dans un premier temps testé différentes architectures de réseau de neurone à convolution et approche par transfert learning pour obtenir une comparaison des performances et surtout de temps d'apprentissage.

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
_________________________________________________________________
```

Observons les résultats de la modélisation sur les données de test.

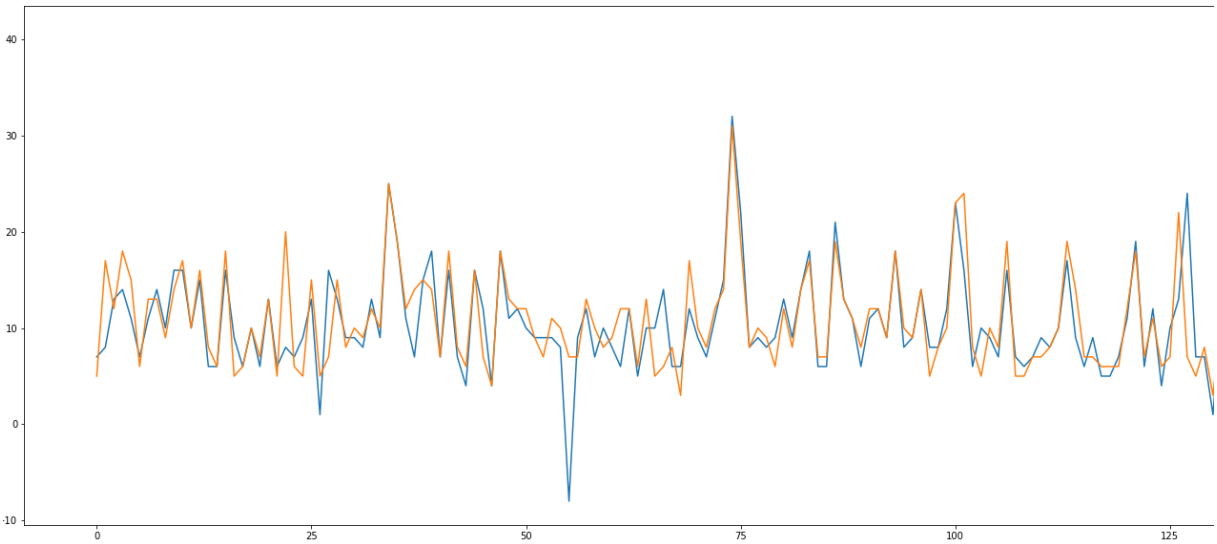



L'algorithme n'apprend pas correctement

Comparons à un CNN classique de 6 couches de convolution + batch_normalisation + max_pooling  sur lequel on ajoute une couche de flatten pour applatir les données une couche de dense de 1024 neurones, un dropout et la couche de sortie

```
_________________________________________________________________
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
_________________________________________________________________
```

On observe sur les données de validation en sortie des résultats qui suivent les valeurs cible




On compare les performances

| | 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 modèle le plus interressant est donc le CNN + Régression qui est plus léger donc plus rapide et qui obtient de meilleurs performances

## Optimisation du modèle

On cherche à optimiser le modèle CNN + régression

| 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 |

On observe des meilleures performances sur le test 6.

On peut entrainer 307 modèles sur chacune des 307 valeurs cibles. On récupère à chaque fois les poids dans un fichier.

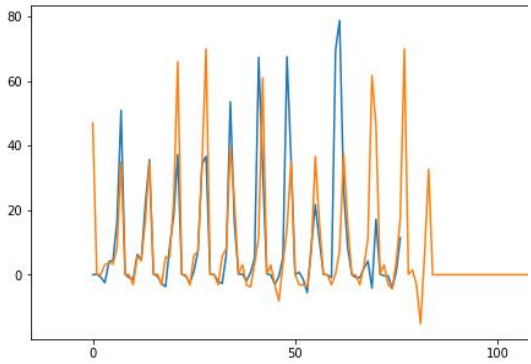
## Evaluation

Pour évaluer le modèle, on commence charger l'ensemble des modèles sur l'ensemble des images. On stocke ces valeurs dans une liste puis on importe les données dans un dataframe.

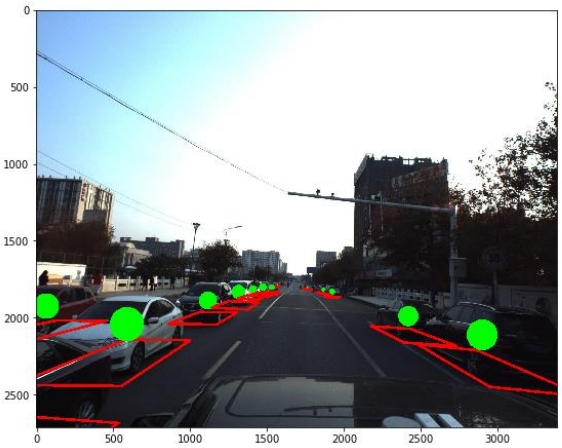
```
model_name = 'nb_car.h5'
first_model.load_weights('/content/' + model_name)
# import val
y_hat_total = []
for i in list(range(len(val_set.filepaths))):

    test_image = Image.open(val_set.filepaths[i])
    test_image_full = np.array(test_image)
    test_image = test_image.resize((512, 512), resample=0)
    test_image = np.array(test_image)/255
    test_image = np.expand_dims(test_image, axis=0)
    result = int(first_model.predict(test_image)*44)
    y_hat = []
    for i in list(range(1,141)):
        if i <= (7*result-1):
            model_name = 'model' + str(i) + '.h5'
            train_model.load_weights('/content/' + model_name)
            result2 = train_model.predict(test_image)
            y_hat.append(result2[0][0])
        else:
            y_hat.append(None)
    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
```




On observe donc les écarts entre l'ensemble des valeurs prédite et des valeurs réelles





On observe que l'algorithme n'est pas suffisamment efficace pour pouvoir développer un système de reconnaissance de positions des voitures, il n'en détecte que quelques-unes.

# Déploiement sur le jeu de test fourni

On déploie l'algorithme sur l'ensemble des valeurs test fourni sur le dataset pour obtenir les prédictions. On transforme les résultats pour obtenir le format demandé. On exporte le fichier csv de soumission et on poste le résultat sur la plateforme Kaggle.

Comme attendu le résultat est très faible.