Extraction des features réalisée sur le training dataset (données avec des labels)

L'histogramme des gradients orientés (**HOG**) est réalisé sur un set de données sur lesquelles ont déjà été assigné des labels (**Vehicles** et **Non-Vehicles**). Cette étape d'assignation de labels a été réalisée après le découpage de la dataset (**Test/Training**) avant la réalisation de l'extraction des features.

Remarque : un gradient représente la variation d'une fonction par rapport à la variation de ses différents paramètres. Dans notre cas, on travaille avec des variations de x et y de la courbe constituant la forme présente dans l'image.

L'utilisation des gradients est très intéressante car on observe une magnitude plus élevée lorsque nous sommes confrontés à des coins ou des bords. Cela est intéressant puisque les coins et les bords donnent énormément d'informations concernant la forme d'un objet par rapport à des régions plus plates. On arrive donc à cerner les images contenant des véhicules de celles contenant des non-véhicules.

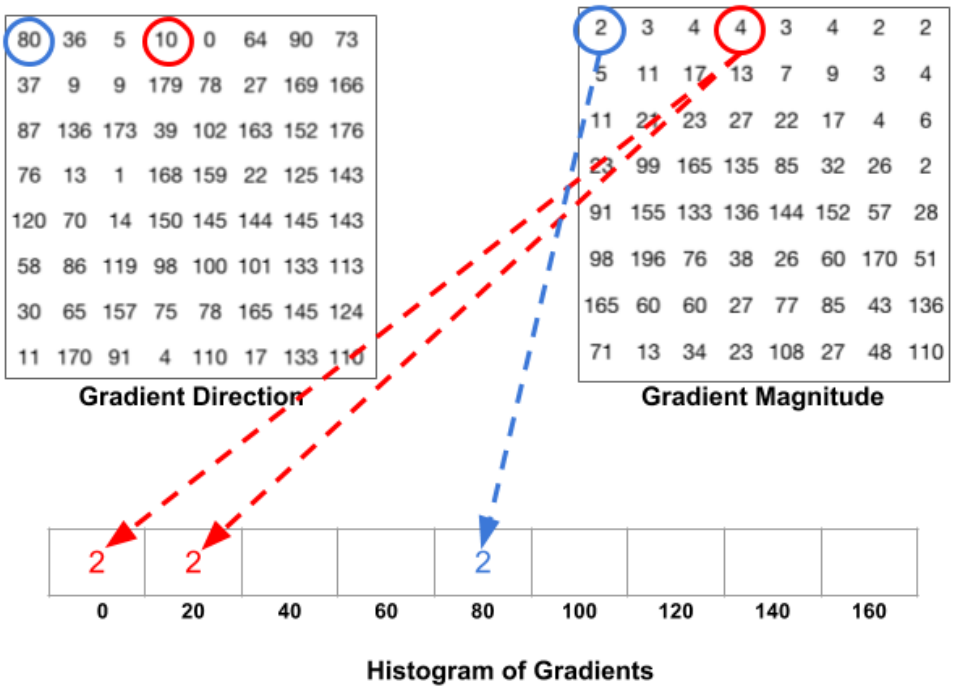
L'extraction des features est importante puisque nous utilisons ensuite une SVM. Nous avons utilisé une fonction très intéressante de la libraire **skimage** qui est **hog**. De manière générale, la réalisation de l'algorithme se réalise en cinq étapes :

- Une normalisation globale optionnelle
- Le calcul des gradients des images en x et y
- Le calcul de l'histogramme des gradients
- Une normalisation entre blocs
- Concaténation en UN vecteur de features

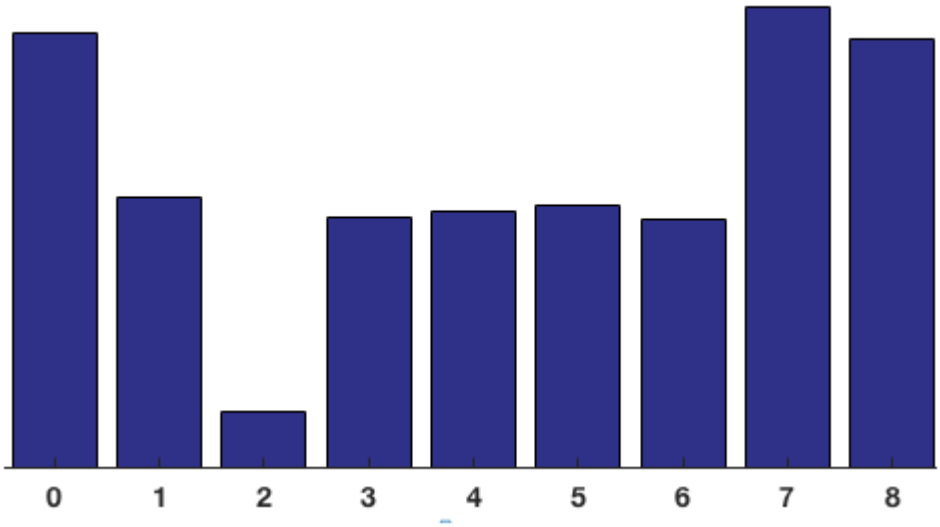
La 1ère étape est optionnelle et elle a pour objectif de diminuer l'influence de la luminosité sur les images.

La 2ème étape correspond à la réalisation du gradient. Les zones les plus sombres possèdent de plus grandes valeurs.

La troisième étape est une détermination locale des gradients afin de déterminer des histogrammes locaux pour chaque bloc qu'on appelle **cell** (ex : 8x8 pixels).



Cette image montre la réalisation d'un histogramme locale (figure ci-dessous) sur une seule cell 8x8. On a ensuite créé des **bins** correspondant aux différentes orientations des gradients et dans lesquels on va accumuler des « votes » correspondant aux différentes valeurs associées à chaque pixel.



La 4ème étape consiste à normaliser des blocs entre eux en les groupant, par exemple, par 4. Tout cela, afin de faciliter la 5ème étape consistant à regrouper les HOG descriptors de tous les blocs en un seul vecteur afin de pouvoir l'utiliser dans la fenêtre de classification.

Cette fonction **hog** de **skimage** prend les paramètres suivants en input :

skimage.feature.**hog** **(***image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(3, 3), block_norm=None, visualize=False, visualise=None, transform_sqrt=False, feature_vector=True, multichannel=None***)**

On retrouve :

- L'image (provenant de nos différents folders Vehicles et Non-Vehicles)
- Le nombre de bins d'orientations souhaité pour l'histogramme
- Le nombre de pixels par cellule
- Le nombre de cellule par bloc
- Une normalisation optionnelle
- Une visualisation sur l'image
- Une transformation avec une loi de compression
- Le choix de retourner un vecteur de features

En sortie, on obtient un vecteur 1D correspondant aux features d'une seule image.

Dans notre code, on a :

- Image en input
- 12 bins d'orientations
- 8x8 pix/cellules
- 1 cellules/blocs

On retrouve tout cela dans la fonction **car_find_roi** :

```
def car_find_roi(self, img, size, roi, overlap)
```

Il y a en tout TROIS TYPES de features (Images entières convertie en LUV et redimensionnées, histogrammes de l'espace couleur LUV et HOG)

1) On prend d'abord les images qui vont être convertie en LUV et normalisée avant d'être utilisée en tant que feature. Cela nous donne un 1er set de features.

2) Histogramme de couleurs où on a choisi 128 bins (intensité allant de 0 à 255)

```
hist_l = np.histogram(img_LUV[:, :, 0], bins=self.hist_bins, range=self.hist_range)
width = 0.7 * (hist_l[1][1] - hist_l[1][0])
center = (hist_l[1][:-1] + hist_l[1][1:]) / 2
```

```
cf = CarFinder(64, hist_bins=128, small_size=20, orientations=12, pix_per_cell=8, cell_per_block=1,
                   classifier=cls, scaler=scaler, window_sizes=window_size, window_rois=window_roi,
                   transform_matrix=perspective_transform, warped_size=UNWARPED_SIZE,
                   pix_per_meter=pixels_per_meter)
```

3) HOG est calculé sur des images de 64x64 pixels avec 8x8 pixels par cellule et 1 cellule 8x8 par bloc. On calcule ce HOG sur les trois canaux différents de couleurs LUV.

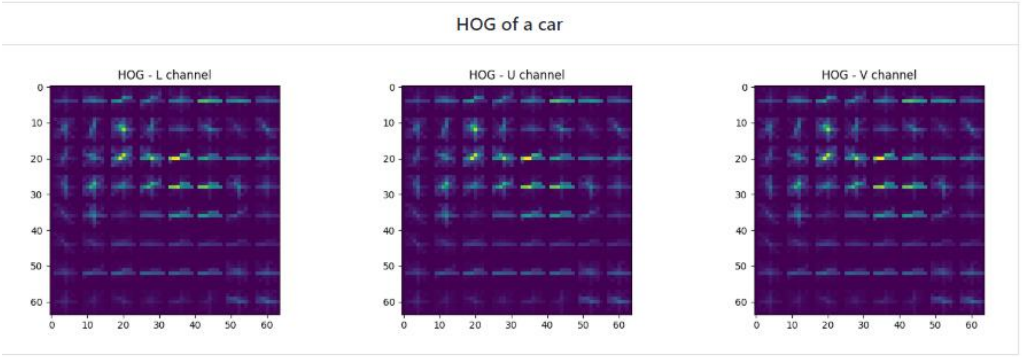
Ce bloc va parcourir l'image cellule par cellule en s'assurant de toujours prendre 8x8.

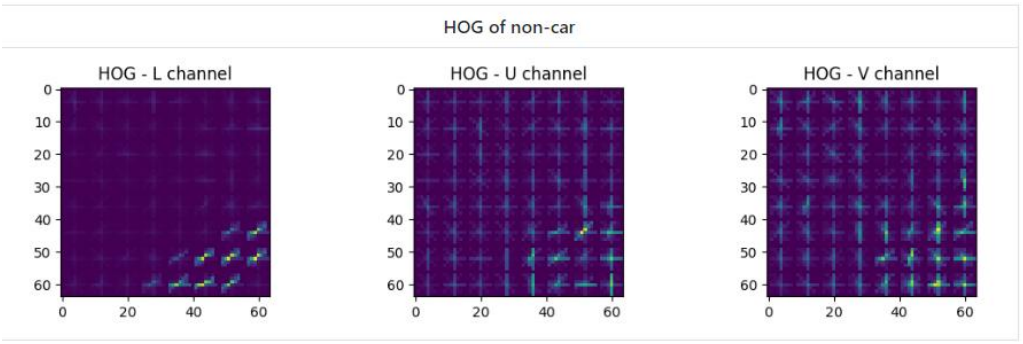
Voir animation l'étape 4 : https://www.learnopencv.com/histogram-of-oriented-gradients/

```
img_hog_l = hog(img_roi[:, :, 0], self.orientations, pixels_per_cell=(self.pix_per_cell, self.pix_per_cell),
                cells_per_block=(self.cell_per_block,  self.cell_per_block), transform_sqrt=False,
                feature_vector=False)
img_hog_u = hog(img_roi[:, :, 1], self.orientations, pixels_per_cell=(self.pix_per_cell, self.pix_per_cell),
                cells_per_block=(self.cell_per_block,  self.cell_per_block), transform_sqrt=False,
                feature_vector=False)
img_hog_v = hog(img_roi[:, :, 2], self.orientations, pixels_per_cell=(self.pix_per_cell, self.pix_per_cell),
                cells_per_block=(self.cell_per_block,  self.cell_per_block), transform_sqrt=False,
                feature_vector=False)
```

Sauf que dans notre cas, c'est avec un bloc de 8x8. On obtient alors 15x15 = 225 features pour un canal de couleur et 3x225 = 675 features pour 3 canaux de couleurs.

Finalement, on combine toutes les features afin d'obtenir **UN** vecteur correspondant à l'image complète + 675 features de couleurs LUV + 675 features de HOG = 1351 features (POUR CHAQUE IMAGE).





Remarque : la fonction **get_features** est utilisée pour entraîner le classificateur et on y réalise également une conversion de l'espace de couleurs RGB vers LUV afin d'obtenir de meilleures performances.

On a donc le return de la fonction **get_features** qui fournit les features à la SVM dans le script **train_svm.py** :

```
return np.hstack((img_feature.ravel(), hist_l[0], hist_u[0], hist_v[0], features_l, features_u, features_v))
```

Dans **train_svm.py** :

```
from car_finder import CarFinder

vehicle_dir = "train_images/vehicles"
non_vehicle_dir = "train_images/non-vehicles"
total_imgs = 0
```

On va chercher nos images dans les dossiers ci-dessus et dans les sous-dossiers respectifs et on fait appel à la fonction **get_features** qu'on va stocker dans un array. Chaque image correspond à un sample.

```
def create_features_from_dir(root_dir):
    all_features = []
    for subdir in (os.listdir(root_dir)):
        for file in os.listdir(os.path.join(root_dir,subdir)):
            img = mpimg.imread(os.path.join(root_dir,subdir,file))
            img = (img.astype(np.float32)/np.max(img)*255).astype(np.uint8)
            features = car_finder.get_features(img)
            all_features.append(features)
    return np.vstack(all_features)
```

La fonction **create_features_from_dir** est lancée sur les dossiers et sous-dossiers de véhicules et de non-véhicules.

```
X_cars = create_features_from_dir(vehicle_dir)
y_cars = np.ones(X_cars.shape[0], dtype=np.uint8)
X_non_cars = create_features_from_dir(non_vehicle_dir)
y_non_cars = np.zeros(X_non_cars.shape[0], dtype=np.uint8)
```

On va ensuite les stacker => on aura un array X contenant DEUX éléments (X_cars, X_non_cars) et chacun des éléments contient leur features.

```
X = np.vstack((X_cars, X_non_cars))
y = np.concatenate((y_cars, y_non_cars))

scaler = StandardScaler().fit(X)
scaled_X = scaler.transform(X)
```

On utilise ensuite un **scaler** de **scikit-learn** (sklearn) afin de normaliser l'array X. Ce dernier contient n_samples contenant chacun n_features. Voici un exemple :

`data = [[0, 0], [0, 0], [1, 1], [1, 1]]` => 4 samples et chaque sample a deux features

La fonction **fit()** réalise le calcul de la moyenne et l'écart type de l'array X et la fonction **transform()** réalise la remise au centre et remise à l'échelle des éléments (features) de X.

```
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=0.10, random_state=40)
```

On va ensuite diviser notre array scaled_X en subsets aléatoires de training et de test. Test_size correspond à un pourcentage de scaled_X et il vaut 10%.

On va finalement instancier un classificateur provenant de **LinearSVC** de **scikit-learn**.

```
cls = LinearSVC(C=1e-4, dual=False, max_iter=5)
cls.fit(X_train, y_train)
result = cls.predict(X_test)
```

Le classificateur est entrainé avec un paramètre de pénalité du terme d'erreur de 0.001. Ce paramètre a été déterminé en se basant sur les précisions des sets de training et de test. Lorsque la différence entre ces deux précisions est élevée => overfitting the data => il faut diminuer le C et lorsque la précision du set de test est faible mais identique à la précision du set de training => underfitting et il faut augmenter le C => un juste milieu !

Une précision finale aux alentours de 99% a été obtenue. Malgré cette grande précision, nous avons tout de même une erreur de type II (False Negative : On ne détecte pas de véhicule alors qu'il y a un véhicule => danger) de l'ordre de 50 ~ 60%. Cette erreur est liée à la faible qualité de certaines données du training data (voir dump folder).

On sauvegarde ensuite les données du classificateur pour l'utiliser sur nos images/videos.