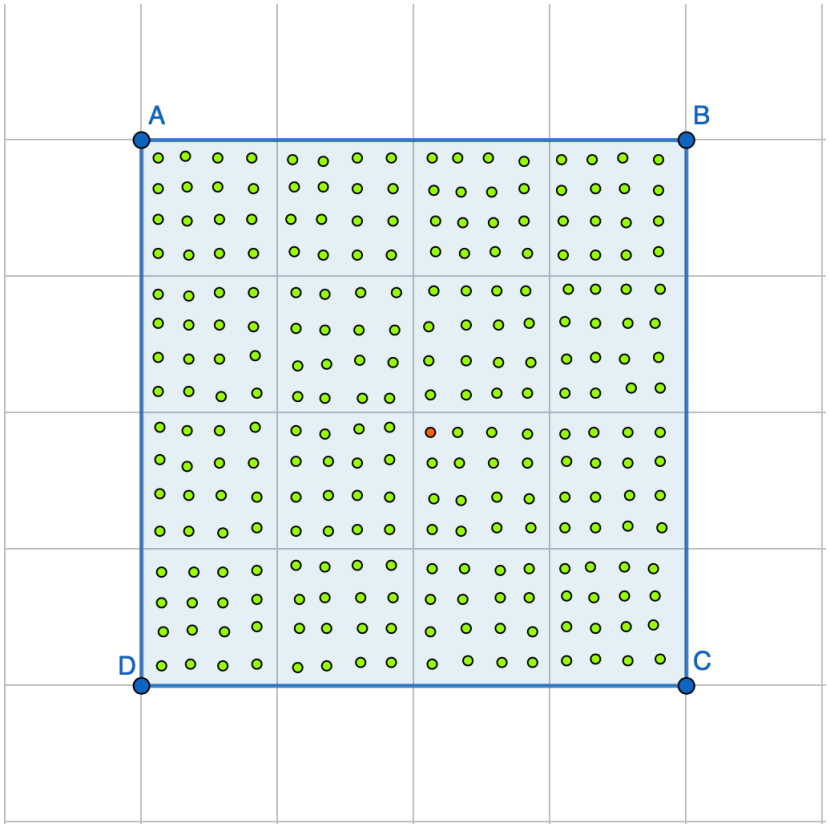
# ASSIGNMENT 5 - CV REPORT

## Bag-of-words Classifier

The function **grid_points** takes as input the image vector as well as the dimensions of the grid, returning an array containing 2D point coordinates. My implementation has been done using the numpy *linspace* function, which returns evenly spaced numbers over a specified interval. A border of 8-pixels was excluded in the process, as suggested in the assignment.




*Here is the location of the grid-point (red) with respect to the set of cells.*

The function **descriptors_hog** returns a set of the descriptors for each image. After having considered a 4 x 4 set of cells around each grid-point, an histogram of the gradient orientations is computed for each 4x4 pixel cell. Since the histogram is constructed using 8 bins, by concatenating all of the 16 histogram vectors, we encode the information regarding a single grid-point in a 8 * 16 = 128 dimensional vector. This procedure is repeated for each of the N grid-points in the image. The histograms are then grouped in a N x 128 matrix, using the numpy *vstack* function.

It is worth noticing that we may consider the gradient of the same pixel more than once, since the number of grid points and the dimensions of the cells are fixed regardless of the magnitude of the image. In other words, if the image is not big enough and the grid points are not far enough from each other, the same set of points may be part of two adjacent cells. Moreover, the 8-bins of the orientations' histograms are defined on the interval $(-\pi,\pi)$, following the convention of the *arctan2* function used to compute the gradient angle. Dividing the interval in eight bins means that the splits occur at remarkable angle values: $(-(3/4)\pi,\ -\pi/2,\ -\pi/4, 0, \pi/4, \pi/2, (3/4)\pi)$. Whenever the orientation of the gradient takes one of these values, ambiguity might arise over which bin the gradient belongs to.
In the function **create_codebook**, a visual vocabulary is constructed. For each image in the training dataset, the 100 grid-points are extracted and the descriptors are computed. Each image should feature descriptors of shape (100,128) where 100 is the number of grid-points in the image. Finally, the K-means clustering algorithm takes as input the entire descriptors matrix, where all the descriptors from all images are stacked on top of each other, and returns k cluster centers. The vCenters matrix has shape (K,128), where K is the number of clusters to be chosen as an hyperparameter in the K-means algorithm.
It is important to point out that the outcome of K-means depend on the random initial clustering. Two hyperparameters have to be tuned for the K-means algorithm:

- K: it is the number of cluster centers; a significant drawback of the algorithm is indeed to choose the number of clusters
- Numiter: the number of iterations of the algorithm

After having experimented different values for these parameters, I opted for K = 10, Numiter = 500. The testing yields an accuracy well above 0,75(required baseline): usually the accuracy is above 0.95 both for the positive and for the negative test set.

In the function **bow_histogram**, the idea is to build an histogram that counts how many features for each image are assigned to each cluster. The function **findnn,** which was already implemented, is used to compute the closest center for each feature. The histogram returned by the function is a K-long vector.

In the function **create_bow_histogram**, looping through the images, a bag of word histogram is built. For every image, the bow_histogram is computed using the previous function. The histograms are then stacked up together to build the object vBow.

Finally, within the function **bow_recognition_nearest**, we use the definition of 2-norm to find DistPos and DistNeg:

- DistPos is the minimum distance between the testing image and the BoW histogram of positive training images.
- DistNeg is the minimum distance between the testing image and the BoW histogram of negative training images.

If DistPos < DistNeg, the image is labelled as 1, which means a car is present in the image. Otherwise, the image is labelled 0, suggesting there is no car in the image

# CNN-based Classifier

The convolutional neural network was built following the simplified version of VGG network proposed in the assignment.To build the correct structure, padding was added for every convolutional layer. In the classifier block, I opted for a middle layer of size **32,** whereas the dropout probability was set to **0.5**.

The **test accuracy** on CIFAR-10 test set is:

```
test accuracy: 79.85
```

Here are the screenshots of the training loss and of the validation accuracy curve, provided by Tensorboard::