detection

July 19, 2019

1 import

```
In [1]: import cv2
        print(cv2.__version__)
        import matplotlib
        import matplotlib.pyplot as plt
        import numpy as np
        from numpy import pi
        import pandas as pd
        import os
        from numba import jit
```

```
4.1.0
```

2 File selection

```
In [2]: path = "/Users/oliviermanette/Desktop/trailer detection challenge/data/P473_Arizona_Da
        os.chdir(path)

In [3]: pwd

Out[3]: '/Users/oliviermanette/Desktop/trailer detection challenge/data/P473_Arizona_Day_Asphal

In [4]: ls

P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.avi*
P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.dat_GT.csv*


In [5]: #fileName='W420_ES_Hi_Snow_Slush_Asphalt_28klux.avi'
        fileName = 'P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.avi'
```

3 Test Video Loop

```
In [6]: cap = cv2.VideoCapture(fileName)  # load the video
        while (cap.isOpened()):  # play the video by reading frame by frame
```

```
            ret, frame = cap.read()
            if ret == True:
                # optional: do some image processing here
                cv2.imshow('frame', frame)
                # show the video
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    #if 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()
        cv2.destroyAllWindows()
```

4 Variables globales

Pour des raisons de lisibilité du code, l'ensemble des variables locales seront précédés du préfixe 'l' afin de les différencier des variables globales qui n'ont pas de préfixe. ## Type de données de position

```
In [7]: posType = np.dtype([('x', 'u1'), ('y', 'u2')])
```

4.1 Type de données de Neurones

```
In [8]: NeuronType = np.dtype([('longueur', 'u1'), ('angle', 'f4'), ('weight', 'f4'),
                               ('precision', 'f4'), ('xPos', 'u1'), ('yPos', 'u2'),
                               ('group', 'u1'), ('layer', 'u1')])
```

4.2 Taille des champs récepteurs neuronaux

```
In [9]: tailleField = 7
```

5 Fonctions

5.1 Calcul d'un neurone champ moyen

A partir d'une liste de neurones, il retourne le neurone moyen

```
In [388]: def getAvgFieldNeuron(lNeuronList, typeList=NeuronType):
              lNeurons = np.zeros(1, dtype=typeList)
              lpNeurons = pd.DataFrame(lNeurons)
              lpNeurons['longueur'] = int(lNeuronList.longueur[0:1])
              lpNeurons['angle'] = float(
                  np.sum((lNeuronList.angle * lNeuronList.weight) /
                         np.sum(lNeuronList.weight)))
              lpNeurons['weight'] = float(
                  np.sum((lNeuronList.weight * lNeuronList.weight) /
                         np.sum(lNeuronList.weight)))
              lpNeurons['precision'] = float(
```

```
            np.sum((lNeuronList.precision * lNeuronList.weight) /
                   np.sum(lNeuronList.weight)))
        lpNeurons['xPos'] = np.around(
            np.sum((lNeuronList.xPos * lNeuronList.weight)) /
                   np.sum(lNeuronList.weight))
        lpNeurons['yPos'] = np.around(
            np.sum((lNeuronList.yPos * lNeuronList.weight)) /
                   np.sum(lNeuronList.weight))
        return lpNeurons
```

5.2 Matrice des directions

Afin de faciliter le calcul des angles des pixels, une matrice de poids est générée afin d'appliquer à chaque pixel centré sur un champs récepteur un poids correspondant à l'angle d'une ligne passant par ce centre. Voici comment les angles sont représentés IMAGE

```
In [11]: @jit(nopython=True, parallel=True)
         def fillAngleMat(lSize):
             lOutput = np.zeros((lSize, lSize))
             lOffset = int(np.floor(lSize / 2))
             for lX in range(0, lSize):
                 for lY in range(0, lSize):
                     if (lX - lOffset) == 0:
                         lOutput[lX, lY] = 90
                     else:
                         lOutput[lX, lY] = 0.01 + np.around(
                             np.arctan((lY - lOffset) / (lOffset - lX)) / pi * 180, 2)
             lOutput[lOffset, lOffset] = 0
             return lOutput
```

5.3 Fonction d'activation des neurones

Chaque neurone retourne une valeur comprise entre 0 et 255 qui reflète son niveau d'activation. Cette activation reflète le niveau de confiance que le neurone a sur le lien existant entre sa fonction de base et les pixels reçus dans son champs récepteur. Plus les pixels sont organisés de façon à former une ligne avec l'angle correspondant à la fonction de base du neurone et plus ce dernier sera activé. Comme on ne souhaite pas obtenir une activation de valeur infinie, on utilise donc une fonction sigmoide qui s'applique à l'écart-type des angles supposés.

```
In [12]: @jit(nopython=True, parallel=True)
         def sigmoidActivationFctN1(lActivationVector):
             lDenom = (1 + np.exp(0.1 * (np.abs(np.std(lActivationVector)) - 30)))
             return 255 / lDenom
```

5.4 Création d'une liste de neurones à champs récepteurs

```
In [440]: #@jit(nopython=True, parallel=True)
          def getNeuronActivationList(idxX, idxY, size, frameE, nbPixelPts, layer=1, lVerbose=
              #commencer par créer le tableau de neurones
```

```
lNeuronType = np.dtype([('longueur', 'u1'), ('angle', 'f4'),
                        ('weight', 'f4'), ('precision', 'f4'),
                        ('xPos', 'u1'), ('yPos', 'u2'), ('group', 'u1')])
lCriterion = nbPixelPts >= size

nbNeurons = sum(lCriterion)
lNeurons = np.zeros(nbNeurons, dtype=lNeuronType)
lpNeurons = pd.DataFrame(lNeurons)
lpNeurons['longueur'] = size
lpNeurons['layer'] = layer

offsetField = int(np.floor(size / 2))
lAngleMat = fillAngleMat(size)

newX = idxX[lCriterion]
newY = idxY[lCriterion]
if lVerbose:
    print("size :" + str(len(newX)))

    print("newX")
    print(np.min(newX))
    print(np.max(newX))
    print("newY")
    print(np.min(newY))
    print(np.max(newY))
    print()
pos = 0
lnPos = 0
for lintX in newX:
    lintY = newY[pos]
    if (lintX - offsetField)<0 or (lintY - offsetField)<0:
        print("exceed the limit of the matrix")
        pos += 1
        continue

    lNeuronFieldFrame = frameE[
        int(lintX - offsetField):int(lintX + offsetField + 1),
        int(lintY - offsetField):int(lintY + offsetField + 1)] / 255

    try:
        tmp = np.multiply(lAngleMat, lNeuronFieldFrame)
    except:
        print("error 10 : ")
        print("lAngleMat")
        print(lAngleMat)
        print("lNeuronFieldFrame")
        print(lNeuronFieldFrame)
        print("lintX")
```

```
                print(lintX)
                print("lintY")
                print(lintY)
                print("offsetField")
                print(offsetField)
                continue

            lNeuronFieldValues = tmp[np.nonzero(tmp)]
            if (lNeuronFieldValues.size > 0):
                lpNeurons.loc[pos, ['angle']] = np.mean(lNeuronFieldValues)
                lpNeurons.loc[pos, ['weight']] = sigmoidActivationFctN1(
                    lNeuronFieldValues)
                lpNeurons.loc[pos, ['precision']] = np.std(lNeuronFieldValues)
                lpNeurons.loc[pos, ['xPos']] = lintX
                lpNeurons.loc[pos, ['yPos']] = lintY

                lnPos += 1
            else:
                True  #print ("error it shouldn't be zero")
            pos += 1
        if lVerbose:
            print("nb de positions couvertes : " + str(lnPos) + " sur " + str(pos))

        return lpNeurons
```

5.5 Nombre de pixels actifs dans chaque champs récepteur

A partir des coordonnées des centres supposés de chaque champs récepteurs et de la taille du champs récepteur, recherche sur la frame bitmap passée en paramètres, retourne un tableau contenant le nombre de pixels allumés à l'intérieur de chacun de ces champs.

```
In [14]: @jit(nopython=True, parallel=True)
         def nbPixelField(lTableX, lTableY, lFrameEdge, lintTailleField=3):
             lIdx = 0
             lResults = np.zeros(lTableX.size)
             lRayon = np.floor(lintTailleField / 2)
             lTailleMaxX = lFrameEdge.shape[0]
             #lTailleMaxY = lFrameEdge.shape[1]
             lHalfX = lTailleMaxX / 3

             for lPosX in lTableX:
                 lPosY = lTableY[lIdx]
                 if lPosX > lHalfX and lPosX >= lRayon and (lPosX +
                                                           lRayon) < lTailleMaxX:
                     lResults[lIdx] = np.sum(
                         lFrameEdge[int(lPosX - lRayon):int(lPosX + lRayon + 1),
                                    int(lPosY - lRayon):int(lPosY + lRayon + 1)] / 255)
                 lIdx += 1
```

```
            return lResults
```

```
In [15]: #@jit(nopython=True, parallel=True)
         def getNonZero(LImg):
             return np.where(LImg != [0])
```

5.6 Coordonnées de la fonction de base (ligne)

```
In [336]: def getNFCoordinate(lNeurone):
              try:
                  lintDist = int(np.floor(lNeurone.longueur / 2))
              except:
                  lP1 = (0, 0)
                  lP2 = (0, 0)
                  return (lP1, lP2)
              if np.abs(lNeurone.angle) < 45:
                  lAlpha = lNeurone.angle / 180 * pi
                  lintX1 = np.around(lNeurone.xPos - lintDist * np.tan(lAlpha))
                  lintY1 = lNeurone.yPos + lintDist
                  lintX2 = np.around(lNeurone.xPos + lintDist * np.tan(lAlpha))
                  lintY2 = lNeurone.yPos - lintDist
              else:
                  lAlpha = (90 - lNeurone.angle) / 180 * pi
                  #print("yPos = "+str(lNeurone.yPos)+"xPos = "+str(lNeurone.xPos))
                  lintY1 = np.around(lNeurone.yPos - lintDist * np.tan(lAlpha))
                  lintX1 = lNeurone.xPos - lintDist
                  lintY2 = np.around(lNeurone.yPos + lintDist * np.tan(lAlpha))
                  lintX2 = lNeurone.xPos + lintDist
              lP1 = (int(lintY1), int(lintX1))
              lP2 = (int(lintY2), int(lintX2))
              return lP1, lP2
```

```
In [323]: np.tan(0)
```

```
Out[323]: 0.0
```

5.7 Calcule la distance entre deux points

```
In [17]: def getDistance(lx1, ly1, lx2, ly2):
             return np.sqrt(
                 np.power(np.abs(lx1 - lx2), 2) + np.power(np.abs(ly1 - ly2), 2))
```

5.8 Retourne les neurones les plus proches d'un point

```
In [256]: def closestFieldNeurons(lneuronList, lposX, lposY, ldistance):

              return lneuronList[(lneuronList.xPos >= lposX - ldistance)
                                 & (lneuronList.xPos <= lposX + ldistance) &
                                 (lneuronList.yPos >= lposY - ldistance) &
                                 (lneuronList.yPos <= lposY + ldistance)]
```

5.9 Dessine les fonctions de base des neurones sur un bitmap

```
In [337]: def drawFieldNeurons(lNeuronList, lBitmap):
              for index, lNeuron in lNeuronList.iterrows():
                  #for lNeuron in lNeuronList:
                  lCoord = getNFCoordinate(lNeuron)
                  #print(lNeuron)
                  #print(lCoord)
                  try:
                      cv2.line(lBitmap, lCoord[0], lCoord[1], (int(
                          lNeuron.weight), int(lNeuron.weight), int(lNeuron.weight)), 1)
                  except:
                      True
              return lBitmap
```

5.10 Find neuronal groups

Un groupe neuronal est un ensemble de neurone dont les champs récepteurs sont complémentaires les uns des autres. Pour faire partie d'un champs récepteur, deux conditions doivent être réunies. (A compléter) ### Translation Retourne les coordonnées d'un point translaté d'une certaine distance avec un certain angle. Cette fonction demande un angle, une distance et les coordonnées d'un point de départ. Il retourne ensuite les coordonnées après translation.

```
In [417]: #@jit(nopython=True, parallel=True)
          def moveCoordDeg(langle, lstartX, lstartY, ldistance, lVerbose=False):
              if lVerbose:
                  ##DEBUG
                  print("moveCoordDeg(" + str(float(langle)) + "," +
                        str(int(lstartX)) + "," + str(int(lstartY)) +
                        "," + str(int(ldistance))+")")
                  ##DEBUG
              ltipX = lstartX + ldistance * np.cos(langle / 180 * pi)
              ltipY = lstartY + ldistance * np.sin(langle / 180 * pi)
              if lVerbose:
                  print ("coord ==> ("+str(ltipX)+","+str(ltipY))
              return ltipX, ltipY
```

Effectue le même calcul que la fonction moveCoordDeg mais prend comme paramètre un neurone. Il effectue la translation en prenant comme point de départ le centre du champs récepteur et effectue un déplacement de la taille de ce champs dans la direction de la fonction de base.

```
In [416]: def getNextPosition(lneuroneMoyen, lVerbose):
              return moveCoordDeg(float(lneuroneMoyen.angle), int(lneuroneMoyen.xPos),
                                  int(lneuroneMoyen.yPos), int(lneuroneMoyen.longueur), lVerbos
```

5.10.1 Calcul des groupes à partir d'une liste de neurones à champs récepteurs

```
In [436]: def findGroups(lneuronList, lVerbose=False):
              # Sélection d'un nouveau numéro de Groupe (GroupID)
```

```
lintCurrentGroupID = 0
lintNbGroups = 0
lIndex = 0

##DEBUG
lnbNeuron = 0
##DEBUG
# liste des neurones sans groupe
lNoGroupList = lneuronList[lneuronList.group == 0]

while lNoGroupList.shape[0] > 0:

    #Sélection d'un neurone dans la liste (ceux sans groupID ou groupID=0)
    lMoyenNeuron = lNoGroupList.iloc[0]
    lIndex = lNoGroupList.head().index.values[0]

    while True:
        #Assignation d'un nouveau numéro de GroupID en cours
        lintNbGroups += 1
        lintCurrentGroupID += 1
        if lneuronList[lneuronList.group ==
                       lintCurrentGroupID].shape[0] == 0:
            break

    lneuronList.loc[lIndex, ['group']] = lintCurrentGroupID

    #déplacement
    lnPos = getNextPosition(lMoyenNeuron, lVerbose)

    #recherche de neurones proches
    lClosestNeurons = closestFieldNeurons(
        lneuronList, lnPos[0], lnPos[1],
        int(np.floor(lMoyenNeuron.longueur / 2)))
    if lVerbose:
        print("")
        print("")
        print("Coordonnées en cours : (" + str(lnPos[0]) + "," +
              str(lnPos[1]) + ")")

        lnbNeuron += 1
        if lClosestNeurons.shape[0] == 0:
            print("Aucun neurone a proximité pour le neurone #" +
                  str(lnbNeuron) + " aux coordonnées : (" + str(lnPos[0]) +
                  "," + str(lnPos[1]) + str(") a la distance :") +
                  str(int(np.floor(lMoyenNeuron.longueur / 2))))

        #Oui ==> retour étape 1
        lNbFindGroup = 0
```

```
while lClosestNeurons.shape[0] != 0:
    #recherche des groupID dans cette sous-sélection
    if lClosestNeurons[lClosestNeurons.group > 0].shape[0] == 0:
        #Non => Assigner à tous les neurones de la sous-sélection
        #le groupID en cours => aller directement à l'étape 7
        if lVerbose:
            print("Aucun neurone dans le groupe : " +
                  str(lintCurrentGroupID))

        for lintIdx in lClosestNeurons.head().index.values:
            lneuronList.loc[lintIdx, ['group']] = lintCurrentGroupID
    else:
        #Oui
        if lVerbose:
            ##DEBUG
            #lNbFindGroup += 1
            print("Trouvé " + str(lClosestNeurons[
                lClosestNeurons.group > 0].shape[0]) +
                  " neurone(s) déja dans des groupes :")
            print("Groupe en cours : " + str(lintCurrentGroupID))

        #Récupération de la liste de tous les groupID utilisés
        #Sélection du groupID le plus petit
        #(en comparant aussi avec le groupID en cours)
        lintPreviousGroupID = lintCurrentGroupID
        lintCurrentGroupID = np.min(
            lClosestNeurons[lClosestNeurons.group > 0].group)
        if lVerbose:
            print("Change pour le groupe #" + str(lintCurrentGroupID))
            print("-")
        #Assigner au neurone en cours le nouveau groupe
        lneuronList.loc[lIndex, ['group']] = lintCurrentGroupID
        #Assigner à tous les neurones de la sous-sélection ce nouveau groupI
        for lintIdx in lClosestNeurons.head().index.values:
            lneuronList.loc[lintIdx, ['group']] = lintCurrentGroupID
            #remplacer dans la liste globale,
            #pour chaque groupID présent dans la liste par le nouveau groupI
            for lintGroupID in lClosestNeurons[
                    lClosestNeurons.group > 0].group:
                lneuronList.loc[lneuronList.group == lintGroupID,
                                'group'] = lintCurrentGroupID
        if lintPreviousGroupID == lintCurrentGroupID:
            #si tous les neurones
            if lClosestNeurons[lClosestNeurons.group >
                               0].shape[0] == lClosestNeurons[
                                   lClosestNeurons.group ==
                                   lintPreviousGroupID].shape[0]:
```

```
                    break  # sortie de la boucle while
            if lVerbose:
                #Calcul du neurone Field moyen
                print("Neurones trouvé :")
                print(lClosestNeurons)
            lMoyenNeuron = getAvgFieldNeuron(lClosestNeurons)
            if lVerbose:
                print("neurone Moyen")
                print(lMoyenNeuron)
            #déplacement
            lnPos = getNextPosition(lMoyenNeuron, lVerbose)

            #recherche de neurones proches
            lClosestNeurons = closestFieldNeurons(
                lneuronList, lnPos[0], lnPos[1],
                int(np.floor(lMoyenNeuron.longueur / 2)))

        lNoGroupList = lneuronList[lneuronList.group == 0]
    return lneuronList
```

6 Video Loop

In [23]:
```
kernelSize = 21  # Kernel Bluring size

# Edge Detection Parameter
parameter1 = 20
parameter2 = 40
intApertureSize = 1

#cap = cv2.VideoCapture(0)
cap = cv2.VideoCapture(fileName)
lCounter = 0
while (cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
        # Our operations on the frame come here
        if lCounter == 1:
            frame = cv2.GaussianBlur(frame, (kernelSize, kernelSize), 0, 0)
            frame = cv2.Canny(frame, parameter1, parameter2,
                              intApertureSize)  # Canny edge detection
            lCounter = 0
        lCounter += 1

        indices = np.where(frame != [0])
        # Display the resulting frame
        cv2.imshow('Edges Video', frame)
```

```
            if cv2.waitKey(1) & 0xFF == ord('q'):  # press q to quit
                break
        else:
            break
    # When everything done, release the capture
    cap.release()
    cv2.destroyAllWindows()
```

7 Sandbox

7.1 Toy data Generator

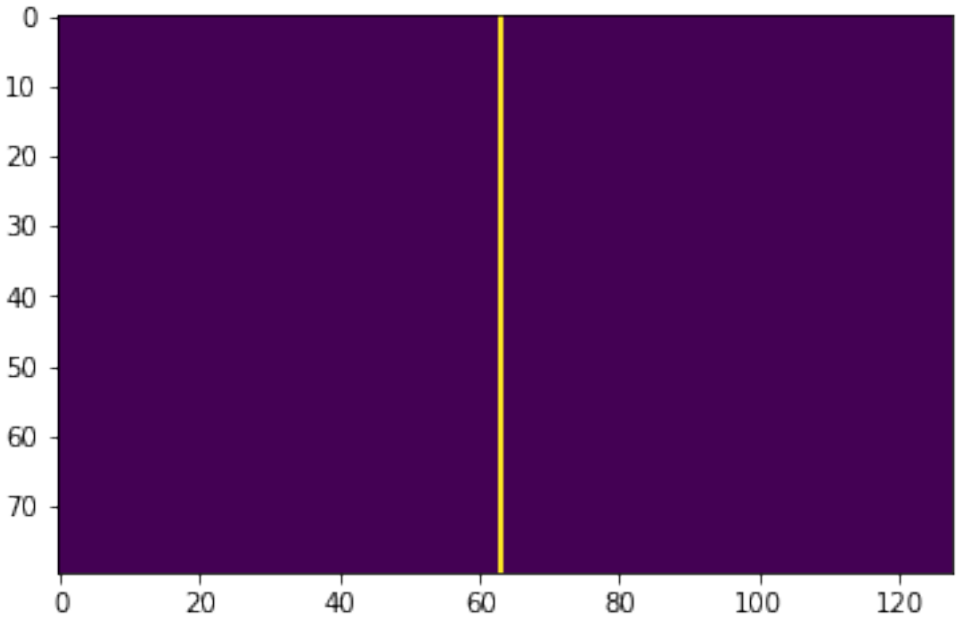
```
In [196]: def generateToy(lType=1, lHauteur=80, lLargeur=128,lepaisseur=1):
              lFrame = 0
              if lType == 1:
                  lFrame = np.zeros((lHauteur, lLargeur))
                  lFrame[:, int((lLargeur-lepaisseur) / 2):int((lLargeur+lepaisseur) / 2)] = 25
              elif lType == 2:
                  lFrame = np.zeros((lHauteur, lLargeur))
                  lFrame[int((lHauteur-lepaisseur) / 2):int((lHauteur+lepaisseur)/2), :] = 255
              elif lType == 3:
                  lFrame = np.zeros((lHauteur, lLargeur))
                  cv2.line(lFrame, (int(lLargeur / 3), lHauteur),
                           (int(2 * lLargeur / 3), 0), (255, 255, 255), lepaisseur)
              elif lType == 4:
                  lFrame = np.zeros((lHauteur, lLargeur))
                  cv2.rectangle(lFrame,
                                (int(lLargeur / 128 * 10), int(lHauteur / 80 * 30)),
                                (int(lLargeur / 128 * 30), int(lHauteur / 80 * 50)),
                                (255, 255, 255), lepaisseur)
                  pts = np.array([[int(lLargeur / 128 *64),
                                   int(lHauteur / 80 * 30)],
                                  [int(lLargeur / 128 * 76),
                                   int(lHauteur / 80 * 50)],
                                  [int(lLargeur / 128 * 53),
                                   int(lHauteur / 80 *50)]], np.int32)
                  ts = pts.reshape((-1, 1, 2))
                  cv2.polylines(lFrame, [pts], True, (255, 255, 255), lepaisseur)
                  cv2.circle(lFrame,
                             (int(lLargeur / 128 * 107), int(lHauteur / 80 * 40)),
                             int(lHauteur / 80 * 10), (255, 255, 255), lepaisseur)
              else:
                  lFrame = np.zeros((lHauteur, lLargeur))
                  print("First parameter should be between 1 to 4")
              return lFrame
```

7.2 Playground

7.2.1 Test 1

Generate data of type 1

```
In [407]: frame = generateToy(1,80,128,1)
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [441]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

In [442]: titi.describe()
```

Out[442]:

	longueur	angle	weight	precision	xPos	yPos	group
count	52.0	5.200000e+01	52.000000	52.0	52.000000	52.0	52.0
mean	3.0	9.999996e-03	242.906311	0.0	52.500000	63.0	0.0
std	0.0	3.761635e-09	0.000092	0.0	15.154757	0.0	0.0
min	3.0	1.000000e-02	242.906403	0.0	27.000000	63.0	0.0
25%	3.0	1.000000e-02	242.906403	0.0	39.750000	63.0	0.0
50%	3.0	1.000000e-02	242.906403	0.0	52.500000	63.0	0.0
75%	3.0	1.000000e-02	242.906403	0.0	65.250000	63.0	0.0
max	3.0	1.000000e-02	242.906403	0.0	78.000000	63.0	0.0

```
       layer
count   52.0
mean     1.0
std      0.0
min      1.0
25%      1.0
50%      1.0
75%      1.0
max      1.0
```

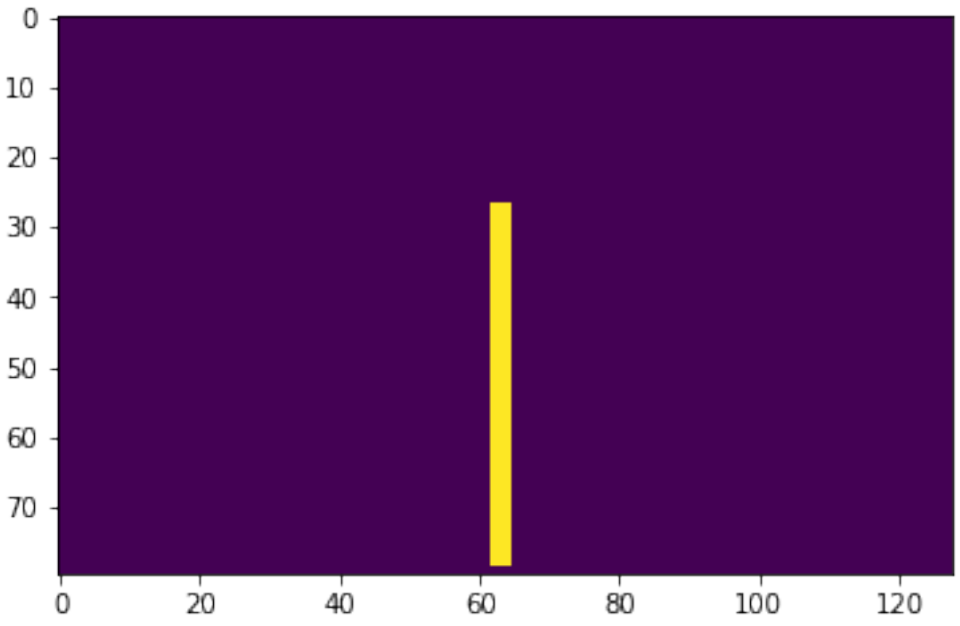
In [443]:
```
titi[0:4]
```

Out[443]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	3	0.01	242.906403	0.0	27	63	0	1
1	3	0.01	242.906403	0.0	28	63	0	1
2	3	0.01	242.906403	0.0	29	63	0	1
3	3	0.01	242.906403	0.0	30	63	0	1

Affichage graphique du champs récepteur des neurones

In [444]:
```
testBitmap = np.zeros(frame.shape)
testBitmap = drawFieldNeurons(titi, testBitmap)
imgplot = plt.imshow(testBitmap)
```



In [445]:
```
np.max(testBitmap)
```

Out[445]:
```
242.0
```

```
In [414]: lintI = 0
          while (lintI < 10):
              cv2.imshow('FRAME', frame)
              if cv2.waitKey(1) & 0xFF == ord('q'):  # press q to quit
                  break
              lintI += 1
```

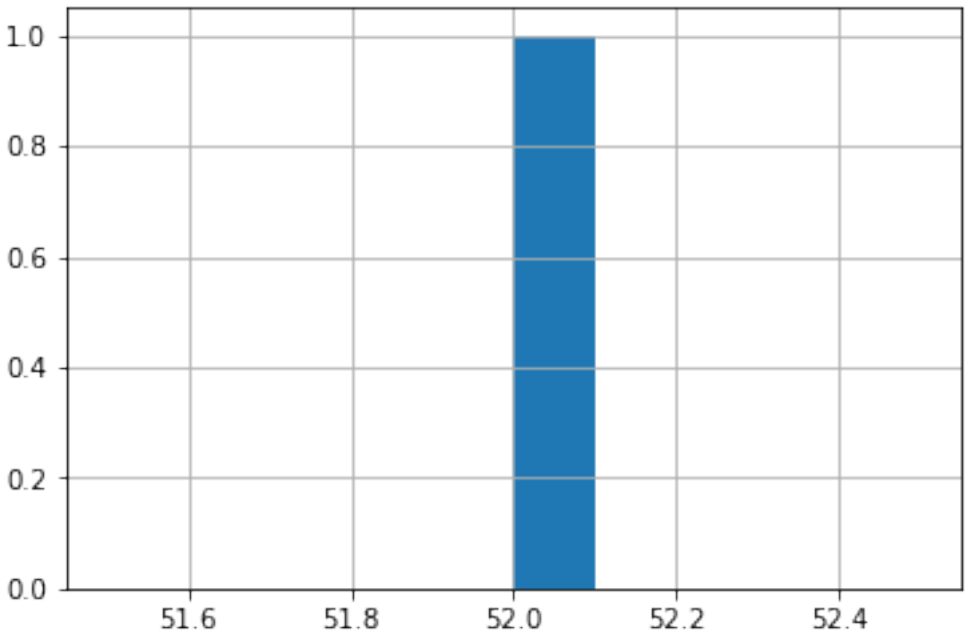
Génération des groupes

```
In [446]: findGroups(titi);

In [447]: titi.groupby('group').agg(['mean', 'count'])[0:5]
```

Out[447]:

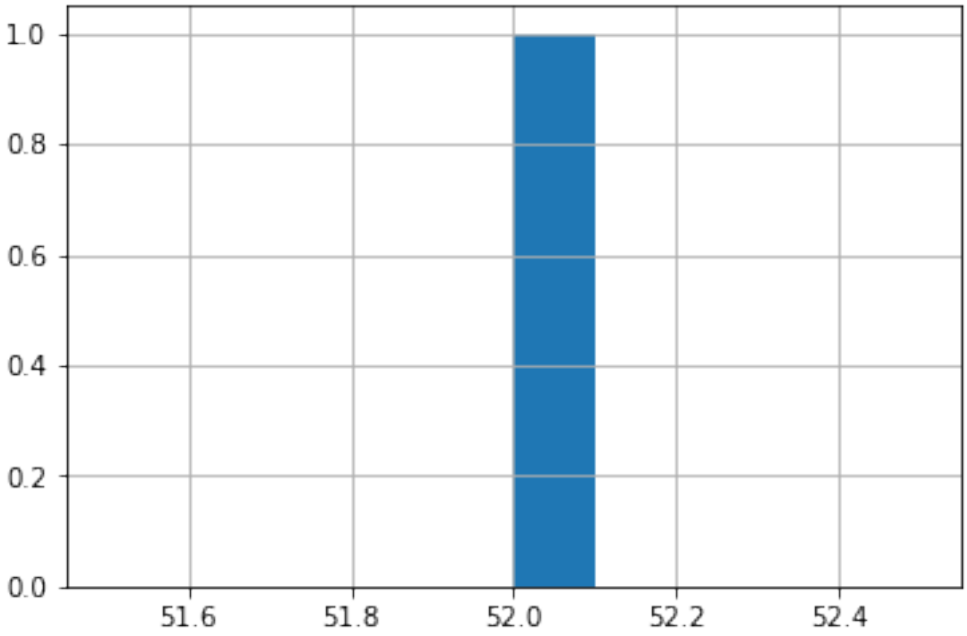
	longueur		angle		weight		precision		xPos		yPos		layer	
	mean	count	mean	count	mean	count	mean	count	mean	count	mean	count	mean	count
group														
1	3	52	0.01	52	242.906403	52	0.0	52	52.5	52	63	52	1	52

```
In [448]: titi.groupby('group').size().hist()

Out[448]: <matplotlib.axes._subplots.AxesSubplot at 0x127fcee48>
```



In [449]:
```
resultGroup = titi.groupby('group').size()
resultGroup[resultGroup>10].hist()
```

Out[449]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1294fee80>
```



In [450]:
```
titi.describe()
```

Out[450]:

	longueur	angle	weight	precision	xPos	yPos	group	\
count	52.0	5.200000e+01	52.000000	52.0	52.000000	52.0	52.0	
mean	3.0	9.999996e-03	242.906311	0.0	52.500000	63.0	1.0	
std	0.0	3.761635e-09	0.000092	0.0	15.154757	0.0	0.0	
min	3.0	1.000000e-02	242.906403	0.0	27.000000	63.0	1.0	
25%	3.0	1.000000e-02	242.906403	0.0	39.750000	63.0	1.0	
50%	3.0	1.000000e-02	242.906403	0.0	52.500000	63.0	1.0	
75%	3.0	1.000000e-02	242.906403	0.0	65.250000	63.0	1.0	
max	3.0	1.000000e-02	242.906403	0.0	78.000000	63.0	1.0	

	layer
count	52.0
mean	1.0
std	0.0
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	1.0

```
In [452]: titi[0:4]
```

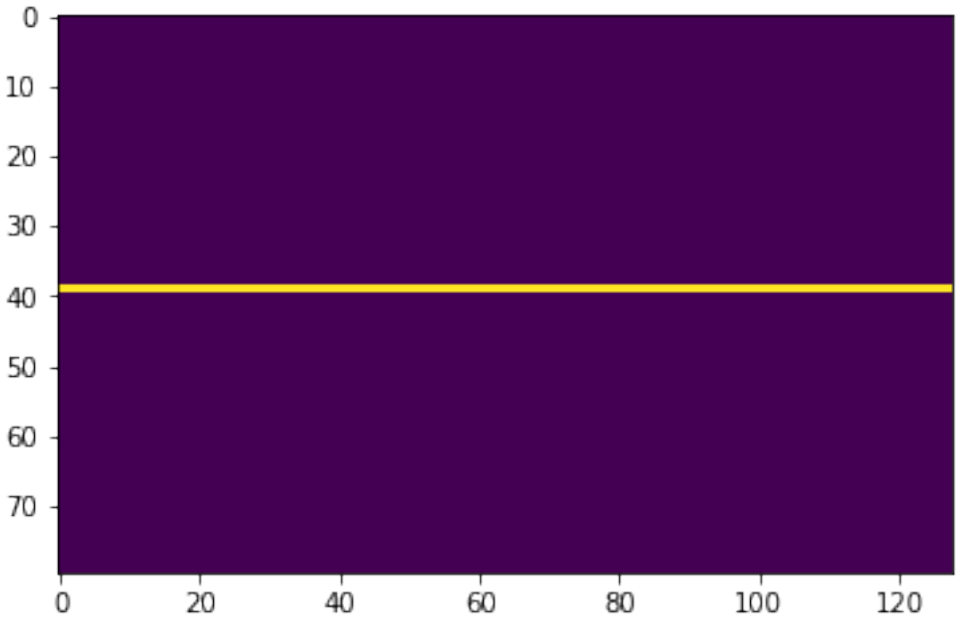
Out[452]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	3	0.01	242.906403	0.0	27	63	1	1
1	3	0.01	242.906403	0.0	28	63	1	1
2	3	0.01	242.906403	0.0	29	63	1	1
3	3	0.01	242.906403	0.0	30	63	1	1

7.2.2 Test 2

Generate data of type 2

```
In [454]: frame = generateToy(2,80,128,1)
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [455]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

exceed the limit of the matrix
error 10 :
lAngleMat
[[-4.499e+01  1.000e-02  4.501e+01]
 [ 9.000e+01  0.000e+00  9.000e+01]
```

```
 [ 4.501e+01  1.000e-02 -4.499e+01]]
lNeuronFieldFrame
[[0. 0.]
 [1. 1.]
 [0. 0.]]
lintX
39
lintY
127
offsetField
1
```

In [456]: `titi.describe()`

Out[456]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
count	128.0	128.000000	128.000000	128.0	128.000000	128.000000	128.0	128.0
mean	3.0	88.593750	239.110870	0.0	38.390625	62.507812	0.0	1.0
std	0.0	11.205622	30.243526	0.0	4.855769	37.081098	0.0	0.0
min	3.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	1.0
25%	3.0	90.000000	242.906403	0.0	39.000000	30.750000	0.0	1.0
50%	3.0	90.000000	242.906403	0.0	39.000000	62.500000	0.0	1.0
75%	3.0	90.000000	242.906403	0.0	39.000000	94.250000	0.0	1.0
max	3.0	90.000000	242.906403	0.0	39.000000	126.000000	0.0	1.0

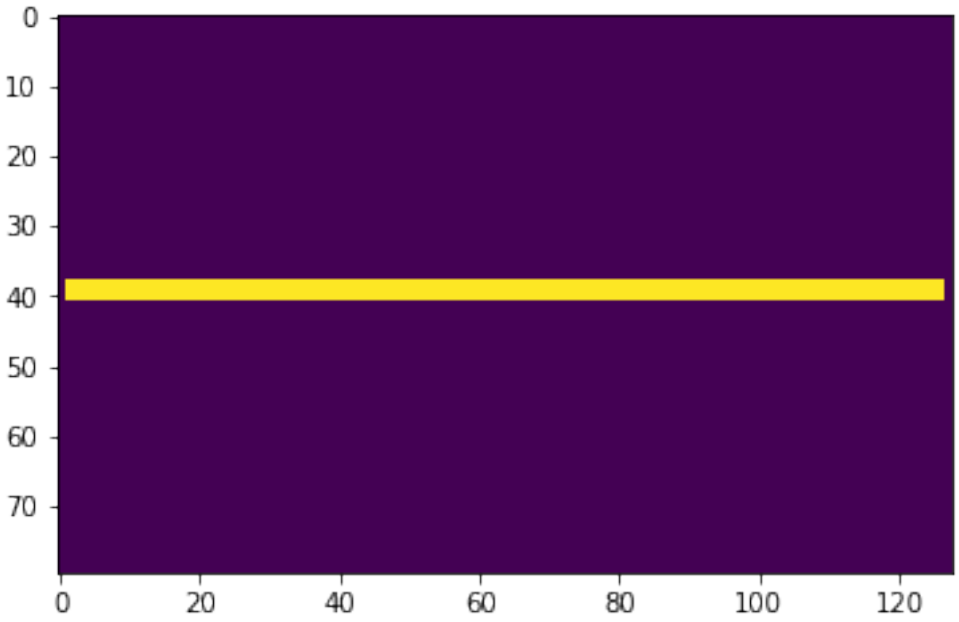
In [457]: `titi[0:4]`

Out[457]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	3	0.0	0.000000	0.0	0	0	0	1
1	3	90.0	242.906403	0.0	39	1	0	1
2	3	90.0	242.906403	0.0	39	2	0	1
3	3	90.0	242.906403	0.0	39	3	0	1

Affichage graphique du champs récepteur des neurones

In [458]:
```
testBitmap = np.zeros(frame.shape)
testBitmap = drawFieldNeurons(titi, testBitmap)
imgplot = plt.imshow(testBitmap)
```



Génération des groupes

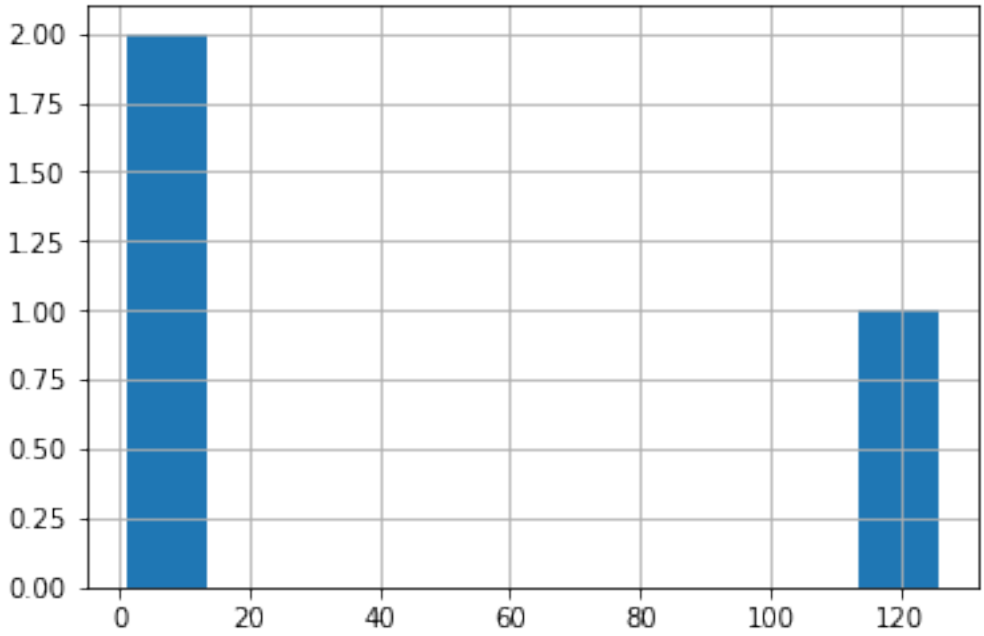
```
In [459]: findGroups(titi);

In [460]: titi.groupby('group').agg(['mean', 'count'])[0:5]
```

Out[460]:

	longueur		angle		weight		precision		xPos		\
	mean	count	mean	count	mean	count	mean	count	mean	count	
group											
1	3	1	0.0	1	0.000000	1	0.0	1	0	1	
2	3	126	90.0	126	242.906403	126	0.0	126	39	126	
3	3	1	0.0	1	0.000000	1	0.0	1	0	1	

	yPos		layer	
	mean	count	mean	count
group				
1	0.0	1	1	1
2	63.5	126	1	126
3	0.0	1	1	1

```
In [461]: titi.groupby('group').size().hist()

Out[461]: <matplotlib.axes._subplots.AxesSubplot at 0x128811080>
```



In [462]: titi[0:4]

Out[462]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	3	0.0	0.000000	0.0	0	0	1	1
1	3	90.0	242.906403	0.0	39	1	2	1
2	3	90.0	242.906403	0.0	39	2	2	1
3	3	90.0	242.906403	0.0	39	3	2	1

8 Errors list

8.1 Error 10

Problème dans la fonction Section **??**

In [ ]: