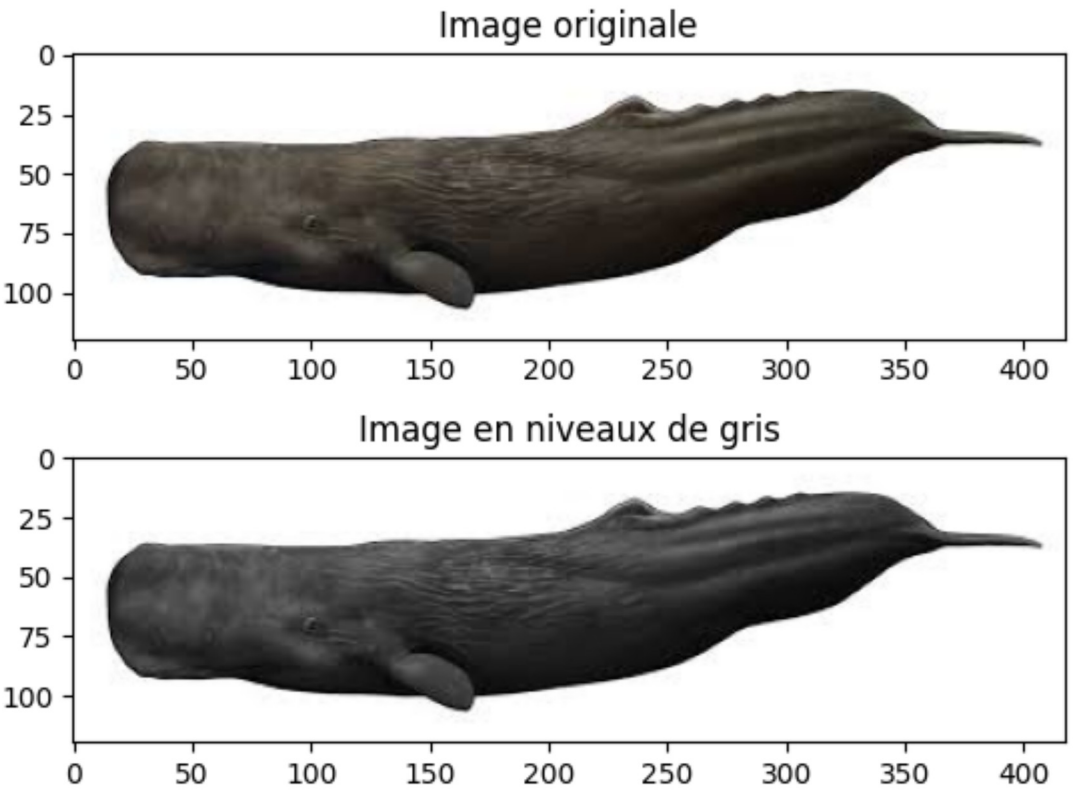
TP1 : Un perceptron comme modèle de rétine

- Coder un perceptron qui modélise les sorties des cellules gangionnaires de la rétine
- Si vous avez le temps, ajouter ensuite une couche de neurones qui détecte les horizontales

In [1]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

im = Image.open(r"C:\Users\phand\Documents\M1\Neurone\TP1\cachalot.jpeg")
m3d = np.array(im)
m2d = np.sum(m3d, axis=2)

plt.subplot(2, 1, 1)
plt.imshow(m3d)
plt.title("Image originale")
plt.subplot(2, 1, 2)
plt.imshow(m2d, cmap="gray")
plt.title("Image en niveaux de gris")
plt.show()
```



Ici, on récupère l'image "cachalot.jpeg", on la convertit en un tableau numpy, puis on crée une version en niveaux de gris en sommant les valeurs des canaux de couleur. Enfin, on affiche les deux images côte à côte pour comparaison. On passe alors d'une image en couleur (3D) à une image en niveaux de gris (2D).

Pour reproduire le fonctionnement d une cellule ganglionnaire de la retine, on va déterminer à quoi correspondent les poids synaptiques entre les neurones récepteurs (les

pixels de l image) et la cellule ganglionnaire. On va alors :

- Fixer des poids synaptiques
- Calculer la somme pondérée des entrées
- Appliquer une fonction d activation

Chaque pixel de l image d entree correspond a un neurone recepteur Chaque neurone recepteur est connecte a la cellule ganglionnaire par un poids synaptique

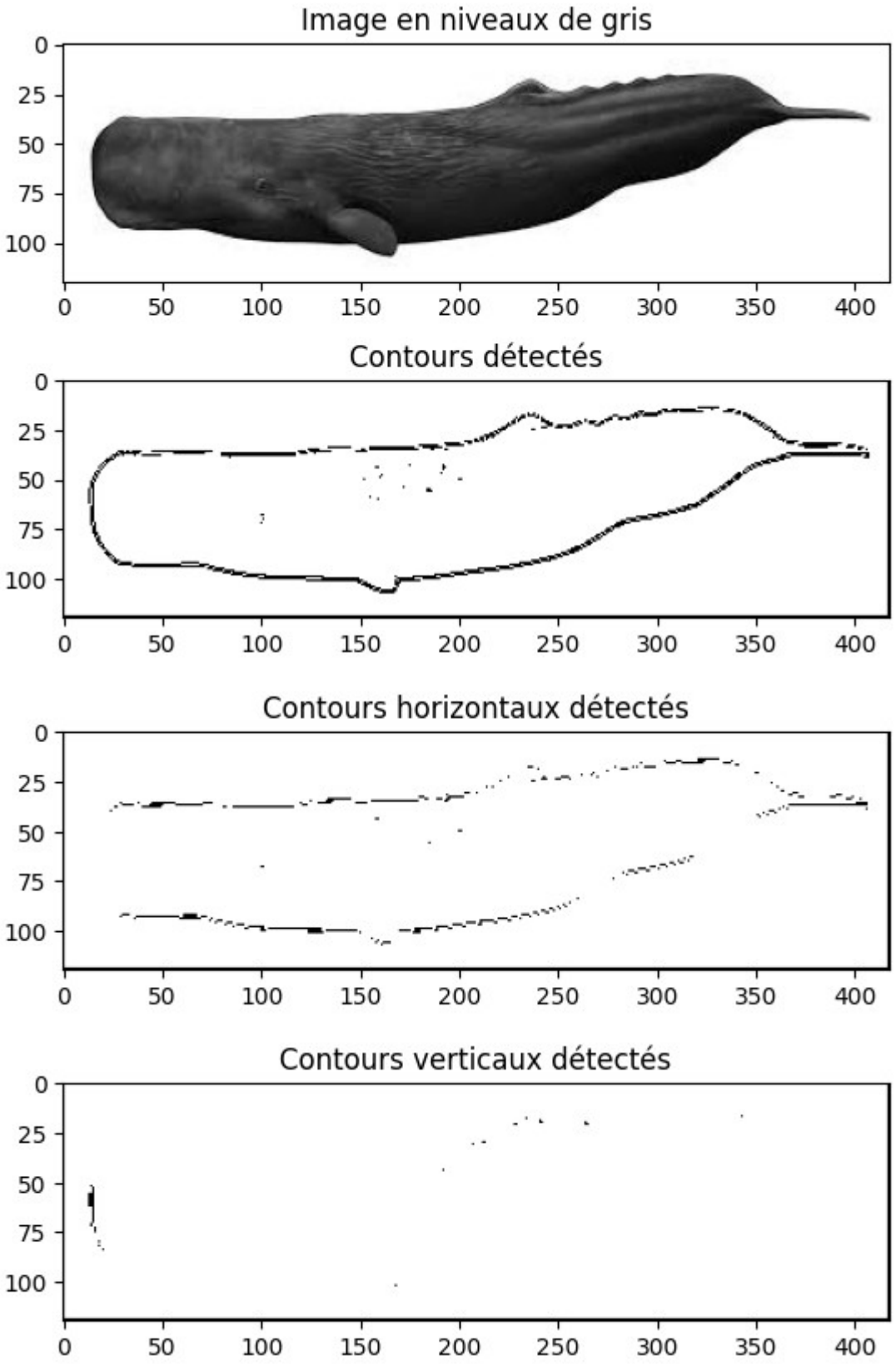
Donc en entréee on a une image 2D (matrice de pixels) On va appliquer a chaque pixel une somme ponderee des poids synaptiques * valeur du pixel Puis appliquer une fonction d activation Et en sortie on a une image 2D (matrice de pixels) qui correspond a la reponse de la cellule ganglionnaire Si tout va bien l image de sortie doit etre des contours de l image d entree.

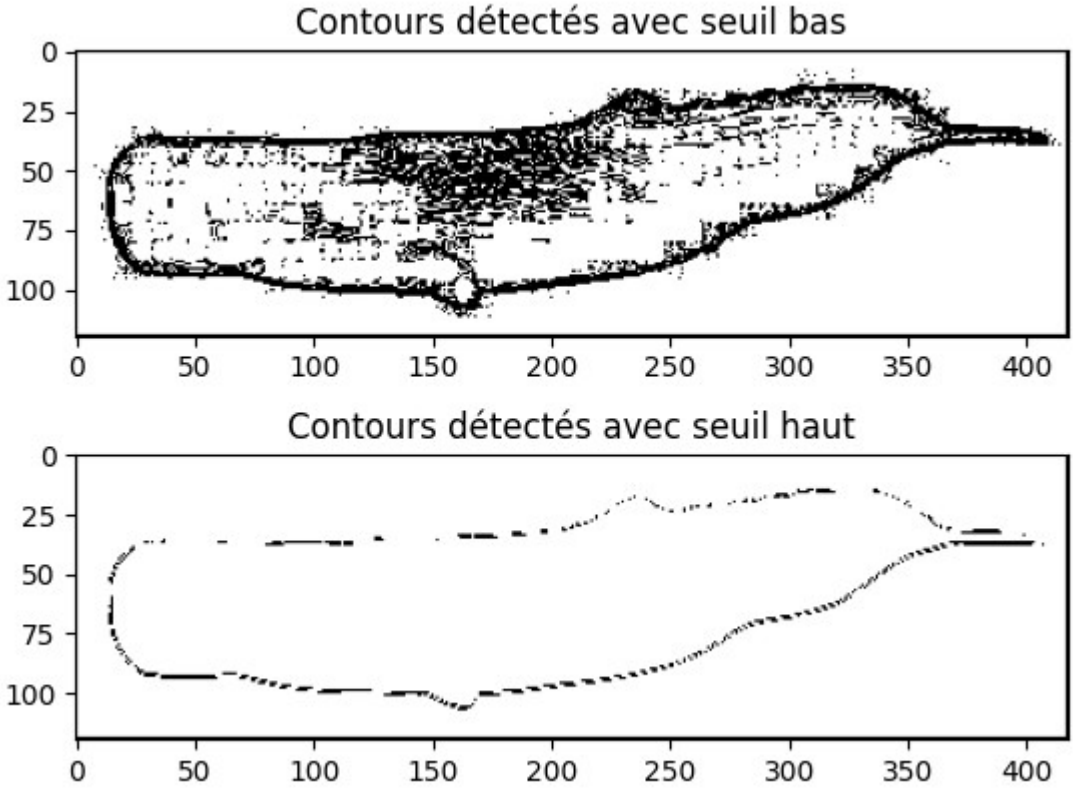
Note : on a i*j de poids sauf qu il y a des poids nuls : les seuls poids non nuls sont ceux qui entourent le pixel central

In [8]:
```
# from PIL import Image
# import numpy as np
# import matplotlib.pyplot as plt

# im = Image.open(r"C:\Users\phand\Documents\M1\Neurone\TP1\cachalot.jpeg")
# m3d = np.array(im)
# m2d = np.sum(m3d, axis=2)

# plt.subplot(2, 1, 1)
# plt.imshow(m3d)
# plt.title("Image originale")
# plt.subplot(2, 1, 2)
# plt.imshow(m2d, cmap="gray")
# plt.title("Image en niveaux de gris")
# plt.show()


# Représente une cellule ganglionnaire de la rétine
poids = np.array([[-1, -1, -1],
                  [-1,  8, -1],
                  [-1, -1, -1]])

poids_verticaux = np.array([[ -1,   2,  -1],
                            [ -1,   2,  -1],
                            [ -1,   2,  -1]])

poids_horizontaux = np.array([[ -1,  -1,  -1],
                              [  2,   2,   2],
                              [ -1,  -1,  -1]])

# Fonctions d'activations
def heavyside(x):
    if x < 0:
        return 0
    else:
        return 1

def identite(x):
```

```
    return x

# Calcul de la sortie de la cellule ganglionnaire
def somme_poids_entree(m2d, poids, seuil=765):
    # On créer une matrice de sortie de meme taille que l image d entree
    lignes, colonnes = m2d.shape
    sortie = np.zeros((lignes, colonnes))

    for i in range(1, lignes - 1):
        for j in range(1, colonnes - 1):
            # Extraction matrice locale 3x3 autour du pixel (i, j)
            matrice_locale = m2d[i-1:i+2, j-1:j+2]

            # E w_ij * matrice_locale
            somme_poids_entree = np.sum(matrice_locale * poids)

            # Seuil
            if abs(somme_poids_entree) > seuil : # Ici on fait une valeur absolu
                # Fonction d'activation
                sortie[i, j] = heavyside(somme_poids_entree)
            else:
                sortie[i, j] = 255  # Pixel blanc
    return sortie


contours = somme_poids_entree(m2d, poids)
plt.subplot(2, 1, 1)
plt.imshow(m2d, cmap="gray")
plt.title("Image en niveaux de gris")
plt.subplot(2, 1, 2)
plt.imshow(contours, cmap="gray")
plt.title("Contours détectés")
plt.show()


# Détection des contours horizontaux et verticaux
horizontales = somme_poids_entree(contours, poids_horizontaux)
plt.subplot(2, 1, 1)
plt.imshow(horizontales, cmap="gray")
plt.title("Contours horizontaux détectés")
plt.show()

verticales = somme_poids_entree(contours, poids_verticaux)
plt.subplot(2, 1, 1)
plt.imshow(verticales, cmap="gray")
plt.title("Contours verticaux détectés")
plt.show()




# Cas seuil trop bas
contours_seuil_bas = somme_poids_entree(m2d, poids, seuil=100)
plt.subplot(2, 1, 1)
plt.imshow(contours_seuil_bas, cmap="gray")
plt.title("Contours détectés avec seuil bas")

# Cas seuil trop haut
contours_seuil_haut = somme_poids_entree(m2d, poids, seuil=1400)
plt.subplot(2, 1, 2)
```

```
plt.imshow(contours_seuil_haut, cmap="gray")
plt.title("Contours détectés avec seuil haut")
plt.show()
```





Ici, on applique un filtre de détection des contours à une image en niveaux de gris en utilisant une matrice de poids spécifique (comme dans une cellule ganglionnaire de la rétine). Puis on regarde par rapport à un seuil si la somme pondérée des pixels locaux dépasse ce seuil pour décider si un contour est détecté ou non. Si le seuil est dépassé, on applique une fonction d'activation (Heaviside) pour déterminer la valeur de sortie. On obtient ainsi une image binaire mettant en évidence les contours présents dans l'image originale.

Pour les contours horizontaux et verticaux, on utilise des matrices de poids différentes adaptées à la détection de ces orientations spécifiques que l'on applique successivement à l'image obtenue après le premier passage dans la cellule ganglionnaire.

Remarques :
Le choix du seuil est crucial pour la détection des contours.
Un seuil trop bas peut entraîner la détection de bruit, tandis qu'un seuil trop élevé peut faire manquer des contours importants.

L'utilisation de différentes matrices de poids permet de cibler des orientations spécifiques des contours, ce qui est utile pour l'analyse d'images. (cf: matrices utilisées pour verticales et horizontales). On pourrait prendre des matrices non carrées ou de tailles différentes pour détecter des contours à d'autres échelles ou orientations.

L'efficacité de cette méthode dépend également de la qualité de l'image d'entrée et des caractéristiques des contours présents.