Video Object Detection

July 21, 2019

1 Preamble

L'objectif de ce document est de présenter un algorithme de détermination de l'angle d'une remorque à l'arrière d'un véhicule équipé d'une caméra vidéo.

2 import

Le programme écrit en python s'appuie sur plusieurs bibliothèques existantes standard.

```
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
```

3 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'
```

4 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()
```

5 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')])
```

5.1 Type de données de Neurones

5.1.1 Neurone sensoriel à champs récepteur

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

5.2 Taille des champs récepteurs neuronaux

```
In [328]: tailleField = 7
```

6 Fonctions

6.1 Calcul d'un neurone champ moyen

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

```
In [10]: 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
```

6.2 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
```

6.3 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
```

6.4 Création d'une liste de neurones à champs récepteurs

```
In [256]: #@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 lVerbose:
                print("lNeuronFieldFrame :")
                print(lNeuronFieldFrame)
                print("np.multiply(lAngleMat, lNeuronFieldFrame)")
                print(tmp)
                print("lNeuronFieldValues")
                print(lNeuronFieldValues)
            if (np.mean(lNeuronFieldValues))<0:
                lNeuronFieldValues[lNeuronFieldValues>89]=-90
            elif np.std(lNeuronFieldValues)>45:
                lNeuronFieldValues[lNeuronFieldValues>89]=-90
            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
```

6.5 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])
```

6.6 Coordonnées de la fonction de base (ligne)

```
In [101]: def getNFCoordinate(lNeurone,lVerbose=False):
              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
                  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
              else:
                  lAlpha = (90 - lNeurone.angle) / 180 * pi
                  if lVerbose:
                      print("Angle : "+str(lNeurone.angle))
                      print("yPos = "+str(lNeurone.yPos)+"xPos = "+str(lNeurone.xPos))
                  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
              lP1 = (int(lintY1), int(lintX1))
```

```
            lP2 = (int(lintY2), int(lintX2))
            if lVerbose:
                print("point 1: "+str(lP1))
                print("point 2: "+str(lP2))
                print("")
            return lP1, lP2
```

6.7 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))
```

6.8 Retourne les neurones les plus proches d'un point

```
In [18]: def closestFieldNeurons(lneuronList, lposX, lposY, ldistance):
             return lneuronList[(lneuronList.xPos >= lposX - ldistance)
                                & (lneuronList.xPos <= lposX + ldistance) &
                                (lneuronList.yPos >= lposY - ldistance) &
                                (lneuronList.yPos <= lposY + ldistance)]
```

6.9 Crée un neurone avec les paramètres passés

```
In [19]: def createNeuron(llong, langle, lXpos, lYpos, lweight=255, lprecis=0, lGroup=0, llaye
             lNeurons = np.zeros(1, dtype=lNType);
             lpNeurons = pd.DataFrame(lNeurons);
             lpNeurons['longueur'] = llong;
             lpNeurons['angle'] = langle;
             lpNeurons['weight'] = lweight;
             lpNeurons['precision'] = lprecis;
             lpNeurons['xPos'] = lXpos;
             lpNeurons['yPos'] = lYpos;
             lpNeurons['group'] = lGroup;
             lpNeurons['layer'] = llayer;
             return lpNeurons;
```

6.10 Dessine les fonctions de base des neurones sur un bitmap

```
In [215]: def drawFieldNeurons(lNeuronList, lBitmap, lVerbose=False, lGroupMember=0):
              lInitShow = 8
              if lVerbose:
                  lInitShow = 0
              lIndexPassOver = lInitShow
              for index, lNeuron in lNeuronList.iterrows():
                  if lGroupMember > 0:
                      if lNeuron.group != lGroupMember:
                          continue
                  lCoord = getNFCoordinate(lNeuron, lVerbose)
```

```
            if lVerbose:
                print(lNeuron)
                print(lCoord)
            if lIndexPassOver > 7:
                lIndexPassOver = lInitShow
                try:
                    cv2.line(lBitmap, lCoord[0], lCoord[1],
                     #        (255, 255, 255), 1)
                    (int(lNeuron.weight), int(lNeuron.weight), int(lNeuron.weight)), 1)
                except:
                    True
            if lVerbose:
                lIndexPassOver += 1
        return lBitmap
```

6.11 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 [21]: #@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 [22]: def getNextPosition(lneuroneMoyen, lVerbose):
             return moveCoordDeg(float(lneuroneMoyen.angle), int(lneuroneMoyen.xPos),
                                 int(lneuroneMoyen.yPos), int(lneuroneMoyen.longueur), lVerbose
```

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

```
In [23]: 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 groupID
                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 groupID
                    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
```

6.12 Get the Weighted Average of the group Angle (WAGA)

```
In [424]: def getWAGA(lNeuronList, lGroupID):
              lnl = lNeuronList[lNeuronList.group==lGroupID]
              return float(np.sum((lnl.angle * lnl.weight) /
                          np.sum(lnl.weight)))
```

6.13 Get two Neuron line spread

```
In [ ]: def getTNLSpread(lNeuron1, lNeuron2):
            return 0;
```

7 Video Loop

```
In [24]: 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, Cannyframe = cap.read()
```

```
        if ret == True:
            # Our operations on the frame come here
            if lCounter == 1:
                Cannyframe = cv2.GaussianBlur(Cannyframe, (kernelSize, kernelSize), 0, 0)
                Cannyframe = cv2.Canny(Cannyframe, parameter1, parameter2,
                                       intApertureSize)  # Canny edge detection
                lCounter = 0
                break
            lCounter += 1

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

8 Sandbox

8.1 Toy data Generator

```
In [339]: 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)
    elif lType == 5:
        lFrame = np.zeros((lHauteur, lLargeur))
        #createNeuron(llong, langle, lXpos, lYpos,
        llN = int(lLargeur / 256 * 50)
        llXTmp = int(lHauteur / 160 * 100)
        lNeuronTest2 = createNeuron(llN, -75, llXTmp, int(lLargeur / 256 * 25))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, -60, llXTmp, int(lLargeur / 256 * 75))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 125))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 60, llXTmp, int(lLargeur / 256 * 175))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 75, llXTmp, int(lLargeur / 256 * 225))
        drawFieldNeurons(lNeuronTest2, lFrame)
    elif lType == 6:
        lFrame = np.zeros((lHauteur, lLargeur))
        #createNeuron(llong, langle, lXpos, lYpos,
        llN = int(lLargeur / 256 * 50)
        llXTmp = int(lHauteur / 160 * 100)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 25))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 75))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 125))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 175))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp, int(lLargeur / 256 * 225))
        drawFieldNeurons(lNeuronTest2, lFrame)
    elif lType == 7:
        lFrame = np.zeros((lHauteur, lLargeur))
        #createNeuron(llong, langle, lXpos, lYpos,
        llN = int(lLargeur / 256 * 50)
        llXTmp = int(lHauteur / 160 * 100)
        lNeuronTest2 = createNeuron(llN, 5, llXTmp, int(lLargeur / 256 * 25))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 5, llXTmp, int(lLargeur / 256 * 75))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp+2, int(lLargeur / 256 * 125))
        drawFieldNeurons(lNeuronTest2, lFrame)
        lNeuronTest2 = createNeuron(llN, 0, llXTmp+2, int(lLargeur / 256 * 175))
```

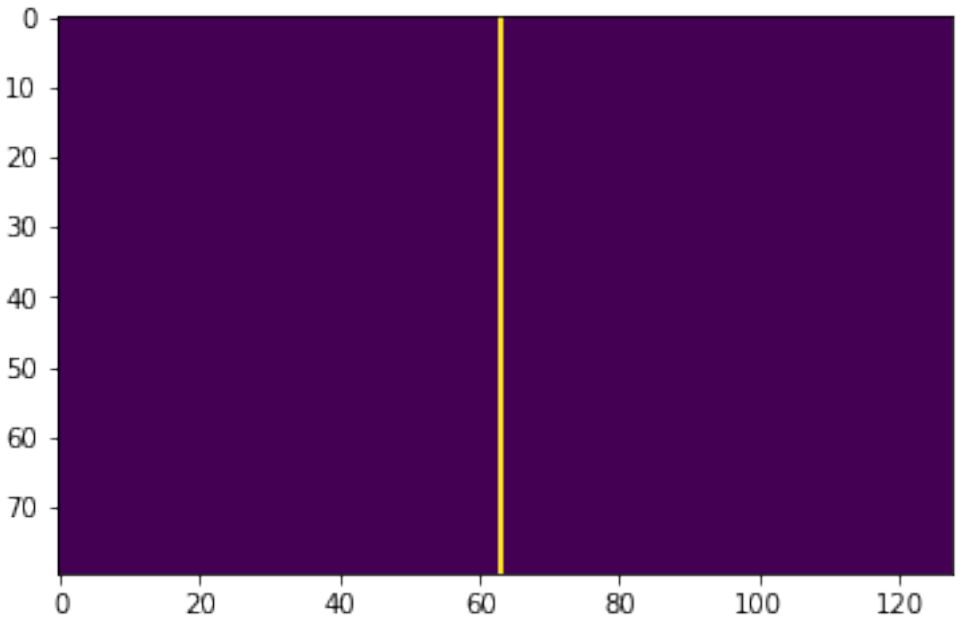
```
            drawFieldNeurons(lNeuronTest2, lFrame)
            lNeuronTest2 = createNeuron(llN, 0, llXTmp+2, int(lLargeur / 256 * 225))
            drawFieldNeurons(lNeuronTest2, lFrame)
        else:
            lFrame = np.zeros((lHauteur, lLargeur))
            print("First parameter should be between 1 to 7")
        return lFrame
```

8.2 Playground

8.2.1 Test 1

Generate data of type 1

```
In [26]: frame = generateToy(1,80,128,1)
         imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [27]: indices = np.where(frame != [0])
         nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
         titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                        nbPixelsAll)
```

```
In [28]: titi.describe()
```

Out[28]:
```
        longueur         angle      weight  precision      xPos  yPos  group  \
count       50.0  5.000000e+01   50.000000       50.0  50.00000  50.0   50.0
mean         7.0  9.999996e-03  242.906311        0.0  51.50000  63.0    0.0
std          0.0  3.763112e-09    0.000092        0.0  14.57738   0.0    0.0
min          7.0  1.000000e-02  242.906403        0.0  27.00000  63.0    0.0
25%          7.0  1.000000e-02  242.906403        0.0  39.25000  63.0    0.0
50%          7.0  1.000000e-02  242.906403        0.0  51.50000  63.0    0.0
75%          7.0  1.000000e-02  242.906403        0.0  63.75000  63.0    0.0
max          7.0  1.000000e-02  242.906403        0.0  76.00000  63.0    0.0

        layer
count    50.0
mean      1.0
std       0.0
min       1.0
25%       1.0
50%       1.0
75%       1.0
max       1.0
```

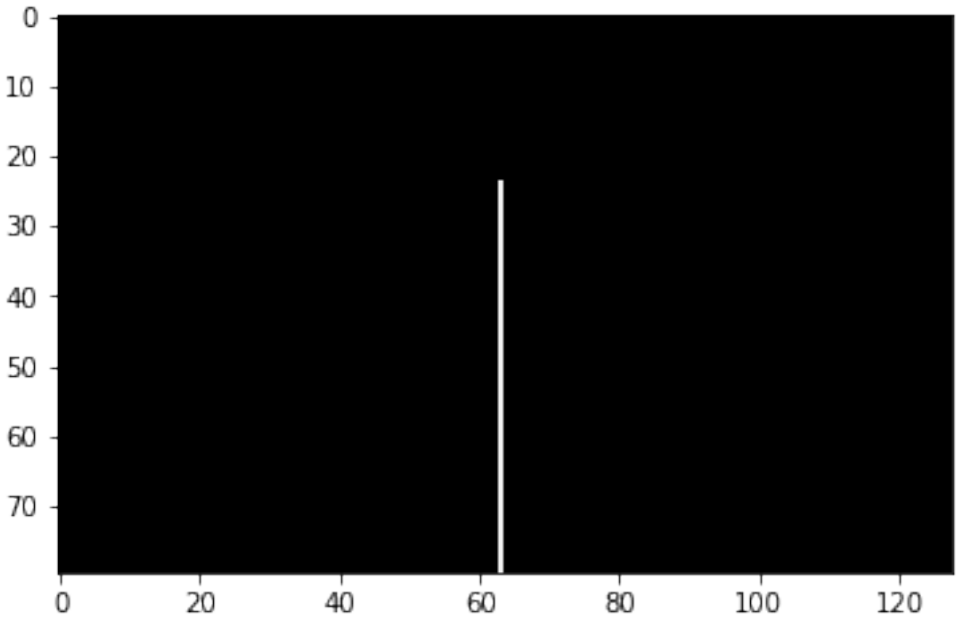
In [29]:
```
titi[0:4]
```

Out[29]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	7	0.01	242.906403	0.0	27	63	0	1
1	7	0.01	242.906403	0.0	28	63	0	1
2	7	0.01	242.906403	0.0	29	63	0	1
3	7	0.01	242.906403	0.0	30	63	0	1

Affichage graphique du champs récepteur des neurones

In [30]:
```
testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
testBitmap = drawFieldNeurons(titi, testBitmap)
imgplot = plt.imshow(testBitmap)
```




```
In [31]: np.max(testBitmap)
```

```
Out[31]: 242
```

```
In [32]: 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

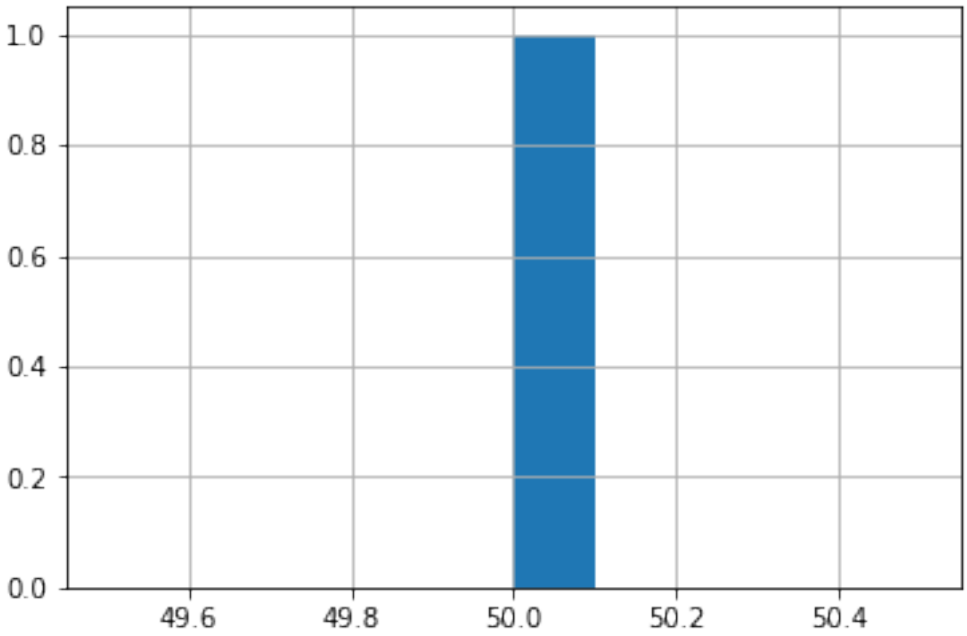
```
In [33]: findGroups(titi);
```

```
In [34]: titi.groupby('group').agg(['mean', 'count'])[0:5]
```

```
Out[34]:       longueur       angle            weight       precision        xPos  \
                   mean count  mean count        mean count      mean count  mean
         group
         1            7    50  0.01    50  242.906403    50       0.0    50  51.5

                    yPos       layer
               count mean count  mean count
         group
         1        50   63    50     1    50
```

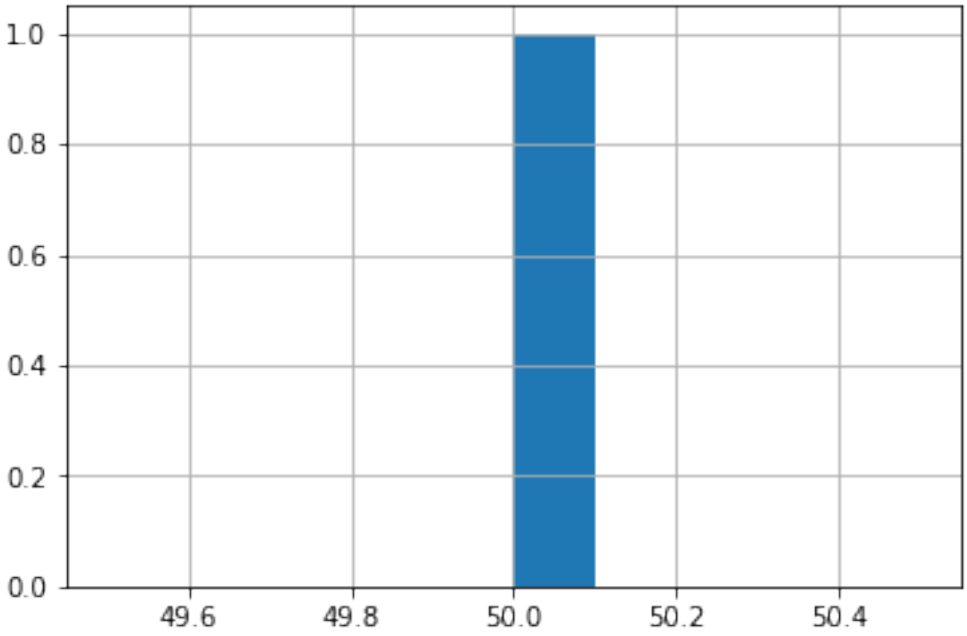
```
In [35]: titi.groupby('group').size().hist()
```

```
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x12e2c5ba8>
```




```
In [36]: resultGroup = titi.groupby('group').size()
         resultGroup[resultGroup>10].hist()
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x12de5aa90>
```



```
In [37]: titi.describe()
```

```
Out[37]:         longueur         angle      weight  precision      xPos  yPos  group  \
         count      50.0  5.000000e+01   50.000000       50.0  50.00000  50.0   50.0
         mean        7.0  9.999996e-03  242.906311        0.0  51.50000  63.0    1.0
         std         0.0  3.763112e-09    0.000092        0.0  14.57738   0.0    0.0
         min         7.0  1.000000e-02  242.906403        0.0  27.00000  63.0    1.0
         25%         7.0  1.000000e-02  242.906403        0.0  39.25000  63.0    1.0
         50%         7.0  1.000000e-02  242.906403        0.0  51.50000  63.0    1.0
         75%         7.0  1.000000e-02  242.906403        0.0  63.75000  63.0    1.0
         max         7.0  1.000000e-02  242.906403        0.0  76.00000  63.0    1.0

                layer
         count   50.0
         mean     1.0
         std      0.0
         min      1.0
         25%      1.0
         50%      1.0
         75%      1.0
         max      1.0
```

```
In [38]: titi[0:4]
```

```
Out[38]:    longueur  angle      weight  precision  xPos  yPos  group  layer
         0         7   0.01  242.906403        0.0    27    63      1      1
         1         7   0.01  242.906403        0.0    28    63      1      1
         2         7   0.01  242.906403        0.0    29    63      1      1
         3         7   0.01  242.906403        0.0    30    63      1      1
```

Simplify

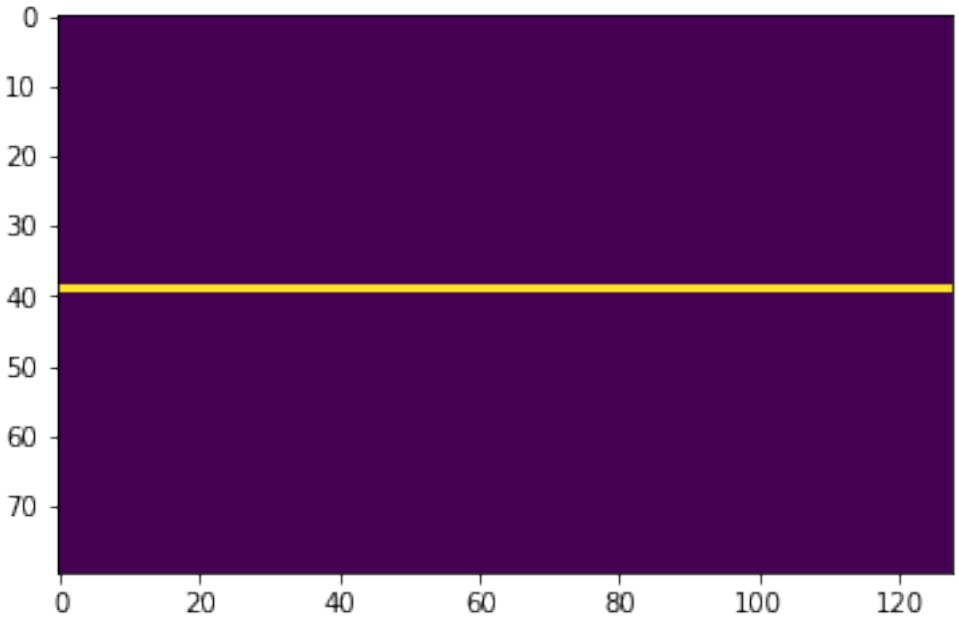
```
In [455]: np.sum(titi.memory_usage())
```

```
Out[455]: 5020
```

8.2.2 Test 2

Generate data of type 2

```
In [39]: frame = generateToy(2,80,128,1)
         imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [40]: 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
exceed the limit of the matrix
exceed the limit of the matrix
error 10 :
lAngleMat
[[-4.499e+01 -3.368e+01 -1.842e+01  1.000e-02  1.844e+01  3.370e+01
   4.501e+01]
 [-5.630e+01 -4.499e+01 -2.656e+01  1.000e-02  2.658e+01  4.501e+01
   5.632e+01]
 [-7.156e+01 -6.342e+01 -4.499e+01  1.000e-02  4.501e+01  6.344e+01
   7.158e+01]
 [ 9.000e+01  9.000e+01  9.000e+01  0.000e+00  9.000e+01  9.000e+01
   9.000e+01]
 [ 7.158e+01  6.344e+01  4.501e+01  1.000e-02 -4.499e+01 -6.342e+01
  -7.156e+01]
 [ 5.632e+01  4.501e+01  2.658e+01  1.000e-02 -2.656e+01 -4.499e+01
  -5.630e+01]
 [ 4.501e+01  3.370e+01  1.844e+01  1.000e-02 -1.842e+01 -3.368e+01
  -4.499e+01]]
```

```
lNeuronFieldFrame
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
lintX
39
lintY
125
offsetField
3
error 10 :
lAngleMat
[[-4.499e+01 -3.368e+01 -1.842e+01  1.000e-02  1.844e+01  3.370e+01
   4.501e+01]
 [-5.630e+01 -4.499e+01 -2.656e+01  1.000e-02  2.658e+01  4.501e+01
   5.632e+01]
 [-7.156e+01 -6.342e+01 -4.499e+01  1.000e-02  4.501e+01  6.344e+01
   7.158e+01]
 [ 9.000e+01  9.000e+01  9.000e+01  0.000e+00  9.000e+01  9.000e+01
   9.000e+01]
 [ 7.158e+01  6.344e+01  4.501e+01  1.000e-02 -4.499e+01 -6.342e+01
  -7.156e+01]
 [ 5.632e+01  4.501e+01  2.658e+01  1.000e-02 -2.656e+01 -4.499e+01
  -5.630e+01]
 [ 4.501e+01  3.370e+01  1.844e+01  1.000e-02 -1.842e+01 -3.368e+01
  -4.499e+01]]
lNeuronFieldFrame
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
lintX
39
lintY
125
offsetField
3
error 10 :
lAngleMat
[[-4.499e+01 -3.368e+01 -1.842e+01  1.000e-02  1.844e+01  3.370e+01
   4.501e+01]
```

```
 [-5.630e+01 -4.499e+01 -2.656e+01  1.000e-02  2.658e+01  4.501e+01
   5.632e+01]
 [-7.156e+01 -6.342e+01 -4.499e+01  1.000e-02  4.501e+01  6.344e+01
   7.158e+01]
 [ 9.000e+01  9.000e+01  9.000e+01  0.000e+00  9.000e+01  9.000e+01
   9.000e+01]
 [ 7.158e+01  6.344e+01  4.501e+01  1.000e-02 -4.499e+01 -6.342e+01
  -7.156e+01]
 [ 5.632e+01  4.501e+01  2.658e+01  1.000e-02 -2.656e+01 -4.499e+01
  -5.630e+01]
 [ 4.501e+01  3.370e+01  1.844e+01  1.000e-02 -1.842e+01 -3.368e+01
  -4.499e+01]]
lNeuronFieldFrame
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
lintX
39
lintY
125
offsetField
3
```

In [41]: `titi.describe()`

Out[41]:

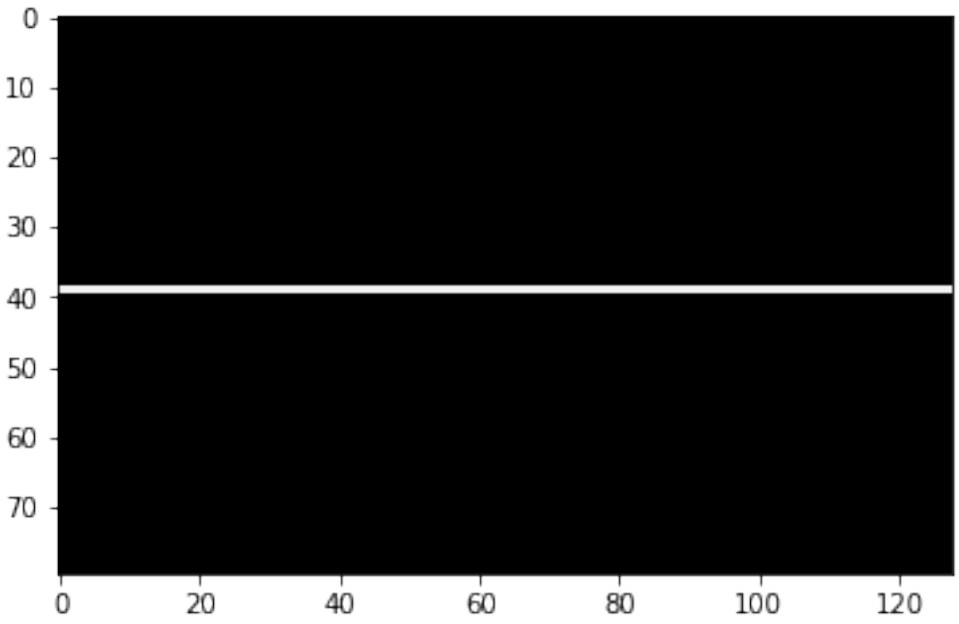
	longueur	angle	weight	precision	xPos	yPos	group	layer
count	128.0	128.000000	128.000000	128.0	128.000000	128.000000	128.0	128.0
mean	7.0	85.781250	231.520050	0.0	37.171875	60.523438	0.0	1.0
std	0.0	19.098122	51.545200	0.0	8.275863	37.053901	0.0	0.0
min	7.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	1.0
25%	7.0	90.000000	242.906403	0.0	39.000000	28.750000	0.0	1.0
50%	7.0	90.000000	242.906403	0.0	39.000000	60.500000	0.0	1.0
75%	7.0	90.000000	242.906403	0.0	39.000000	92.250000	0.0	1.0
max	7.0	90.000000	242.906403	0.0	39.000000	124.000000	0.0	1.0

```
In [42]: titi[0:4]
```

```
Out[42]:    longueur  angle      weight  precision  xPos  yPos  group  layer
         0         7    0.0    0.000000        0.0     0     0      0      1
         1         7    0.0    0.000000        0.0     0     0      0      1
         2         7    0.0    0.000000        0.0     0     0      0      1
         3         7   90.0  242.906403        0.0    39     3      0      1
```

Affichage graphique du champs récepteur des neurones

```
In [43]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
         testBitmap = drawFieldNeurons(titi, testBitmap)
         imgplot = plt.imshow(testBitmap)
```



Génération des groupes

```
In [44]: findGroups(titi);
```

```
In [45]: titi.groupby('group').agg(['mean', 'count'])[0:5]
```

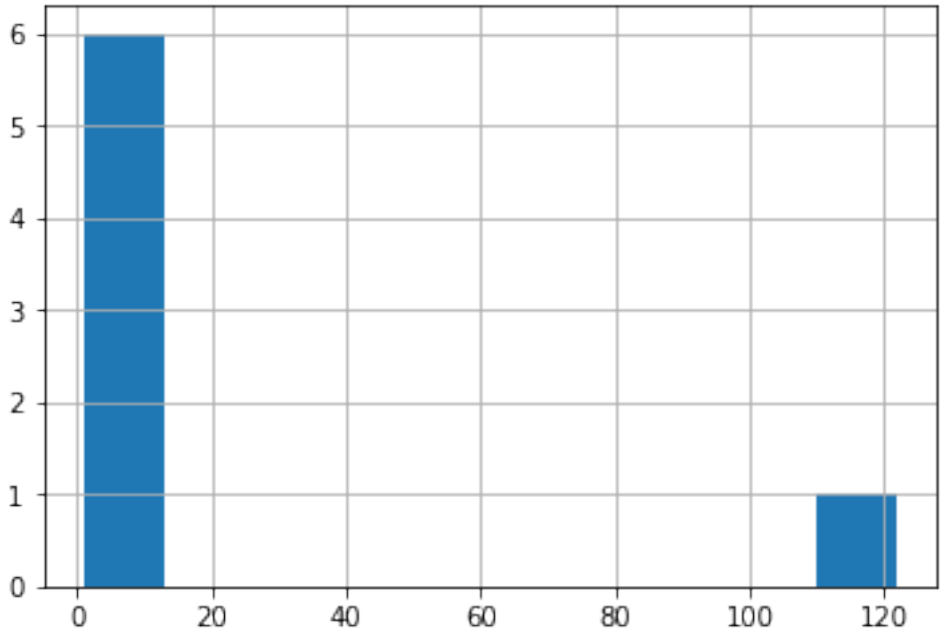
```
Out[45]:       longueur       angle            weight       precision       xPos        \
                   mean count  mean count        mean count      mean count mean count
         group
         1            7     1   0.0     1    0.000000     1       0.0     1    0      1
         2            7     1   0.0     1    0.000000     1       0.0     1    0      1
         3            7     1   0.0     1    0.000000     1       0.0     1    0      1
         4            7   122  90.0   122  242.906403   122       0.0   122   39    122
```

```
5            7     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       0.0     1     1     1
3       0.0     1     1     1
4      63.5   122     1   122
5       0.0     1     1     1
```

```
In [46]: titi.groupby('group').size().hist()
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x12dd8cb38>
```



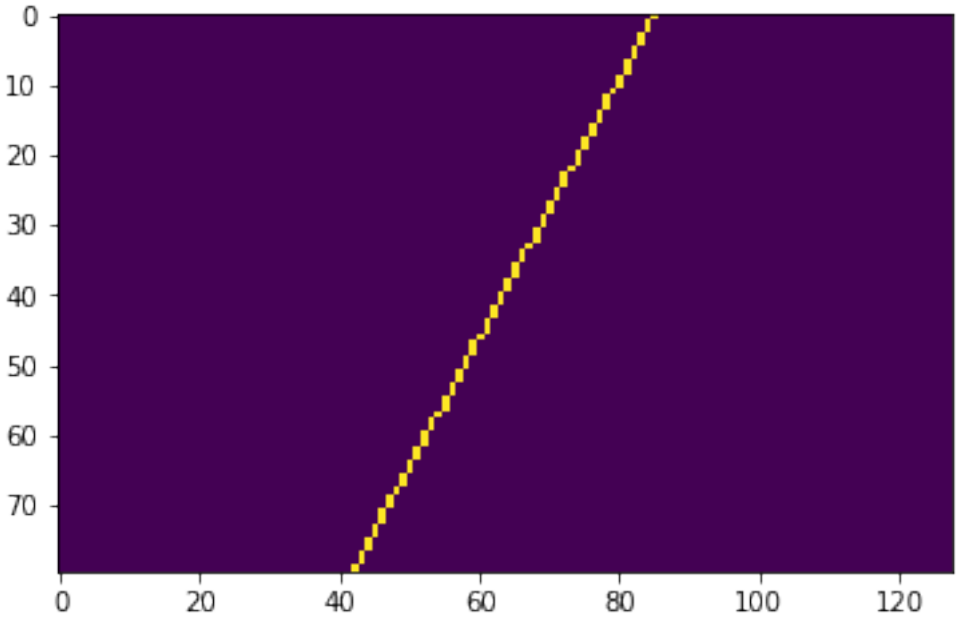

```
In [47]: titi[0:4]
```

```
Out[47]:    longueur  angle      weight  precision  xPos  yPos  group  layer
0         7    0.0    0.000000        0.0     0     0      1      1
1         7    0.0    0.000000        0.0     0     0      2      1
2         7    0.0    0.000000        0.0     0     0      3      1
3         7   90.0  242.906403        0.0    39     3      4      1
```

8.2.3 Test 3

Generate data of type 3

```
In [48]: frame = generateToy(3,80,128,1)
         imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [49]: indices = np.where(frame != [0])
         nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
         titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                        nbPixelsAll)

In [50]: titi.describe()
```

Out[50]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
count	50.0	50.000000	50.000000	50.000000	50.00000	50.000000	50.0	50.0
mean	7.0	26.755207	213.043991	13.634507	51.50000	56.980000	0.0	1.0
std	0.0	2.806284	5.832972	1.934354	14.57738	7.924362	0.0	0.0
min	7.0	25.053333	205.570328	7.588554	27.00000	44.000000	0.0	
25%	7.0	25.053333	212.746216	13.835940	39.25000	50.250000	0.0	
50%	7.0	25.053333	212.746216	13.835940	51.50000	57.000000	0.0	
75%	7.0	27.596666	212.746216	13.835940	63.75000	63.750000	0.0	
max	7.0	35.096668	230.490402	15.747627	76.00000	70.000000	0.0	

```
min      1.0
25%      1.0
50%      1.0
75%      1.0
max      1.0
```

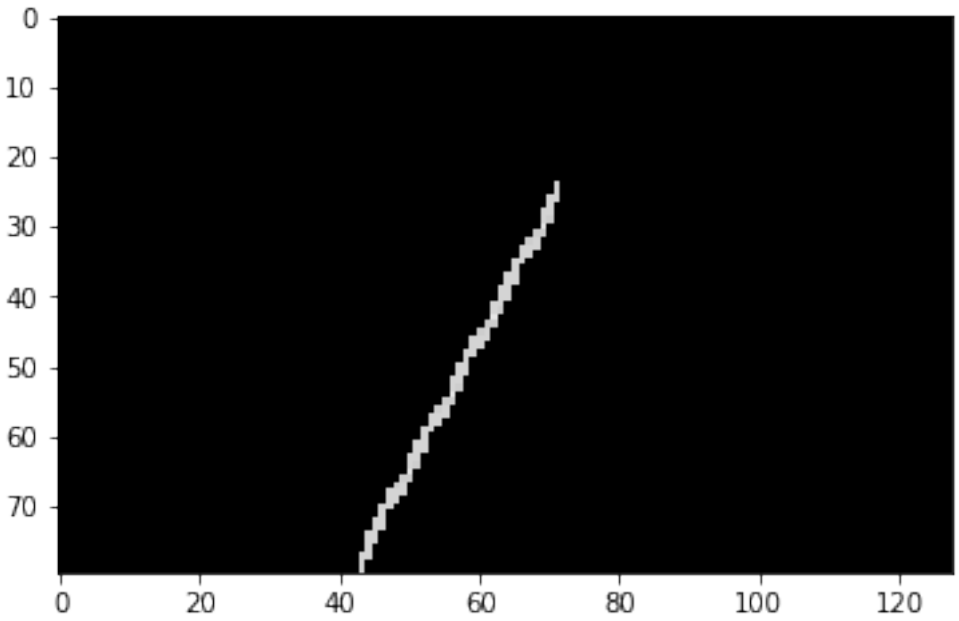
```
In [51]: titi[0:4]
```

Out[51]:

	longueur	angle	weight	precision	xPos	yPos	group	layer
0	7	25.053333	212.746216	13.83594	27	70	0	1
1	7	25.053333	212.746216	13.83594	28	70	0	1
2	7	25.053333	212.746216	13.83594	29	69	0	1
3	7	25.053333	212.746216	13.83594	30	69	0	1

Affichage graphique du champs récepteur des neurones

```
In [52]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
         testBitmap = drawFieldNeurons(titi, testBitmap)
         imgplot = plt.imshow(testBitmap)
```



Génération des groupes

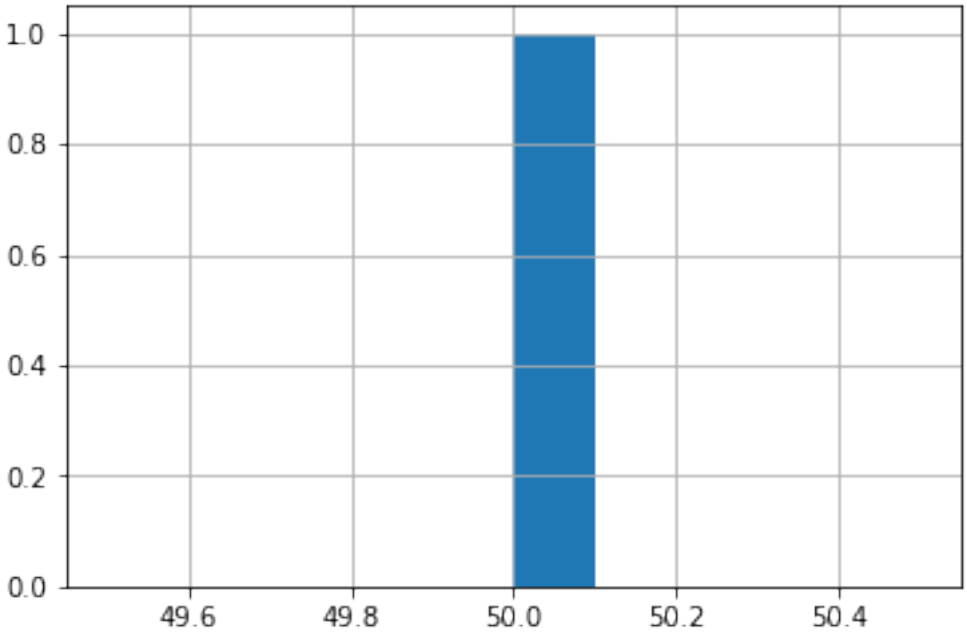
```
In [53]: findGroups(titi);
```

```
In [54]: titi.groupby('group').agg(['mean', 'count'])[0:5]
```

```
Out[54]:       longueur            angle            weight        precision        \
                   mean count       mean count        mean count       mean count
         group
         1            7    50  26.755199    50  213.044052    50  13.634507    50

                xPos          yPos       layer
                mean count    mean count  mean count
         group
         1      51.5    50   56.98    50     1    50
```

```
In [55]: titi.groupby('group').size().hist()
```

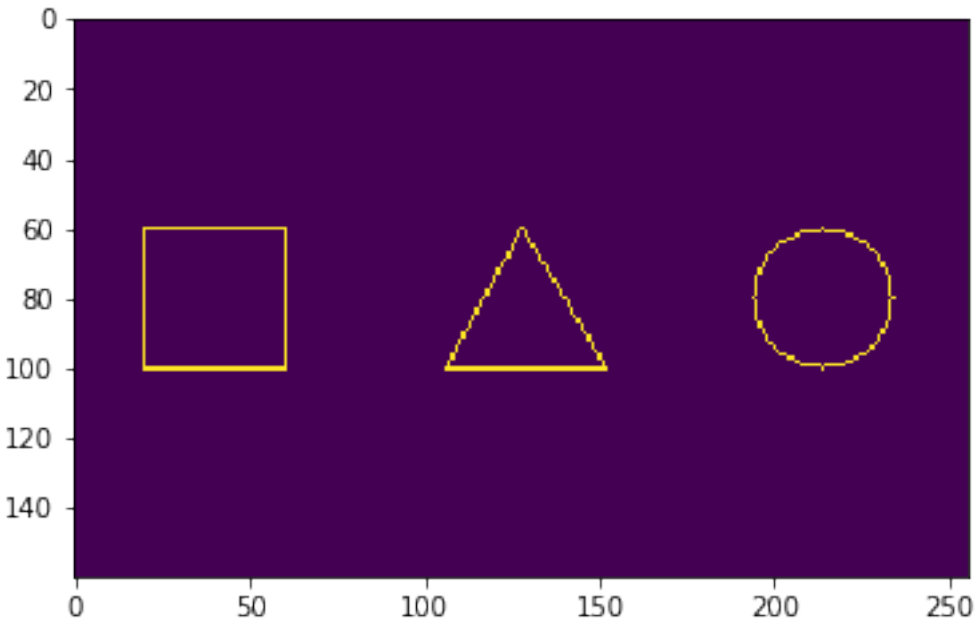
```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x12df24b38>
```



8.2.4 Test 4

Generate data of type 4

```
In [372]: frame = generateToy(4,160,256,1)
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [373]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

In [374]: titi.describe()

Out[374]:        longueur       angle      weight   precision        xPos        yPos  \
          count     398.0  398.000000  398.000000  398.000000  398.000000  398.000000
          mean        7.0   22.983759  214.483398    9.913938   82.311558  117.037688
          std         0.0   52.803646   44.123013   11.387993   15.140764   72.832620
          min         7.0  -86.926666   10.970448    0.000000   60.000000   20.000000
          25%         7.0   -7.785227  205.570328    0.000000   67.000000   50.000000
          50%         7.0   13.627500  230.490402    7.588554   84.000000  120.000000
          75%         7.0   90.000000  242.906403   15.747627   99.000000  195.000000
          max         7.0   90.000000  242.906403   61.020844  100.000000  234.000000

                 group  layer
          count  398.0  398.0
          mean     0.0    1.0
          std      0.0    0.0
          min      0.0    1.0
          25%      0.0    1.0
          50%      0.0    1.0
```

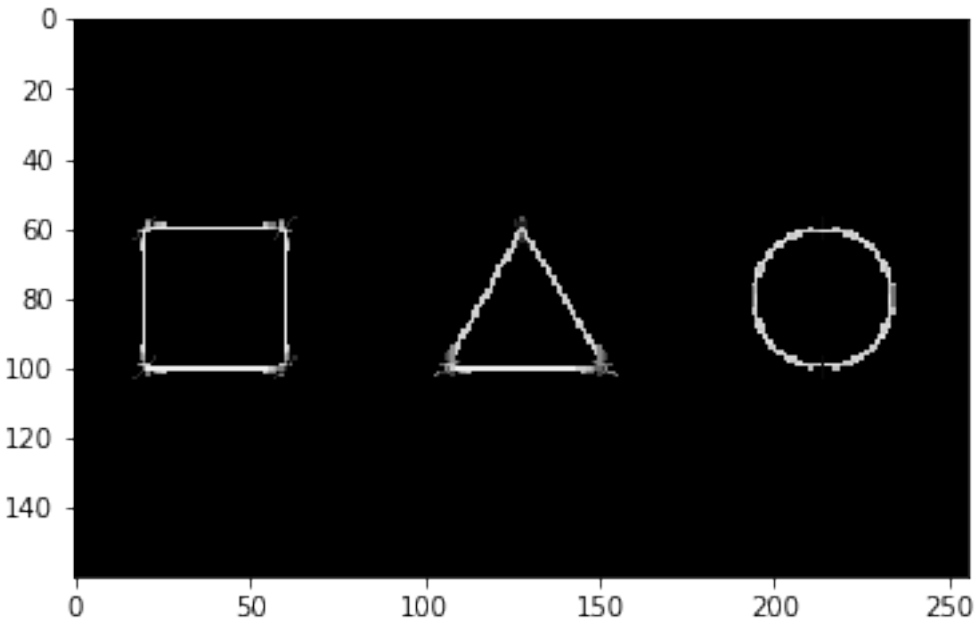
```
          75%      0.0    1.0
          max      0.0    1.0
```

```
In [375]: titi[0:4]
```

```
Out[375]:    longueur      angle      weight  precision  xPos  yPos  group  layer
          0         7  45.005001   46.537529  44.994999    60    20      0      1
          1         7  64.290001  123.892136  30.566092    60    21      0      1
          2         7  74.018753  176.191147  21.954557    60    22      0      1
          3         7  79.212219  202.501602  16.500349    60    23      0      1
```

Affichage graphique du champs récepteur des neurones

```
In [376]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap)
          imgplot = plt.imshow(testBitmap)
```

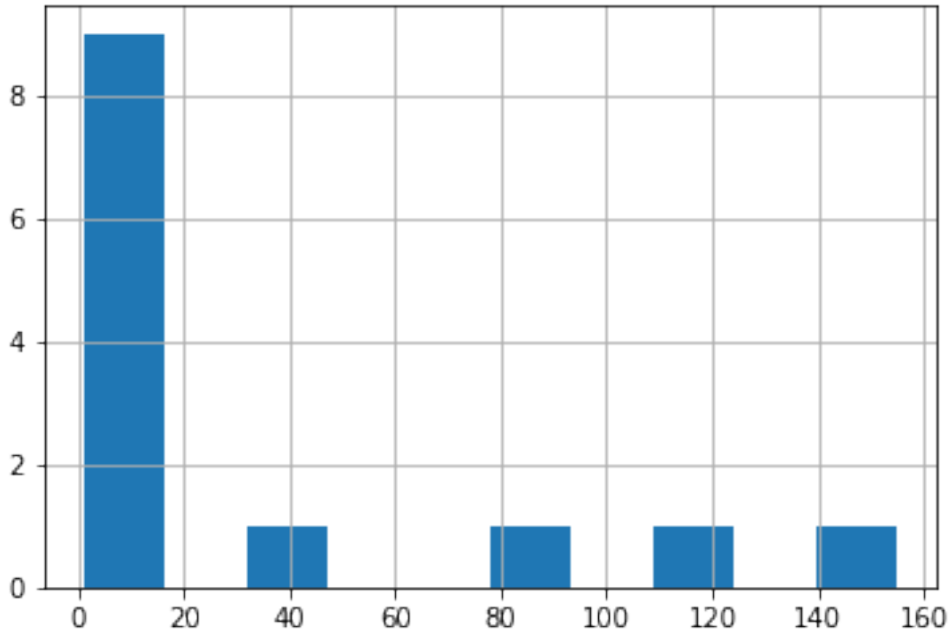


Génération des groupes

```
In [377]: findGroups(titi);
```

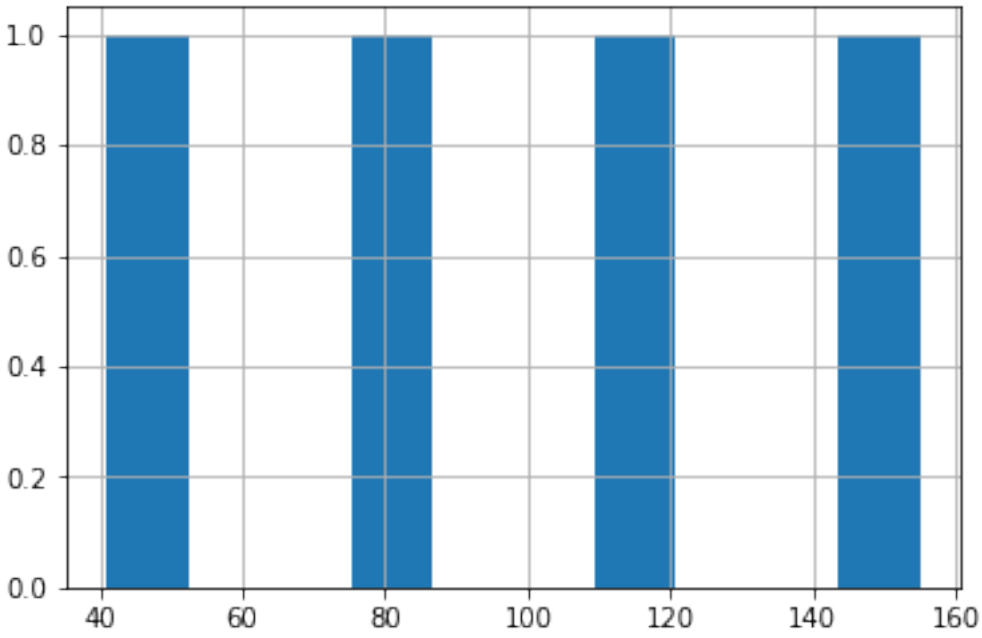
```
In [378]: titi.groupby('group').size().hist()
```

```
Out[378]: <matplotlib.axes._subplots.AxesSubplot at 0x134986a58>
```




```
In [379]: resultGroup = titi.groupby('group').size()
          resultGroup[resultGroup>10].hist()
```

```
Out[379]: <matplotlib.axes._subplots.AxesSubplot at 0x132711da0>
```



```
In [396]: titi.groupby('group').agg(['mean', 'count'])[resultGroup>10]
```

Out[396]:

group	longueur		angle		weight		precision		xPos		yPos		layer	
	mean	count	mean	count	mean	count	mean	count	mean	count	mean	count	mean	count
1	7	155	40.768021	155	229.658386	155	3.791248	155	80.638710	155	39.393548	155	1	155
6	7	83	48.416748	83	211.846710	83	10.593689	83	90.156627	83	122.337349	83	1	83
9	7	41	-21.370747	41	187.712830	41	18.333818	41	80.902439	41	140.390244	41	1	41
10	7	110	0.008000	110	214.484055	110	12.641945	110	80.000000	110	214.000000	110	1	110

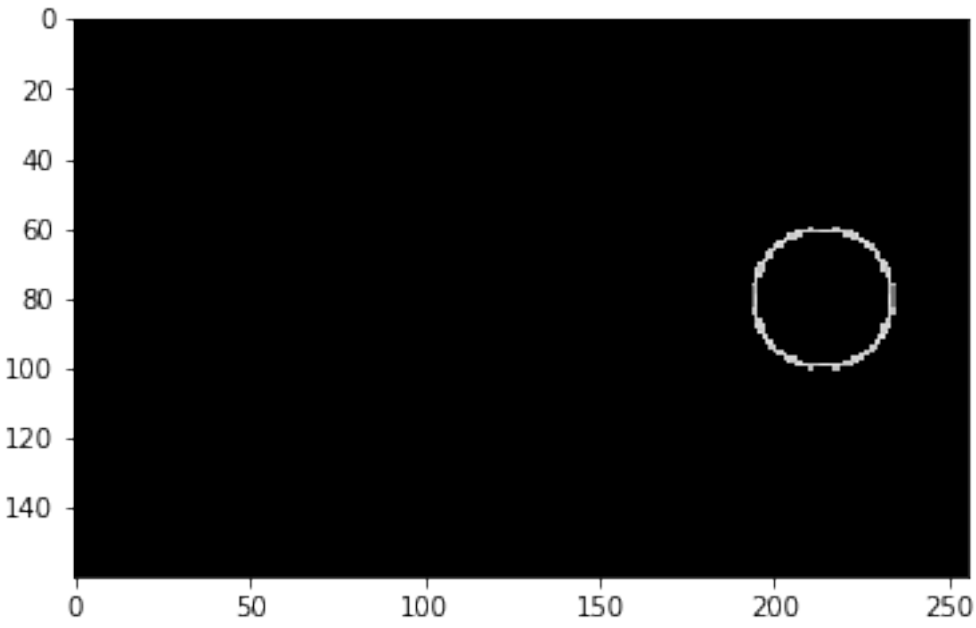
```
In [423]: for lidx, langle in titi.groupby('group').count().sort_values('angle', ascending=Fals
              print (lidx)
```

```
1
10
6
9
2
3
4
5
7
8
11
12
13
```

```
In [426]: getWAGA(titi,6)
```

Out[426]: 54.005035400390625

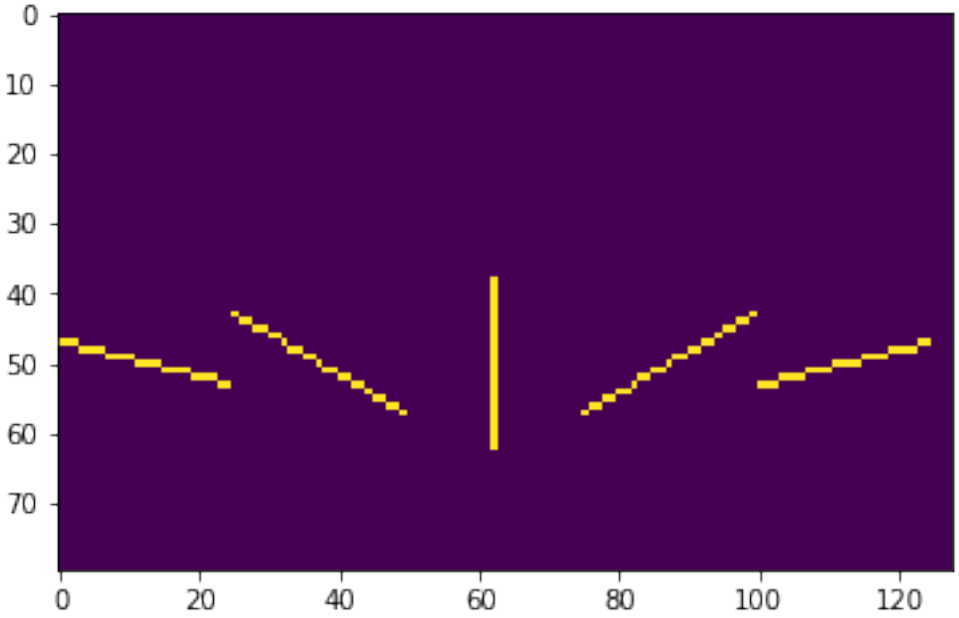
```
In [401]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap,0,10)
          imgplot = plt.imshow(testBitmap)
```



8.2.5 Test 5

Generate data of type 5

```
In [427]: frame = generateToy(5,80,128,1)
          imgplot = plt.imshow(frame)
```



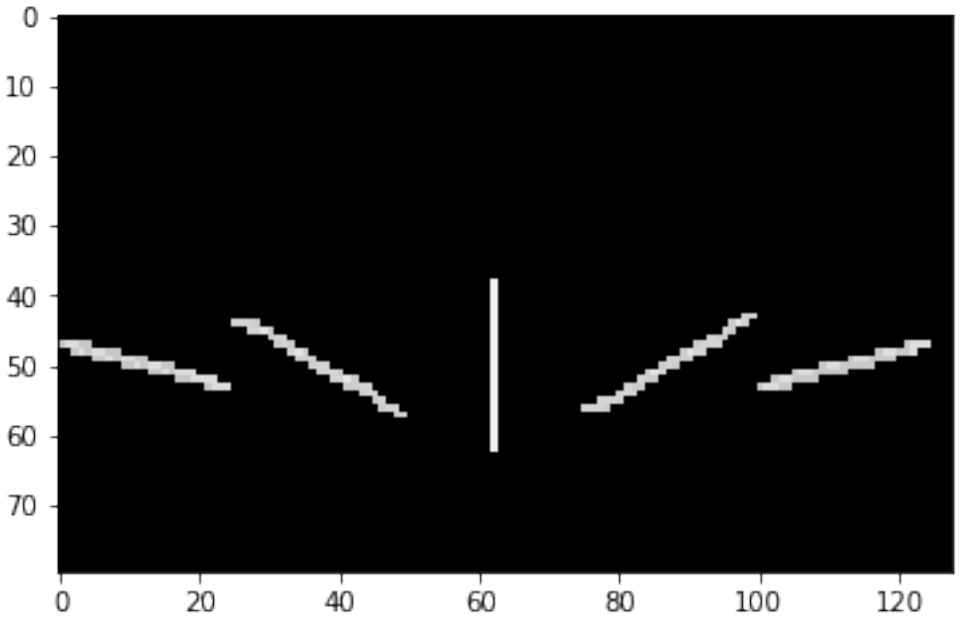
Génération des neurones à champs récepteur

```
In [428]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

In [429]: titi.describe()

Out[429]:        longueur      angle      weight  precision       xPos        yPos  \
          count      95.0  95.000000   95.000000  95.000000  95.000000   95.000000
          mean        7.0   0.007439  218.687790  10.881714  50.000000   62.000000
          std         0.0  62.906307   14.875514   6.040379   3.313416   35.882491
          min         7.0 -79.423332  200.650269   0.000000  41.000000    3.000000
          25%         7.0 -62.404999  205.570328   7.588554  48.000000   32.500000
          50%         7.0   0.010000  212.923187  13.785652  50.000000   62.000000
          75%         7.0  62.421665  230.490402  15.747627  52.000000   91.500000
          max         7.0  79.433334  242.906403  16.938763  59.000000  121.000000

                 group  layer
          count   95.0   95.0
          mean     0.0    1.0
          std      0.0    0.0
          min      0.0    1.0
          25%      0.0    1.0
          50%      0.0    1.0
          75%      0.0    1.0
          max      0.0    1.0

In [430]: titi.groupby('angle').agg(['mean', 'count'])

Out[430]:            longueur            weight        precision             xPos        \
                         mean count        mean count       mean count       mean count
          angle
          -79.423332        7    10  222.054947    10  10.919160    10  50.000000    10
          -74.995003        7     9  200.650269     9  16.938763     9  49.777778     9
          -64.948334        7     4  212.735580     4  13.838958     4  48.500000     4
          -62.404999        7     6  212.899750     6  13.792321     6  50.333333     6
          -61.876667        7     6  205.558472     6  15.750603     6  50.333333     6
          -54.903332        7     3  230.490402     3   7.588554     3  50.333333     3
           0.010000         7    19  242.906403    19   0.000000    19  50.000000    19
           54.923332        7     3  230.490402     3   7.588554     3  49.666667     3
           61.893333        7     6  205.582184     6  15.744652     6  49.666667     6
           62.421665        7     6  212.923187     6  13.785652     6  49.666667     6
           64.964996        7     4  212.756851     4  13.832924     4  51.500000     4
           75.004997        7     9  200.688141     9  16.929905     9  50.222222     9
           79.433334        7    10  222.082733    10  10.909474    10  50.000000    10

                           yPos       group       layer
```

	mean	count	mean	count	mean	count
angle						
-79.423332	12.500000	10	0	10	1	10
-74.995003	11.444444	9	0	9	1	9
-64.948334	34.500000	4	0	4	1	4
-62.404999	37.666667	6	0	6	1	6
-61.876667	37.666667	6	0	6	1	6
-54.903332	37.666667	3	0	3	1	3
0.010000	62.000000	19	0	19	1	19
54.923332	87.666667	3	0	3	1	3
61.893333	87.666667	6	0	6	1	6
62.421665	87.666667	6	0	6	1	6
64.964996	84.500000	4	0	4	1	4
75.004997	111.444444	9	0	9	1	9
79.433334	112.500000	10	0	10	1	10

Affichage graphique du champs récepteur des neurones

```
In [431]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap)
          imgplot = plt.imshow(testBitmap)
```

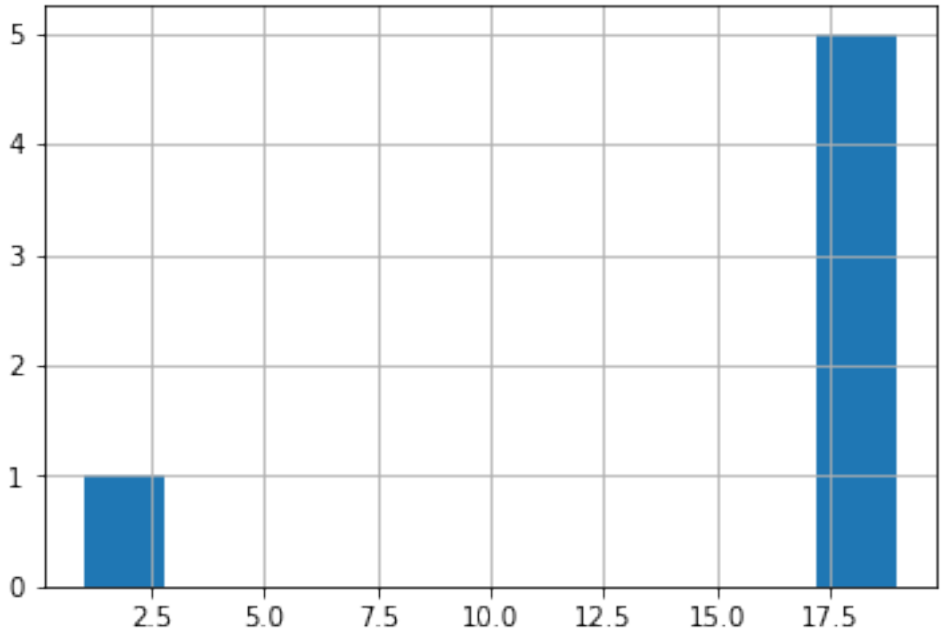


Génération des groupes

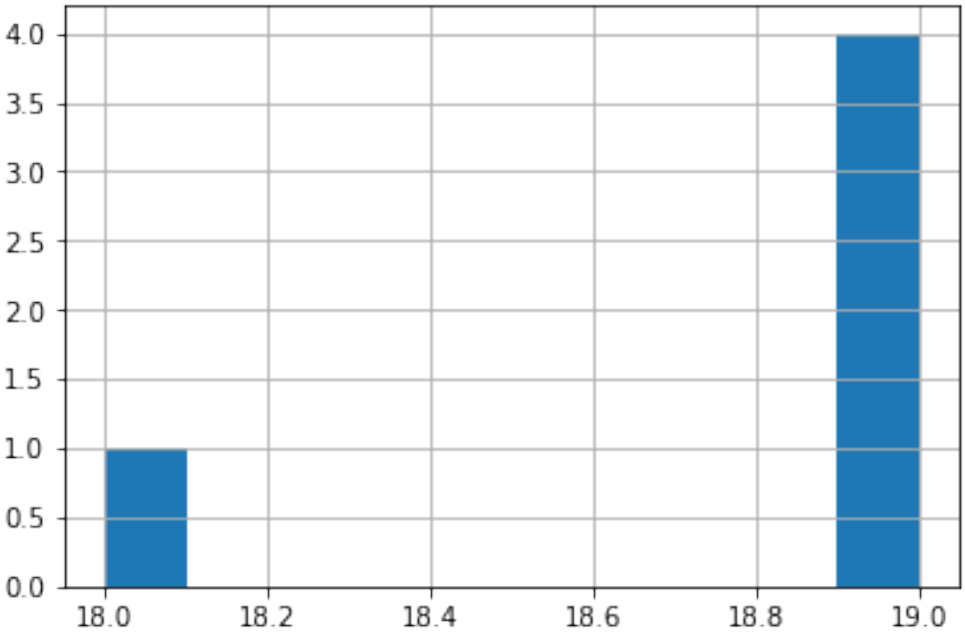
```
In [432]: findGroups(titi);
```

```
In [433]: titi.groupby('group').size().hist()
```

```
Out[433]: <matplotlib.axes._subplots.AxesSubplot at 0x12e882ba8>
```




```
In [434]: resultGroup = titi.groupby('group').size()
          resultGroup[resultGroup>10].hist()

Out[434]: <matplotlib.axes._subplots.AxesSubplot at 0x13546e048>
```



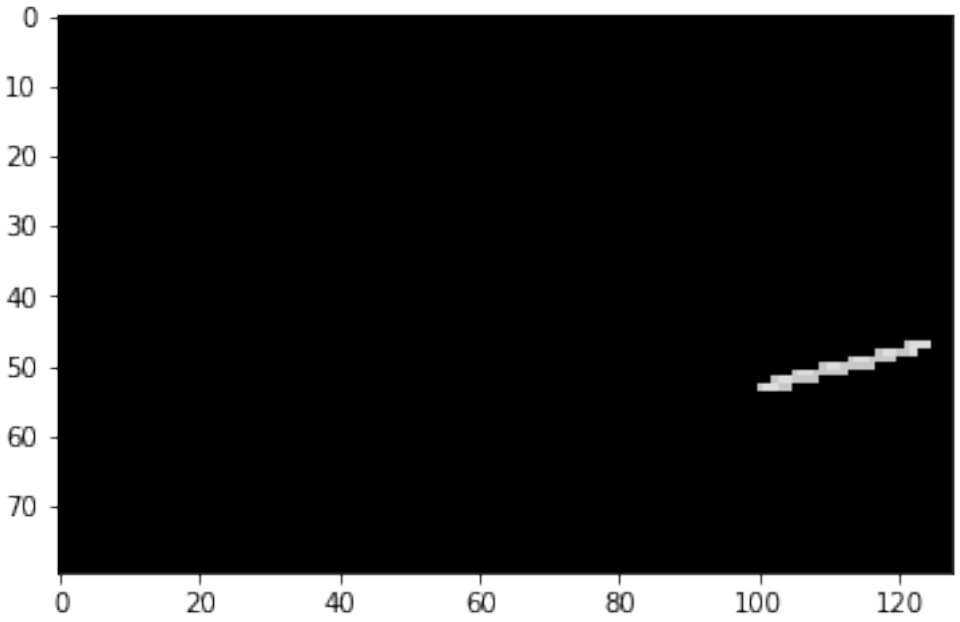
```
In [435]: titi.groupby('group').agg(['mean', 'count'])[resultGroup>10]
```

Out[435]:

	longueur		angle		weight		precision		\
	mean	count	mean	count	mean	count	mean	count	
group									
1	7	19	0.010000	19	242.906403	19	0.000000	19	
2	7	19	-61.589123	19	213.324356	19	13.441002	19	
3	7	18	61.419720	18	213.376328	18	13.413681	18	
4	7	19	-77.325699	19	211.915894	19	13.770551	19	
5	7	19	77.335701	19	211.948456	19	13.761257	19	

	xPos		yPos		layer	
	mean	count	mean	count	mean	count
group						
1	50.000000	19	62.000000	19	1	19
2	49.947368	19	37.000000	19	1	19
3	49.777778	18	87.444444	18	1	18
4	49.894737	19	12.000000	19	1	19
5	50.105263	19	112.000000	19	1	19

```
In [436]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap,0,5)
          imgplot = plt.imshow(testBitmap)
```



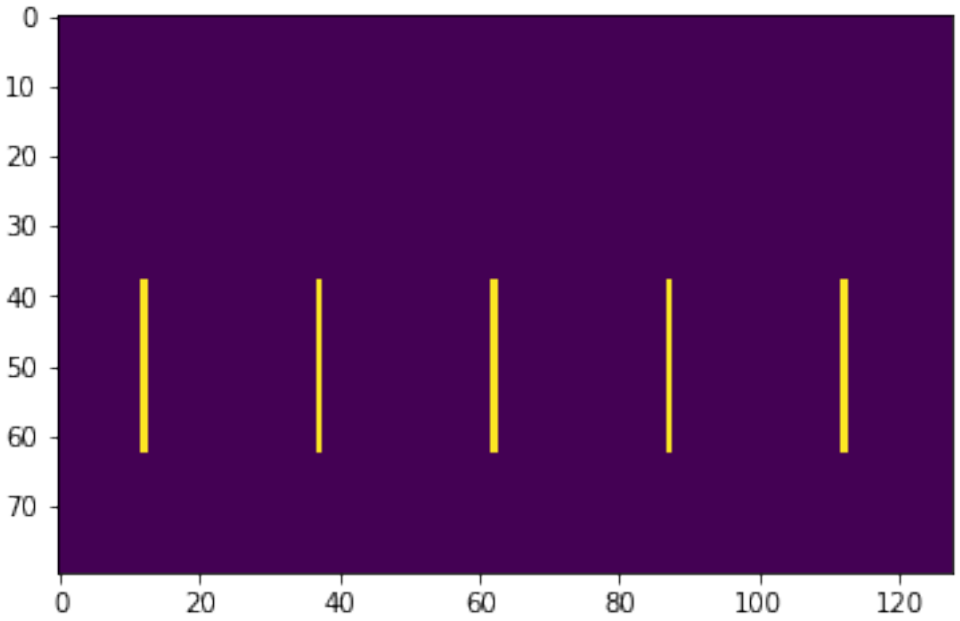
```
In [440]: getWAGA(titi,5)
```

```
Out[440]: 77.44712829589844
```

8.2.6 Test 6

Generate data of type 6

```
In [354]: frame = generateToy(6,80,128,1)
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

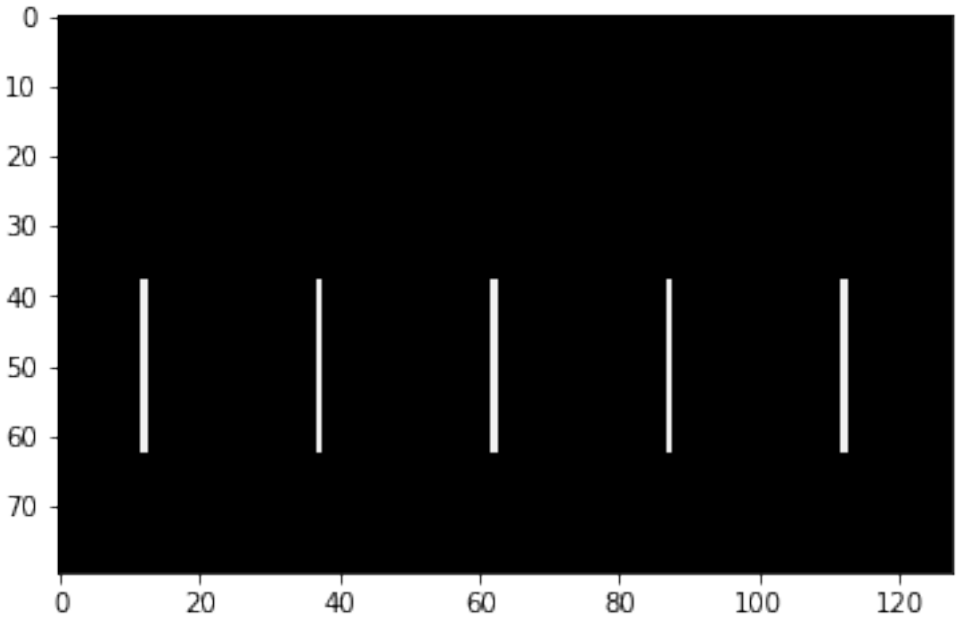
```
In [355]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)
```

```
In [356]: titi.groupby('angle').agg(['mean', 'count'])
```

```
Out[356]:       longueur            weight       precision       xPos       yPos        \
                    mean count        mean count      mean count mean count mean count
          angle
          0.01         7    95  242.906403    95       0.0    95   50    95   62    95

                group       layer
                 mean count  mean count
          angle
          0.01      0    95     1    95
```

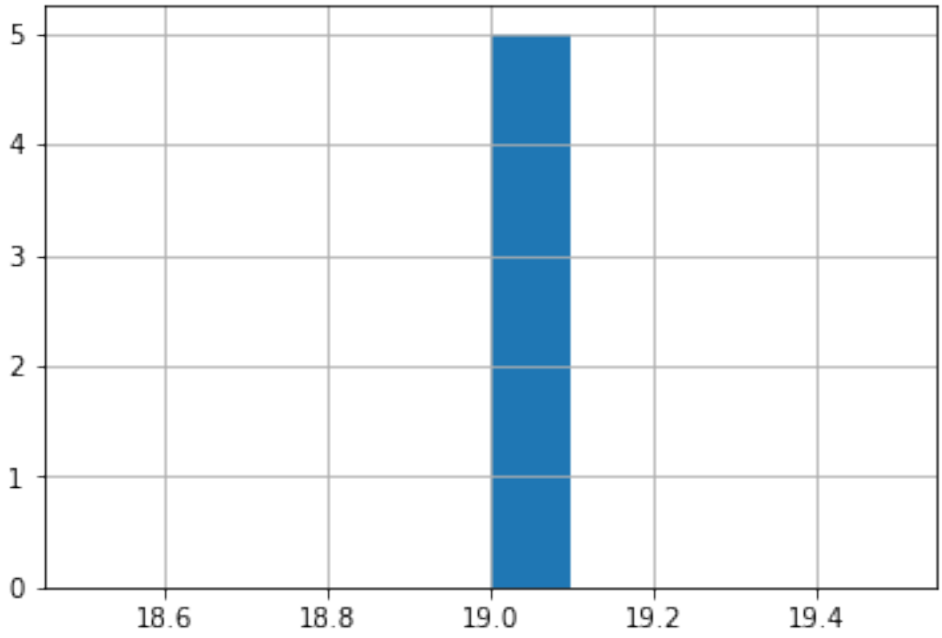
Affichage graphique du champs récepteur des neurones

```
In [357]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap)
          imgplot = plt.imshow(testBitmap)
```



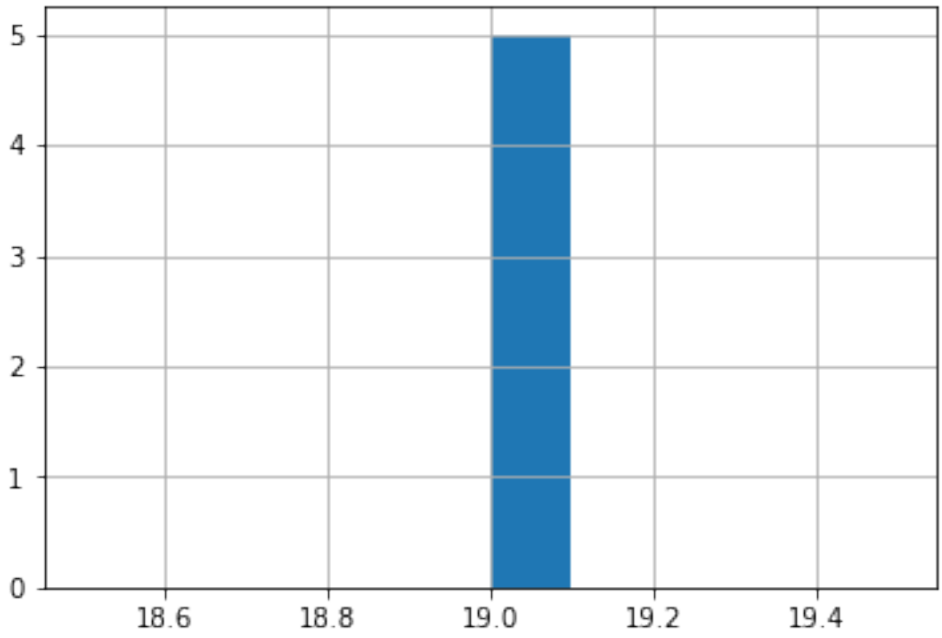
Génération des groupes

```
In [358]: findGroups(titi);

In [359]: titi.groupby('group').size().hist()

Out[359]: <matplotlib.axes._subplots.AxesSubplot at 0x133f20908>
```




```
In [361]: resultGroup = titi.groupby('group').size()
          resultGroup[resultGroup>10].hist()
```

```
Out[361]: <matplotlib.axes._subplots.AxesSubplot at 0x135c3b390>
```



```
In [362]: titi.groupby('group').agg(['mean', 'count'])[resultGroup>10]
```

Out[362]:

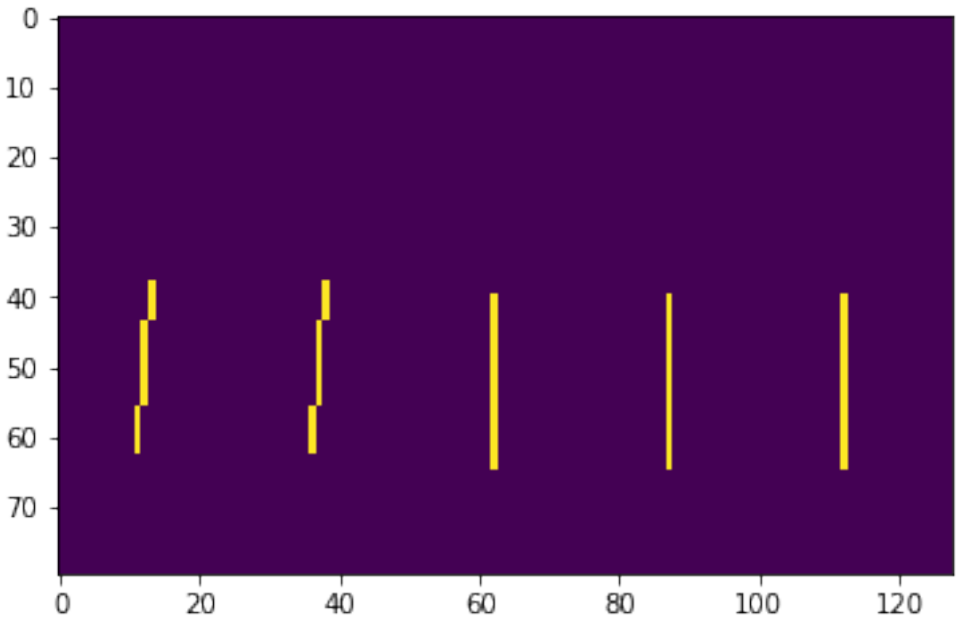
	longueur		angle		weight		precision		xPos		\
	mean	count	mean	count	mean	count	mean	count	mean	count	
group											
1	7	19	0.01	19	242.906403	19	0.0	19	50	19	
2	7	19	0.01	19	242.906403	19	0.0	19	50	19	
3	7	19	0.01	19	242.906403	19	0.0	19	50	19	
4	7	19	0.01	19	242.906403	19	0.0	19	50	19	
5	7	19	0.01	19	242.906403	19	0.0	19	50	19	

	yPos		layer	
	mean	count	mean	count
group				
1	12	19	1	19
2	37	19	1	19
3	62	19	1	19
4	87	19	1	19
5	112	19	1	19

8.2.7 Test 7

Generate data of type 7

```
In [441]: frame = generateToy(7,80,128,1)
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [442]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

In [443]: titi.groupby('angle').agg(['mean', 'count'])
```

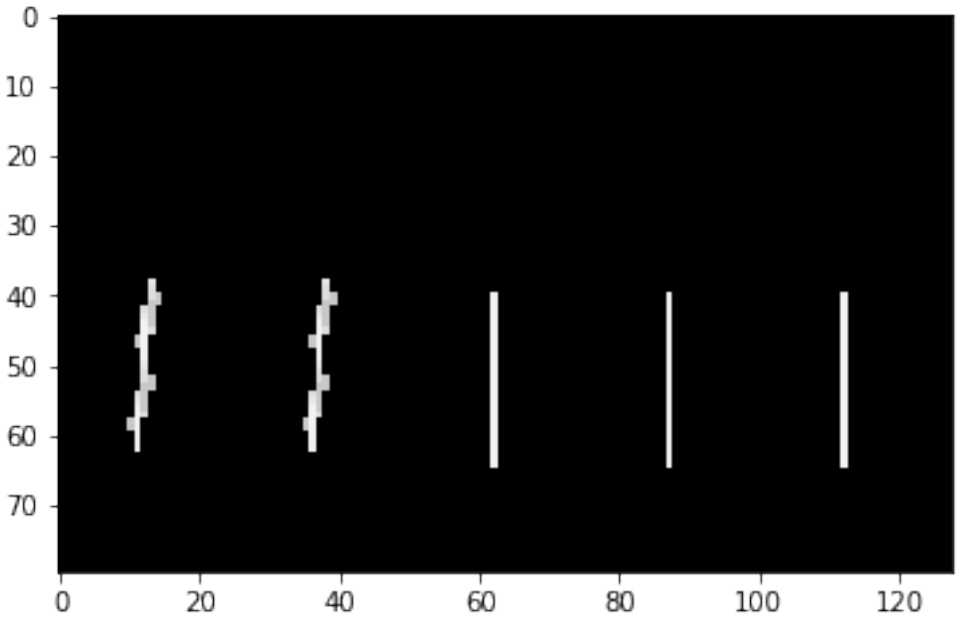
Out[443]:

	longueur		weight		precision		xPos		\
	mean	count	mean	count	mean	count	mean	count	
angle									
0.010000	7	71	242.906403	71	0.000000	71	51.774648	71	
3.081667	7	8	232.039963	8	6.868455	8	49.500000	8	
7.510000	7	8	222.213516	8	10.863776	8	49.500000	8	
15.010000	7	8	200.669205	8	16.934334	8	49.500000	8	

	yPos		group		layer	
	mean	count	mean	count	mean	count
angle						
0.010000	74.647887	71	0	71	1	71
3.081667	24.500000	8	0	8	1	8
7.510000	24.500000	8	0	8	1	8
15.010000	24.500000	8	0	8	1	8

Affichage graphique du champs récepteur des neurones

```
In [444]: testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
          testBitmap = drawFieldNeurons(titi, testBitmap)
          imgplot = plt.imshow(testBitmap)
```

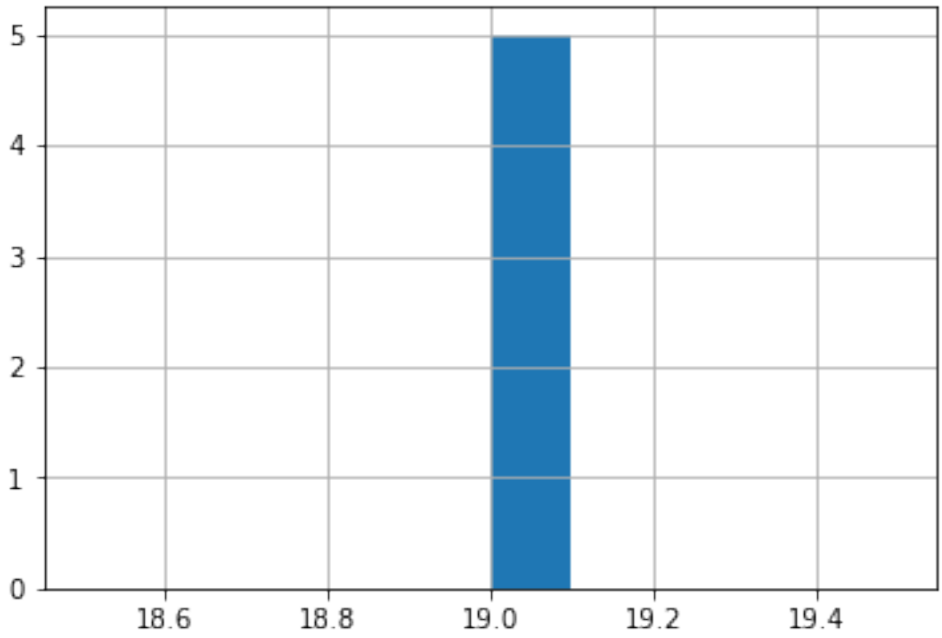


Génération des groupes

```
In [445]: findGroups(titi);
```

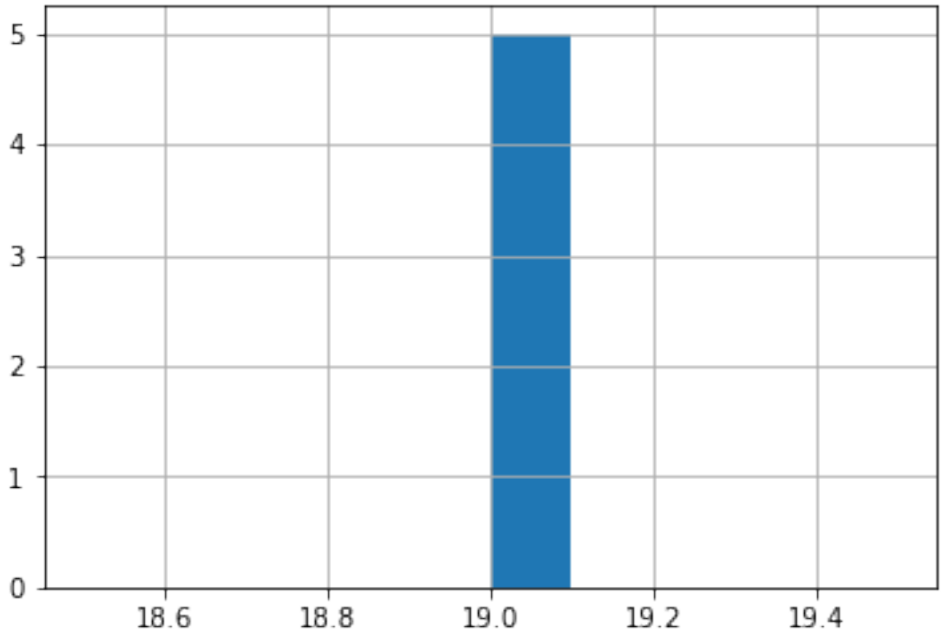
```
In [446]: titi.groupby('group').size().hist()
```

```
Out[446]: <matplotlib.axes._subplots.AxesSubplot at 0x136c1d358>
```




```
In [447]: resultGroup = titi.groupby('group').size()
          resultGroup[resultGroup>10].hist()
```

```
Out[447]: <matplotlib.axes._subplots.AxesSubplot at 0x132eea0f0>
```




```
In [448]: titi.groupby('group').agg(['mean', 'count'])[resultGroup>10]
```

Out[448]:

	longueur		angle		weight		precision		xPos	\
	mean	count	mean	count	mean	count	mean	count	mean	
group										
1	7	19	5.393509	19	227.370285	19	7.298224	19	50	
2	7	19	5.393509	19	227.370285	19	7.298224	19	50	
3	7	19	0.010000	19	242.906403	19	0.000000	19	52	
4	7	19	0.010000	19	242.906403	19	0.000000	19	52	
5	7	19	0.010000	19	242.906403	19	0.000000	19	52	

	yPos		layer	
count	mean	count	mean	count
group				
1	19	11.947368	19	1	19
2	19	36.947368	19	1	19
3	19	62.000000	19	1	19
4	19	87.000000	19	1	19
5	19	112.000000	19	1	19

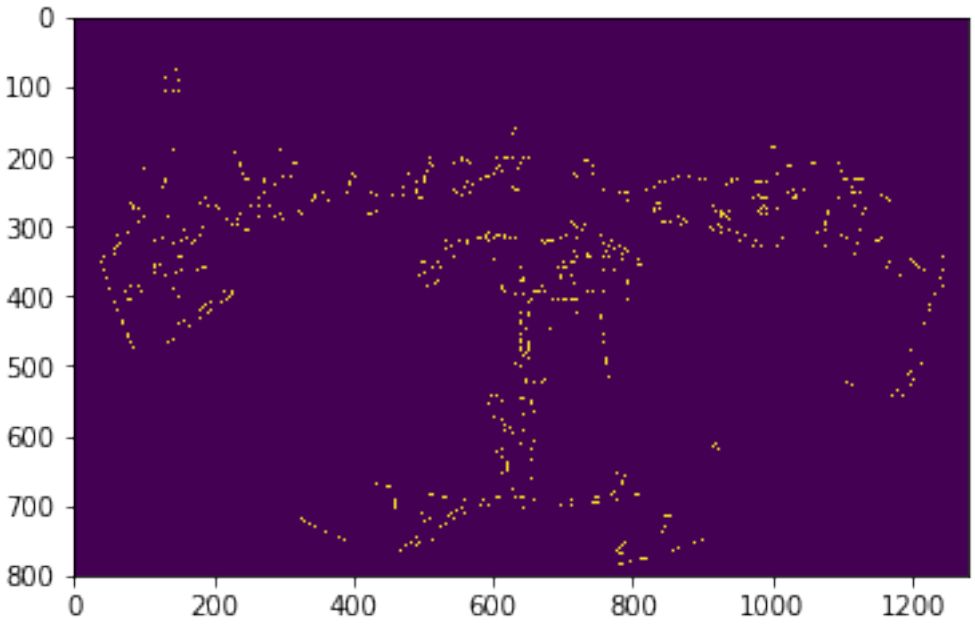
```
In [450]: getWAGA(titi,2)
```

Out[450]: 5.000133991241455

8.2.8 Test 8 : video frame

Get Video frame

```
In [302]: frame = Cannyframe
          imgplot = plt.imshow(frame)
```



Génération des neurones à champs récepteur

```
In [303]: indices = np.where(frame != [0])
          nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField)
          titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                                         nbPixelsAll)

In [304]: titi.describe()

Out[304]:        longueur        angle       weight    precision         xPos  \
          count     6352.0  6352.000000  6352.000000  6352.000000  6352.000000
          mean         9.0     3.500042   155.807388    25.445107   444.441751
          std          0.0    49.856220    75.384323    17.663477   153.220702
          min          9.0   -88.243752     1.932616     0.000000   267.000000
          25%          9.0   -35.947117   106.064775    13.851732   325.000000
          50%          9.0     0.010000   184.319527    20.414985   388.000000
          75%          9.0    47.406427   212.690514    33.394614   547.000000
          max          9.0    90.000000   242.906403    78.747810   784.000000

                       yPos   group   layer
          count  6352.000000  6352.0  6352.0
          mean    649.877204     0.0     1.0
          std     296.119564     0.0     0.0
```

```
min       39.000000     0.0     1.0
25%      523.750000     0.0     1.0
50%      653.000000     0.0     1.0
75%      788.000000     0.0     1.0
max     1246.000000     0.0     1.0
```

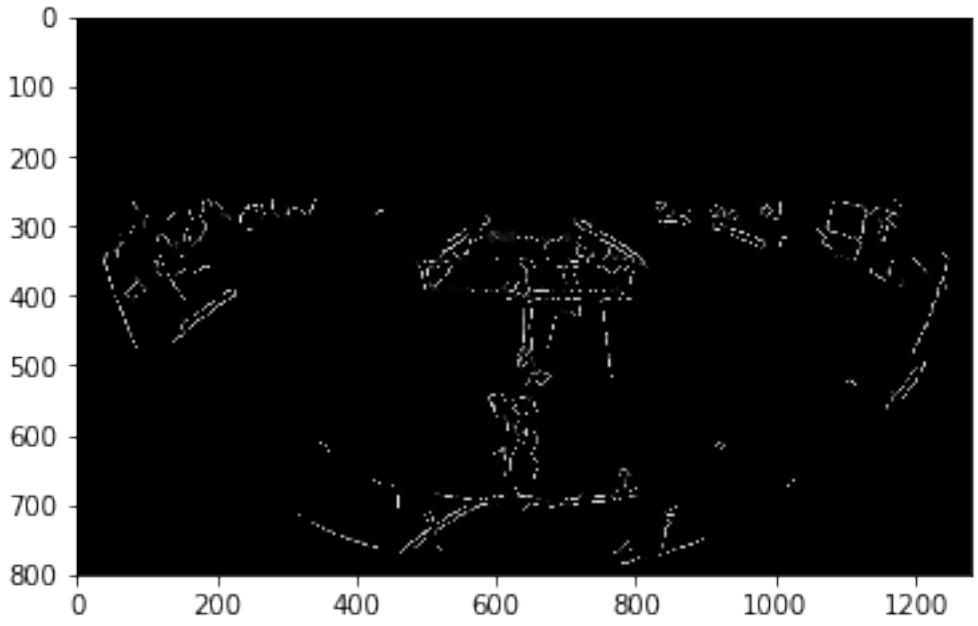
In [305]:
```
titi[0:4]
```

Out[305]:

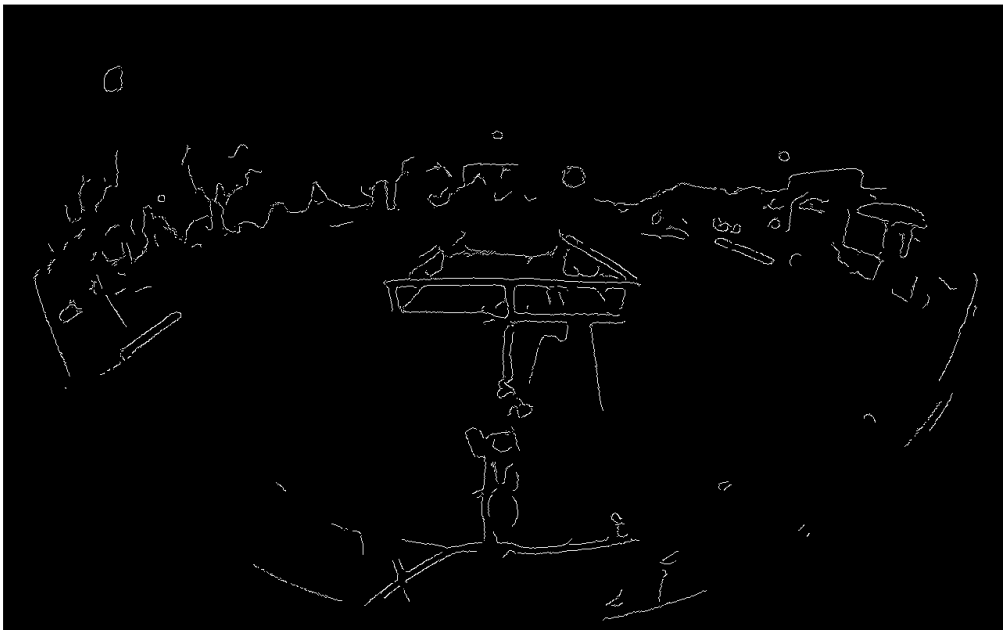
	longueur	angle	weight	precision	xPos	yPos	group	layer
0	9	-14.750000	209.718018	14.671453	267	82	0	1
1	9	-18.597778	178.442535	21.537748	267	180	0	1
2	9	-40.653751	139.374527	28.131916	267	191	0	1
3	9	-2.295000	151.148193	26.247042	267	266	0	1

Affichage graphique du champs récepteur des neurones

In [306]:
```
testBitmap = np.zeros((frame.shape[0],frame.shape[1],3), np.uint8)
testBitmap = drawFieldNeurons(titi, testBitmap)
imgplot = plt.imshow(testBitmap)
```



In [307]:
```
lintI = 0
while (lintI < 10):
    cv2.imshow('testBitmap', testBitmap)
    if cv2.waitKey(1) & 0xFF == ord('q'): # press q to quit
        break
    lintI += 1
```



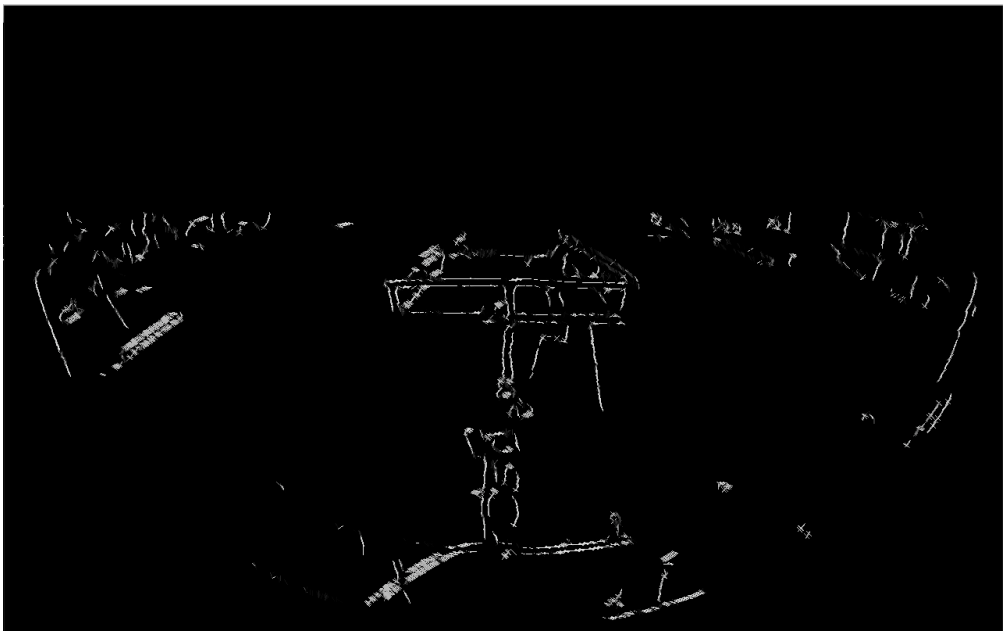
Edge detection frame image, with the Canny Algorithm

Affichage de la frame :

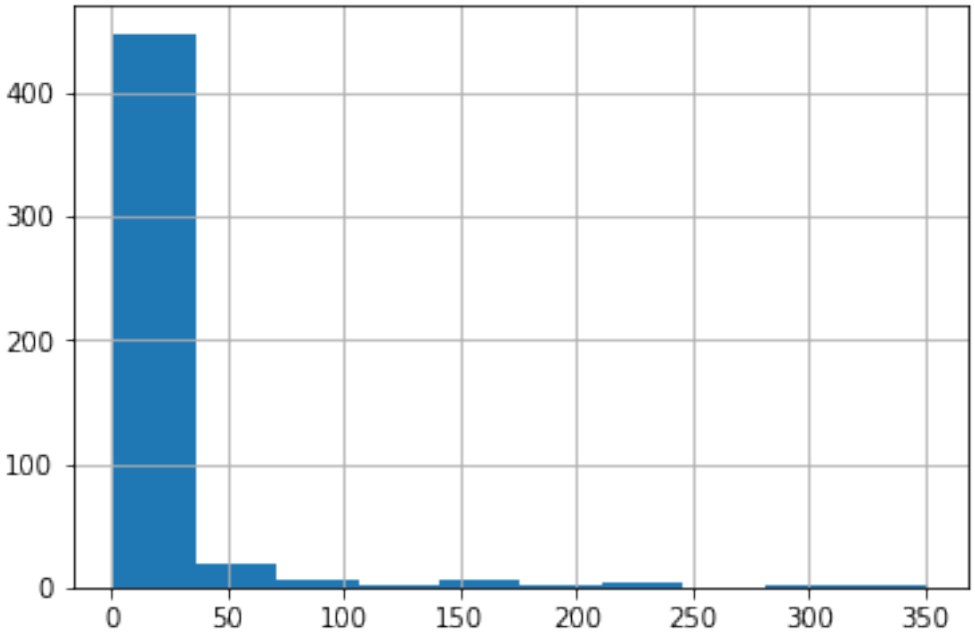
Affichage des champs récepteurs des neurones

Génération des groupes

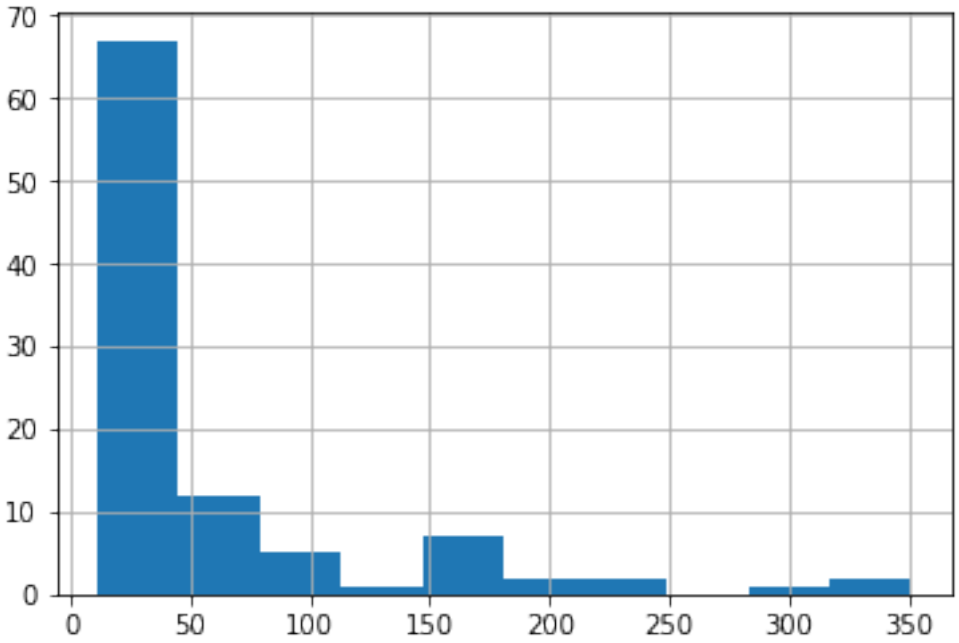
```
In [308]: findGroups(titi);

In [309]: titi.groupby('group').size().hist()

Out[309]: <matplotlib.axes._subplots.AxesSubplot at 0x13460b908>
```



NeuronField Image : Le niveau de gris correspond au niveau d'activation du neurone




```
In [310]: resultGroup = titi.groupby('group').size()
          resultGroup[resultGroup>10].hist()

Out[310]: <matplotlib.axes._subplots.AxesSubplot at 0x13464f630>
```




```
In [371]: titi.groupby('group').agg(['mean', 'count'])[resultGroup>130]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-371-c1c777f13774> in <module>
----> 1 titi.groupby('group').agg(['mean', 'count'])[resultGroup>130].sort_values(by=['cou


/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in sort_values(self, by, a
   4717
   4718             by = by[0]
-> 4719             k = self._get_label_or_level_values(by, axis=axis)
   4720
   4721             if isinstance(ascending, (tuple, list)):


/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in _get_label_or_level_v
   1704             values = self.axes[axis].get_level_values(key)._values
   1705         else:
-> 1706             raise KeyError(key)
   1707
   1708         # Check for duplicates
```

```
KeyError: 'count'
```

9 Errors list

9.1 Error 10

Problème dans la fonction Section **??**

In [ ]: