Détection de fractures osseuses

candidat #12184

HBI-120 de *Viken Detection*

Principe général

$$\theta > \varepsilon$$

Détection des contours

Avec l'algorithme Canny

cv2.Canny →

seuils: 40, 120

Détection des contours

Avec l'algorithme Canny

Détection des contours
Avec l'algorithme Canny

Détection des contours
Avec l'algorithme Canny

Non-contour
< seuil bas

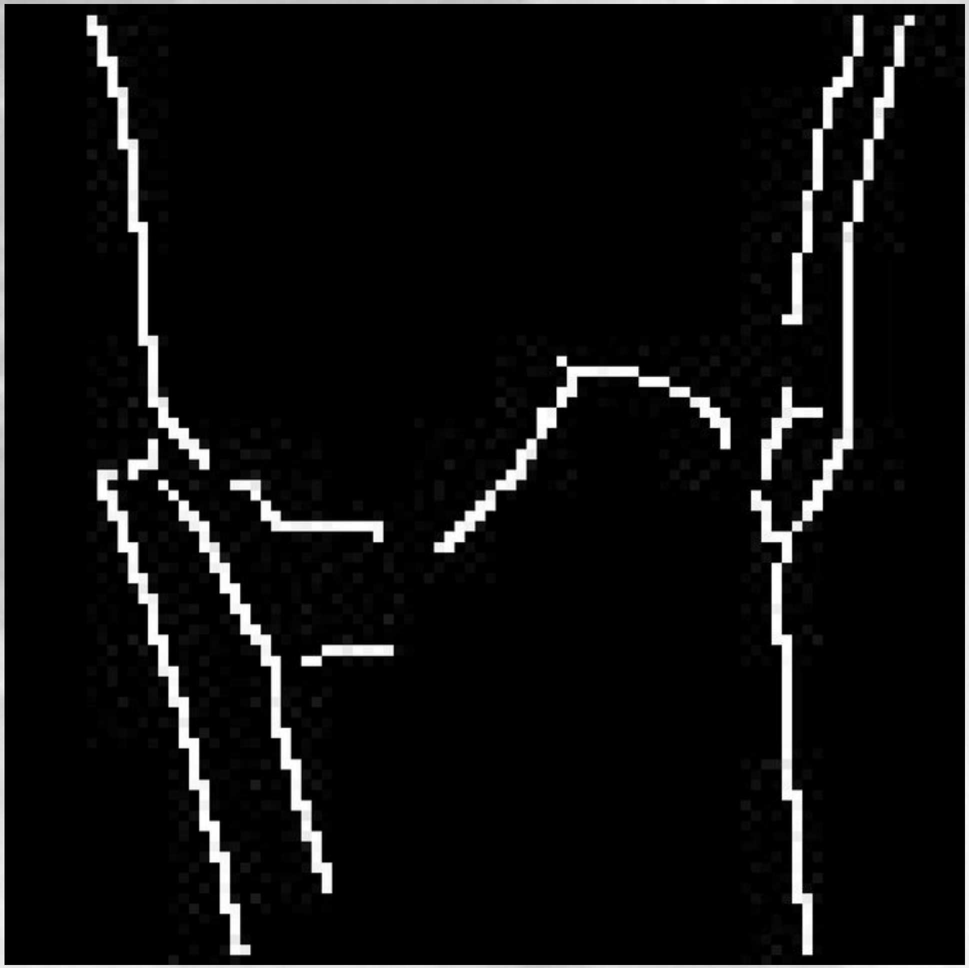
Détection des contours
Avec l'algorithme Canny

Contour faible
∈ [seuil bas, seuil haut[

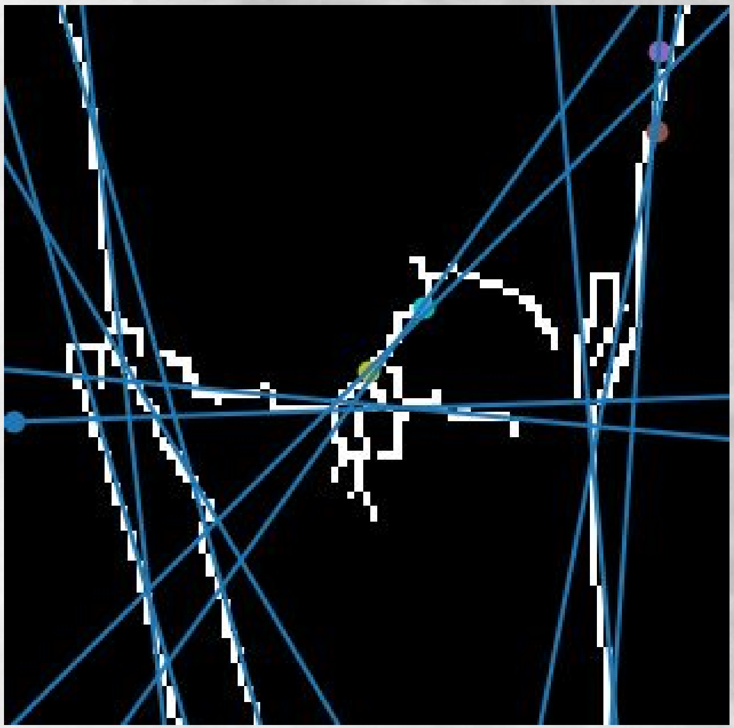
Contour fort
≥ seuil haut

Détection des contours

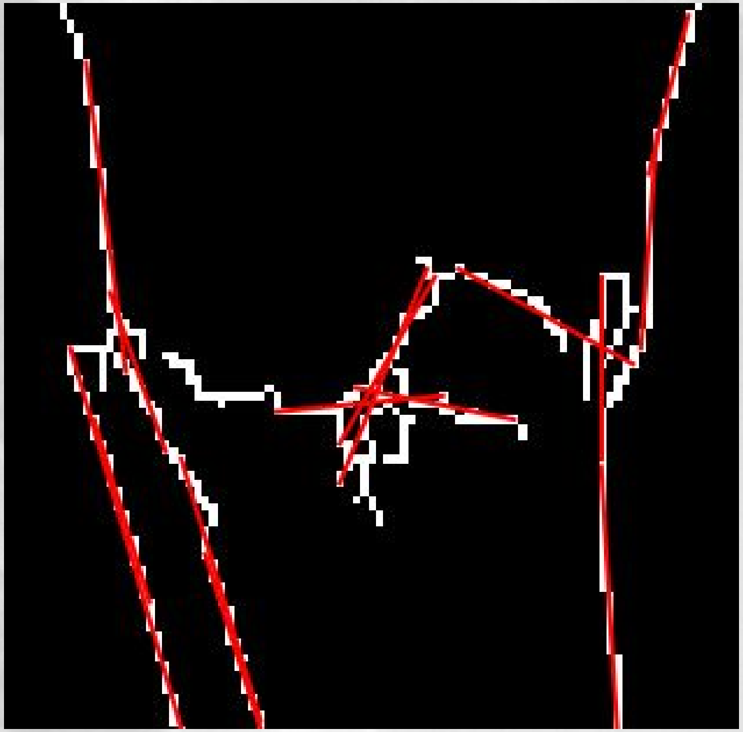
Avec l'algorithme Canny



Détection des traits
Avec la Transformée de Hough



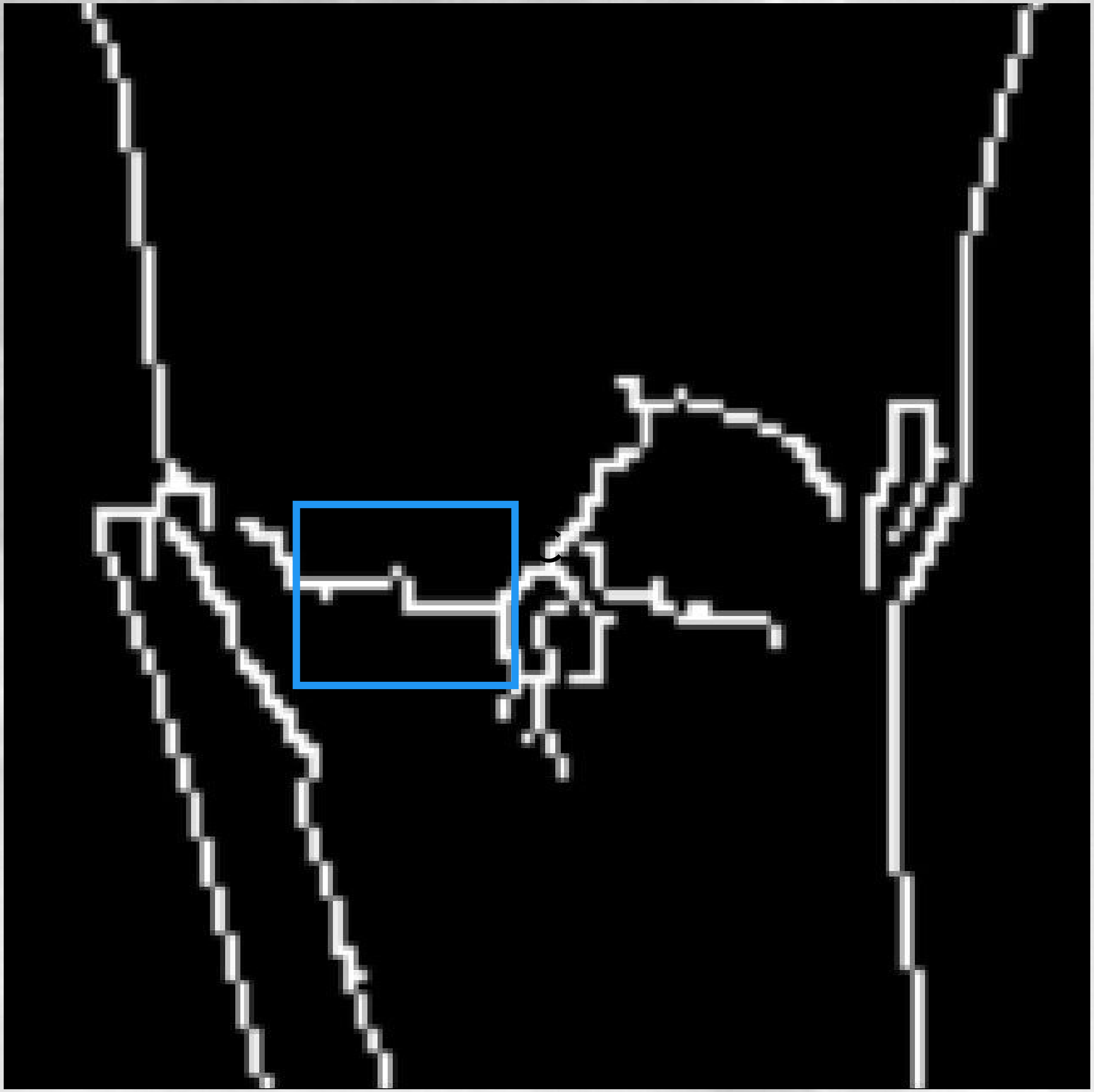
Classique
(détecte des droites)



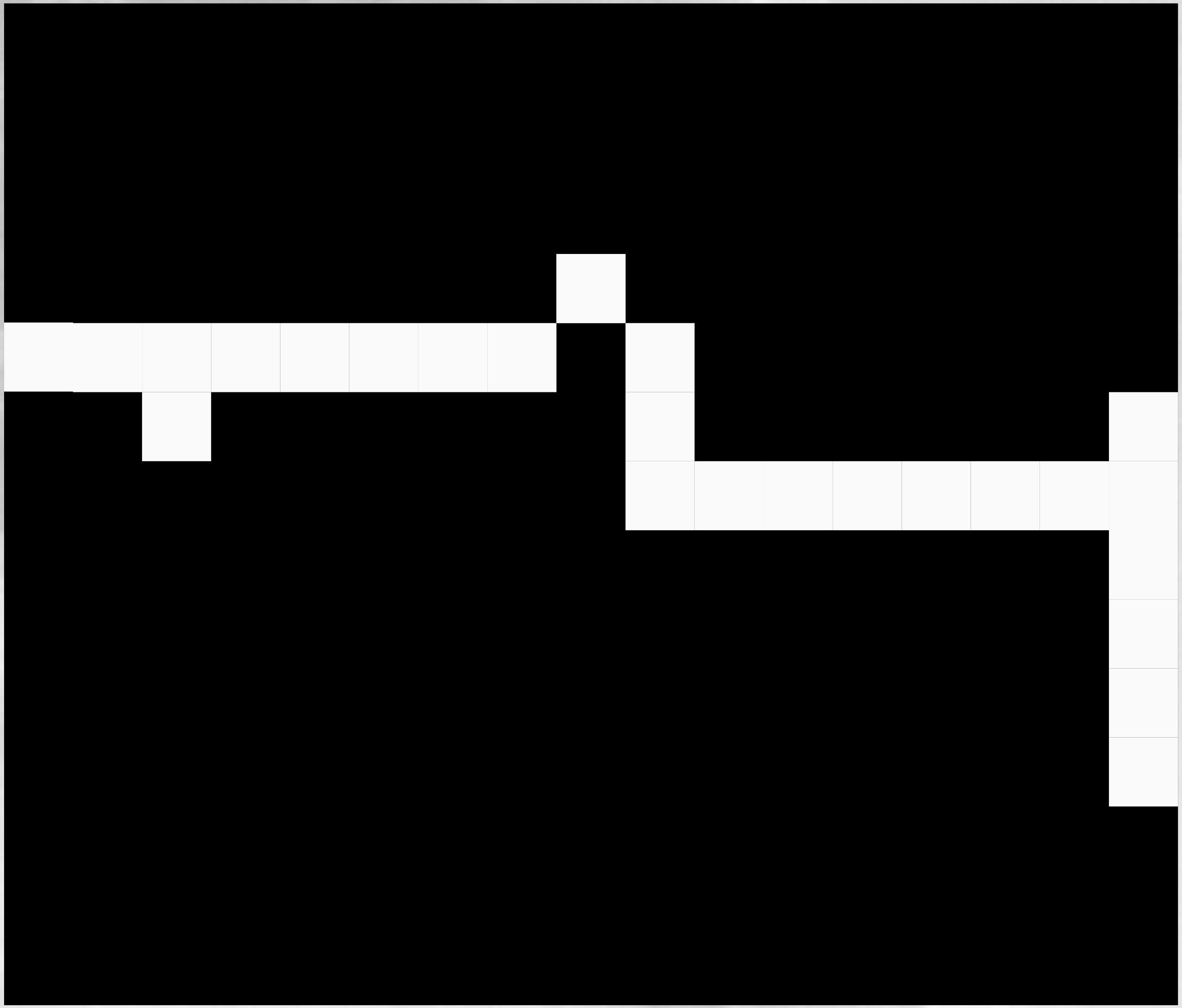
Probabiliste
(détecte des segments)

Détection des traits

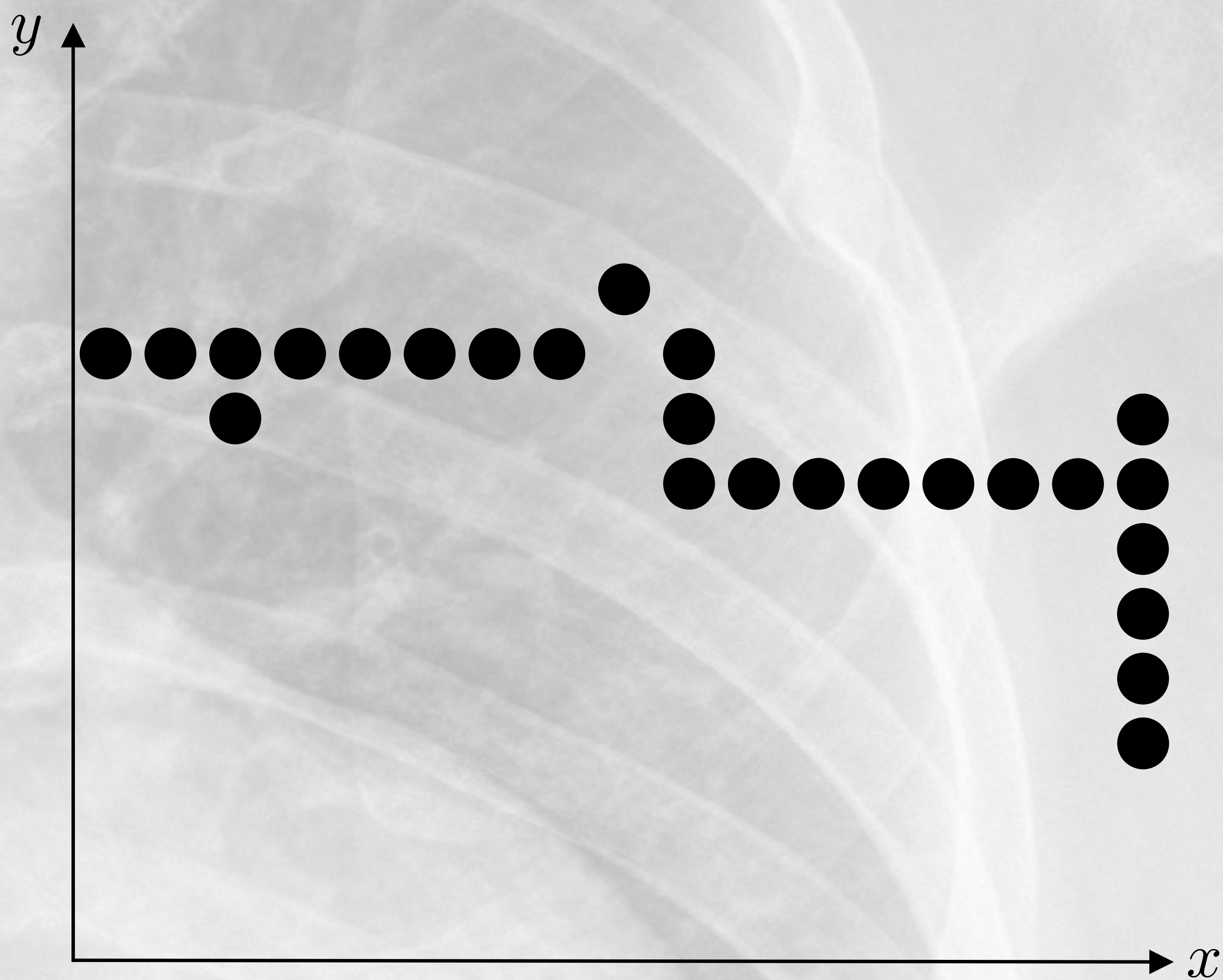
Avec la Transformée de Hough



Détection des traits
Avec la Transformée de Hough



Détection des traits
Avec la Transformée de Hough



Détection des traits
Avec la Transformée de Hough

Détection des traits

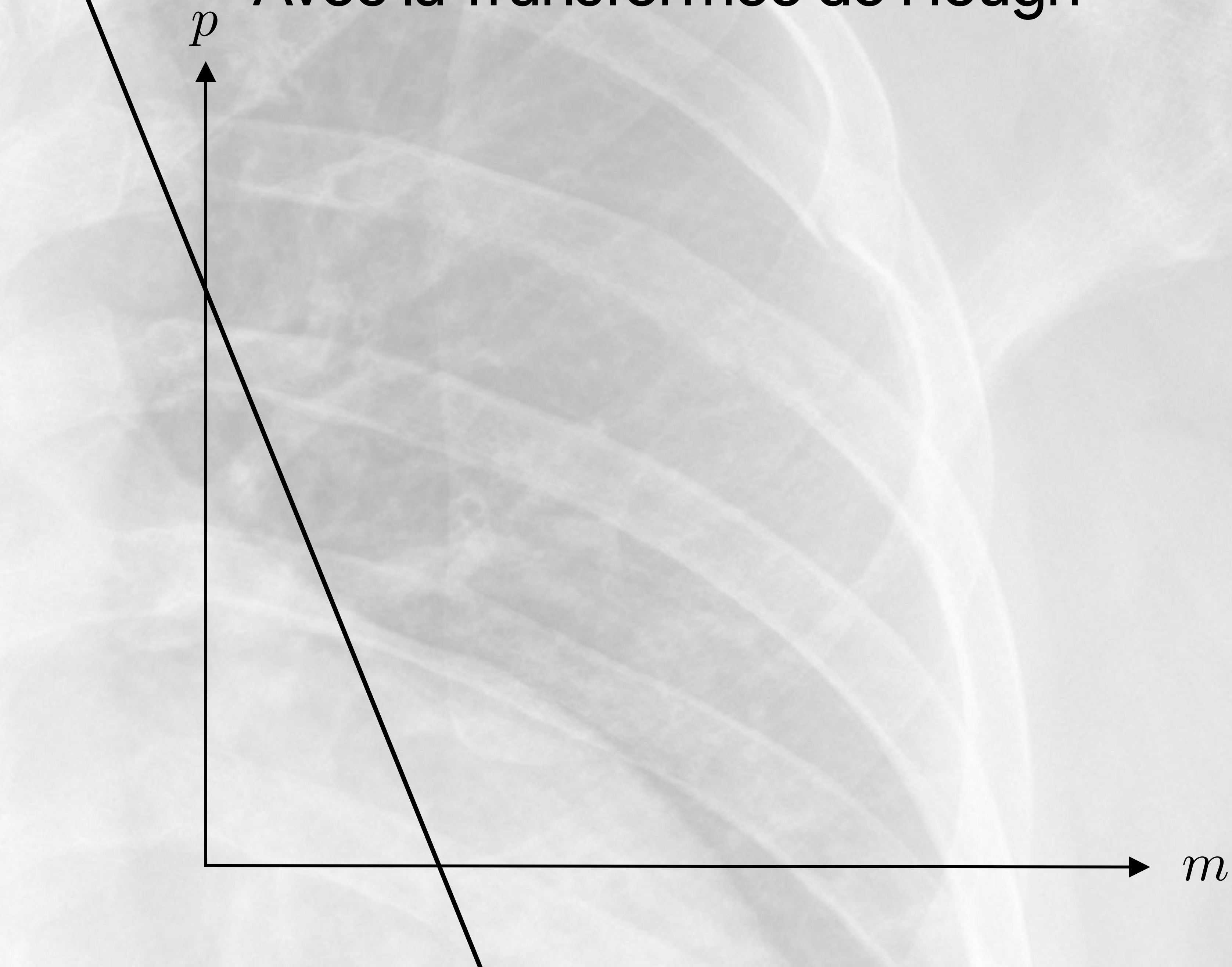
Avec la Transformée de Hough

$$y = mx + p$$

$$10 = m(4) + p \iff p = 10 - 4m$$

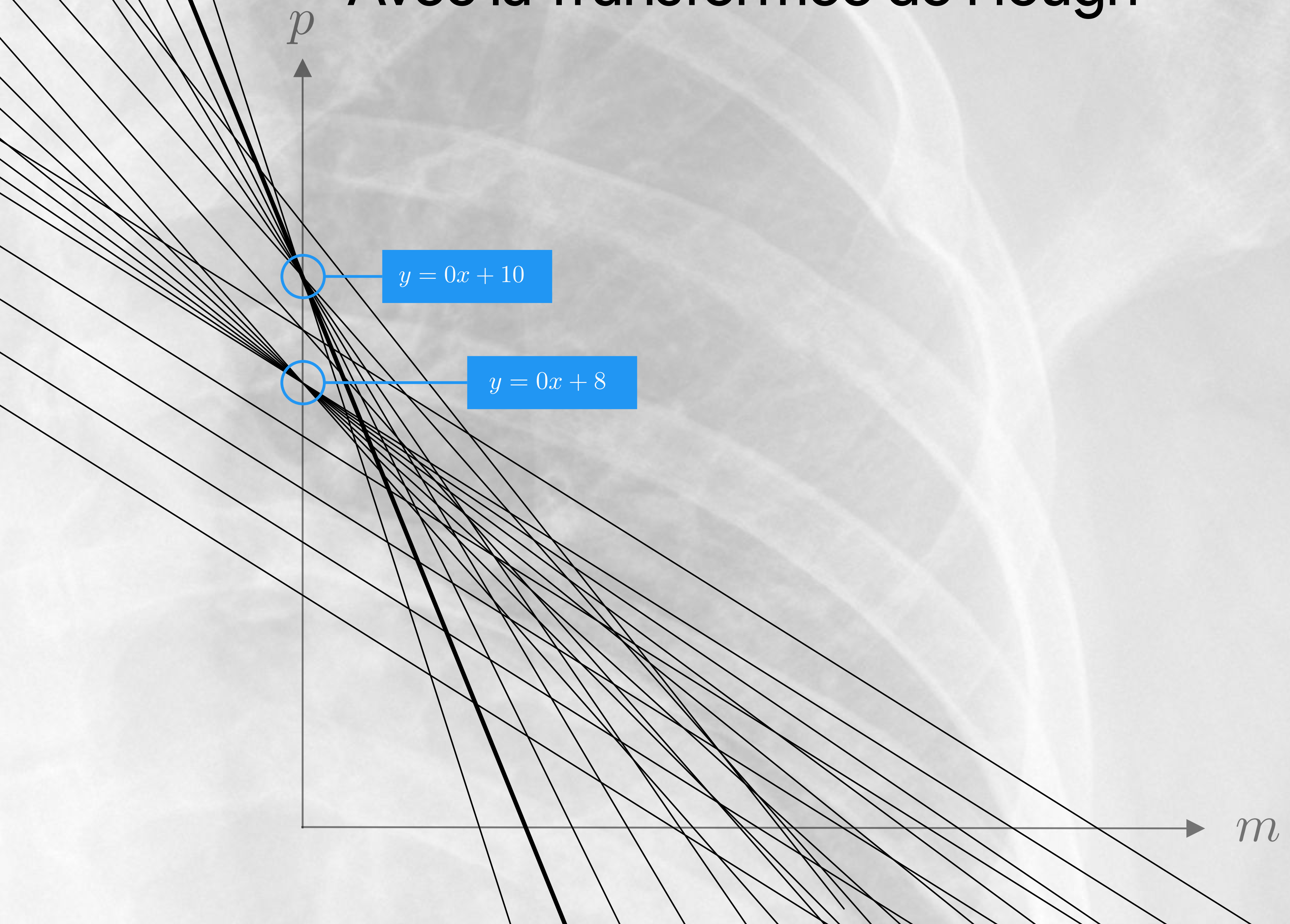
Détection des traits

Avec la Transformée de Hough



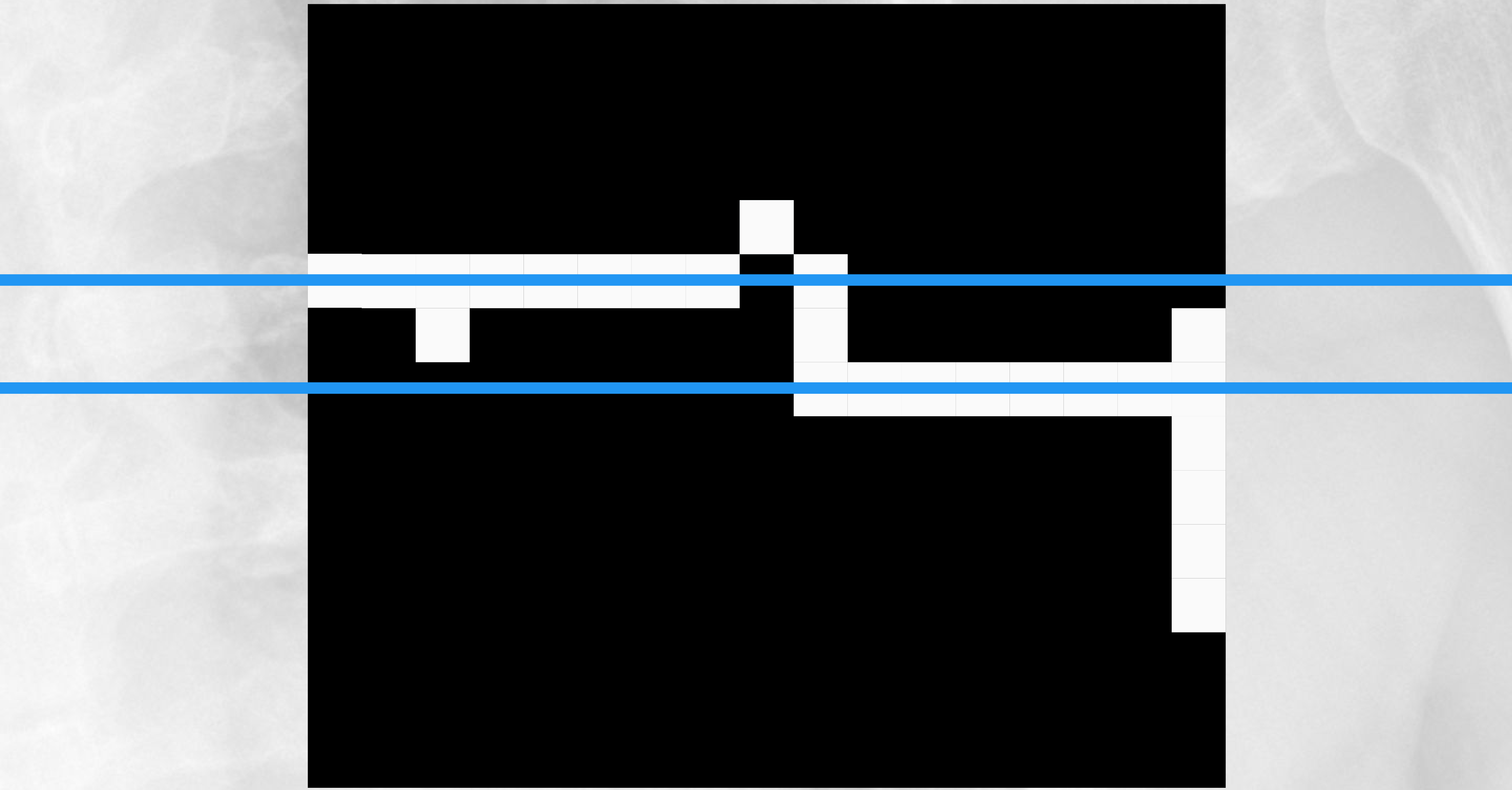
Détection des traits

Avec la Transformée de Hough



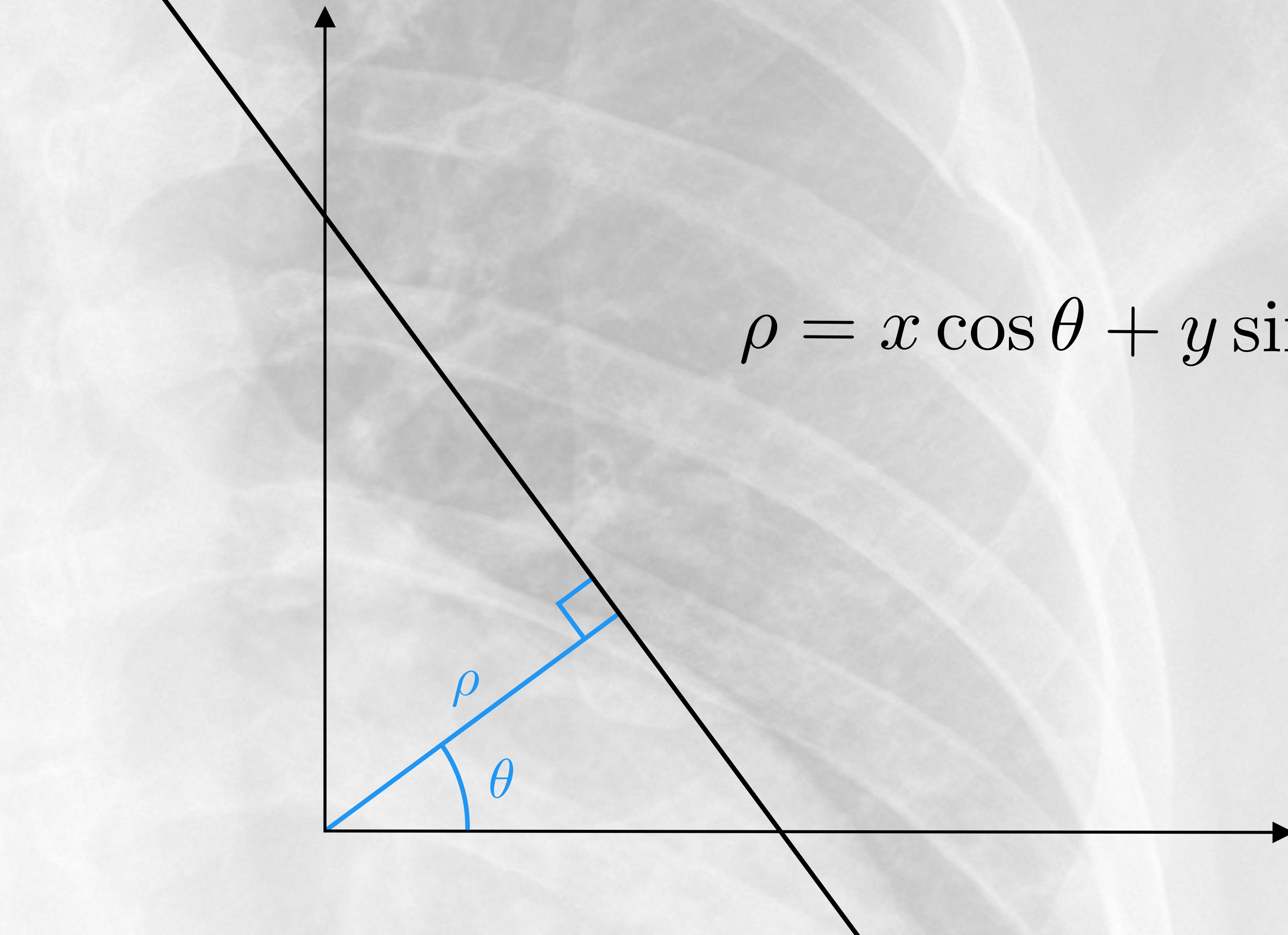
Détection des traits

Avec la Transformée de Hough



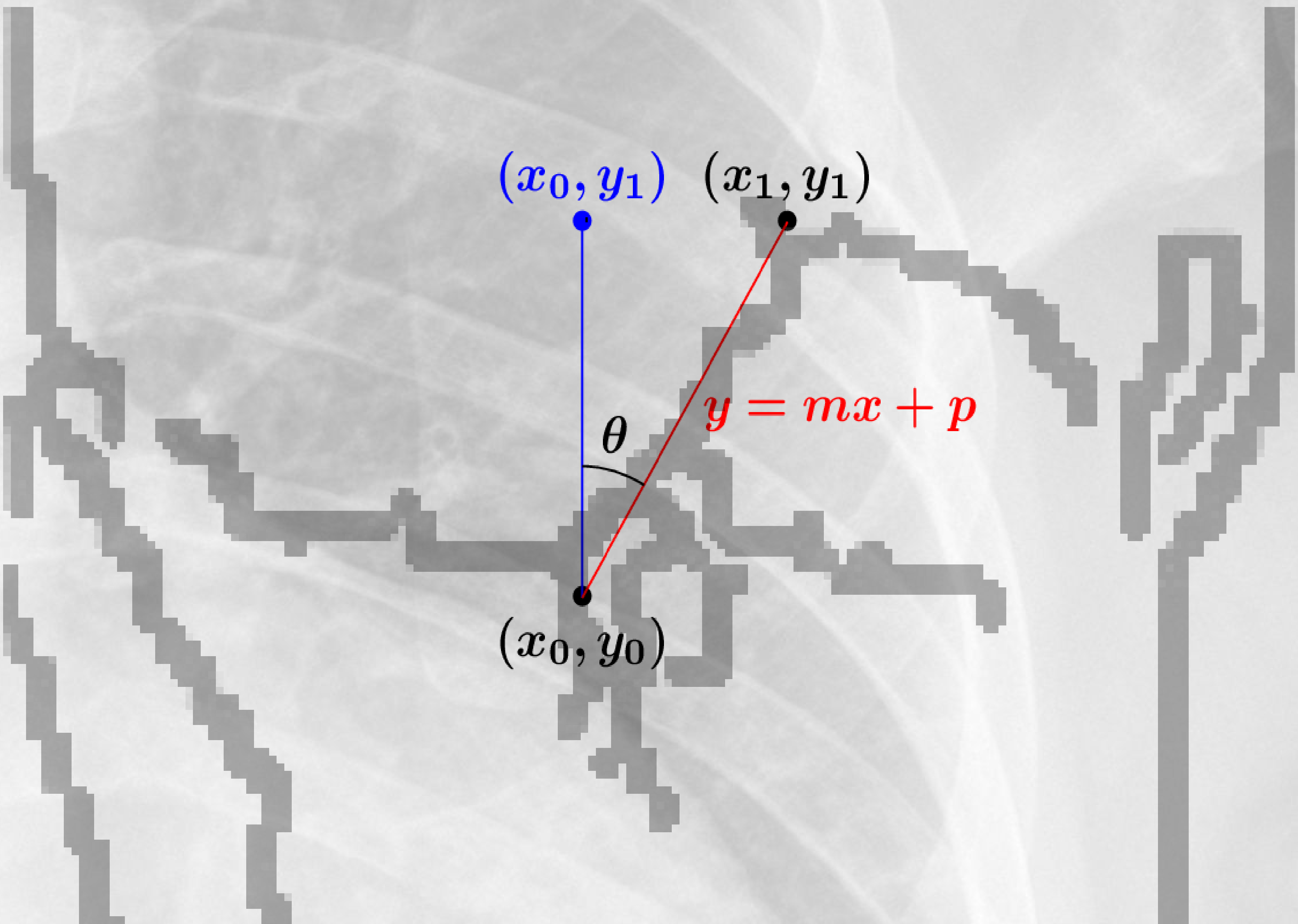
Détection des traits

Avec la Transformée de Hough

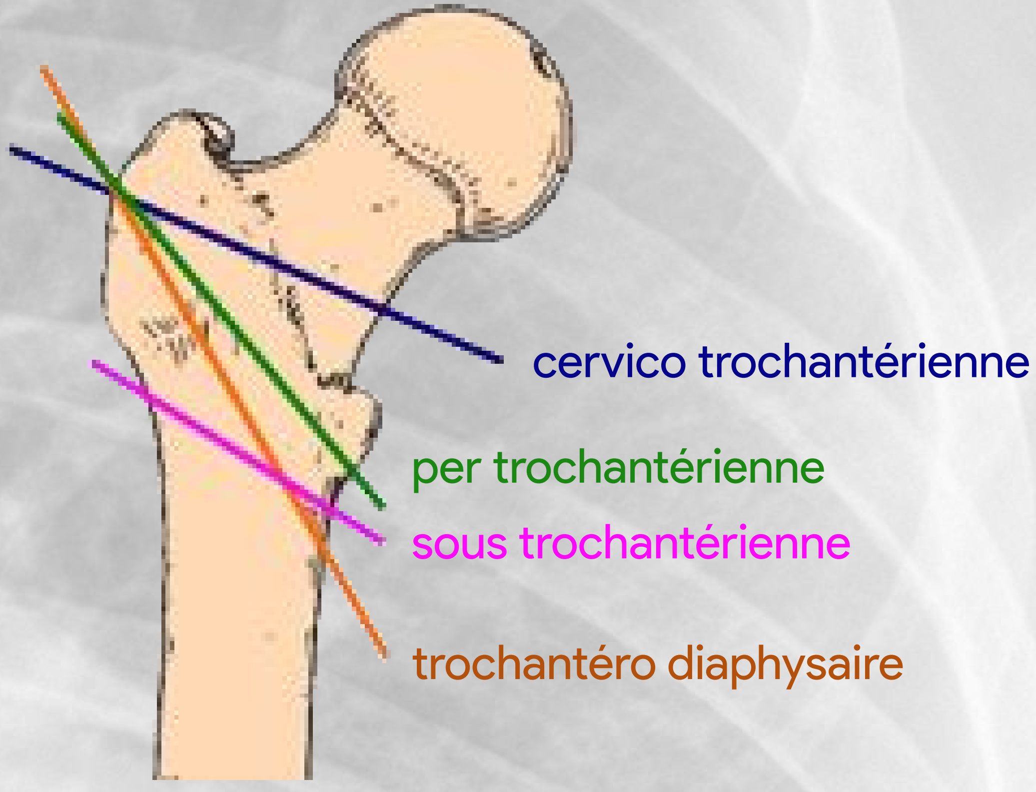


$$\rho = x \cos \theta + y \sin \theta$$

Calcul des angles
Avec de la trigonométrie



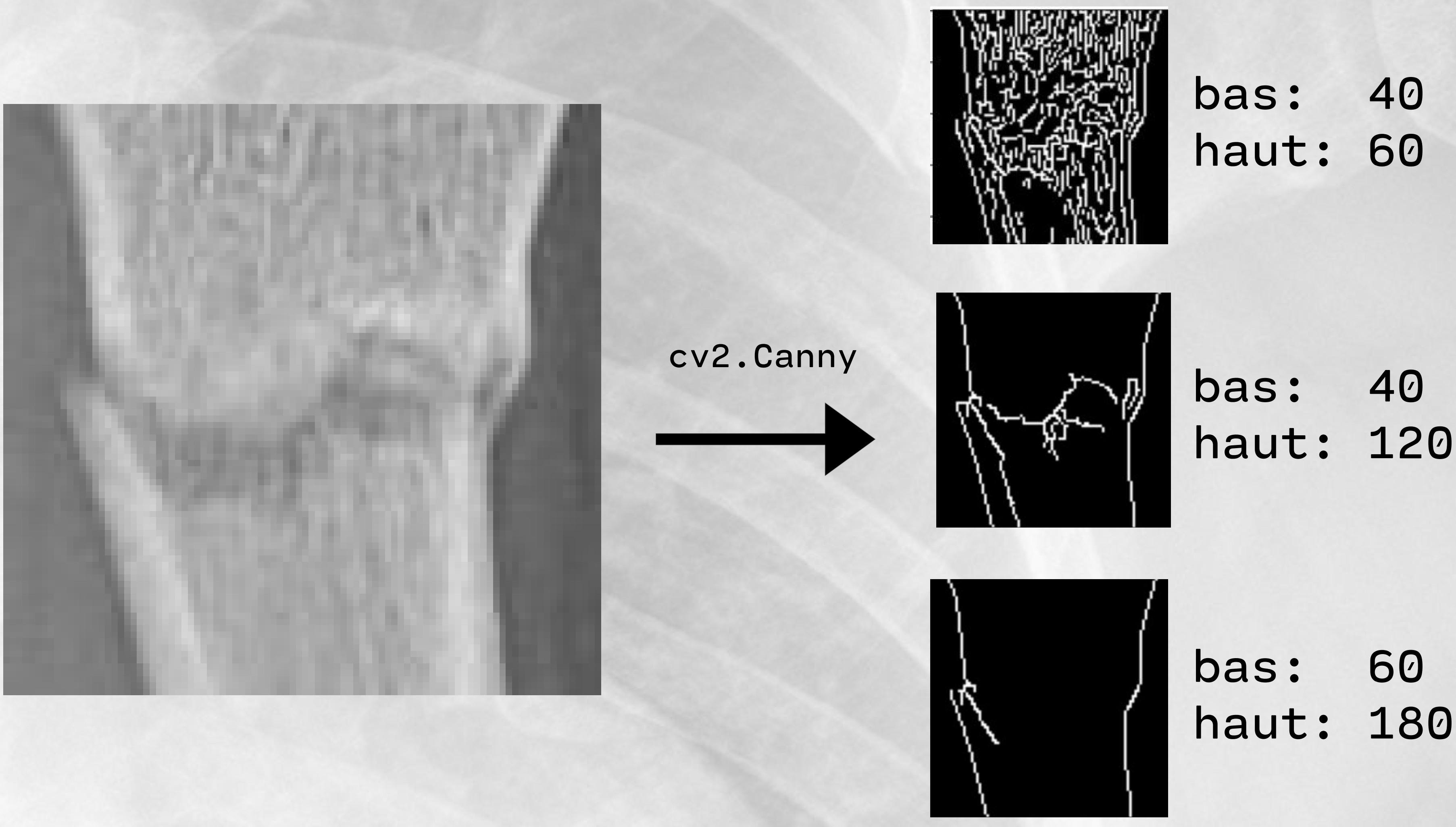
Identification du type de fracture



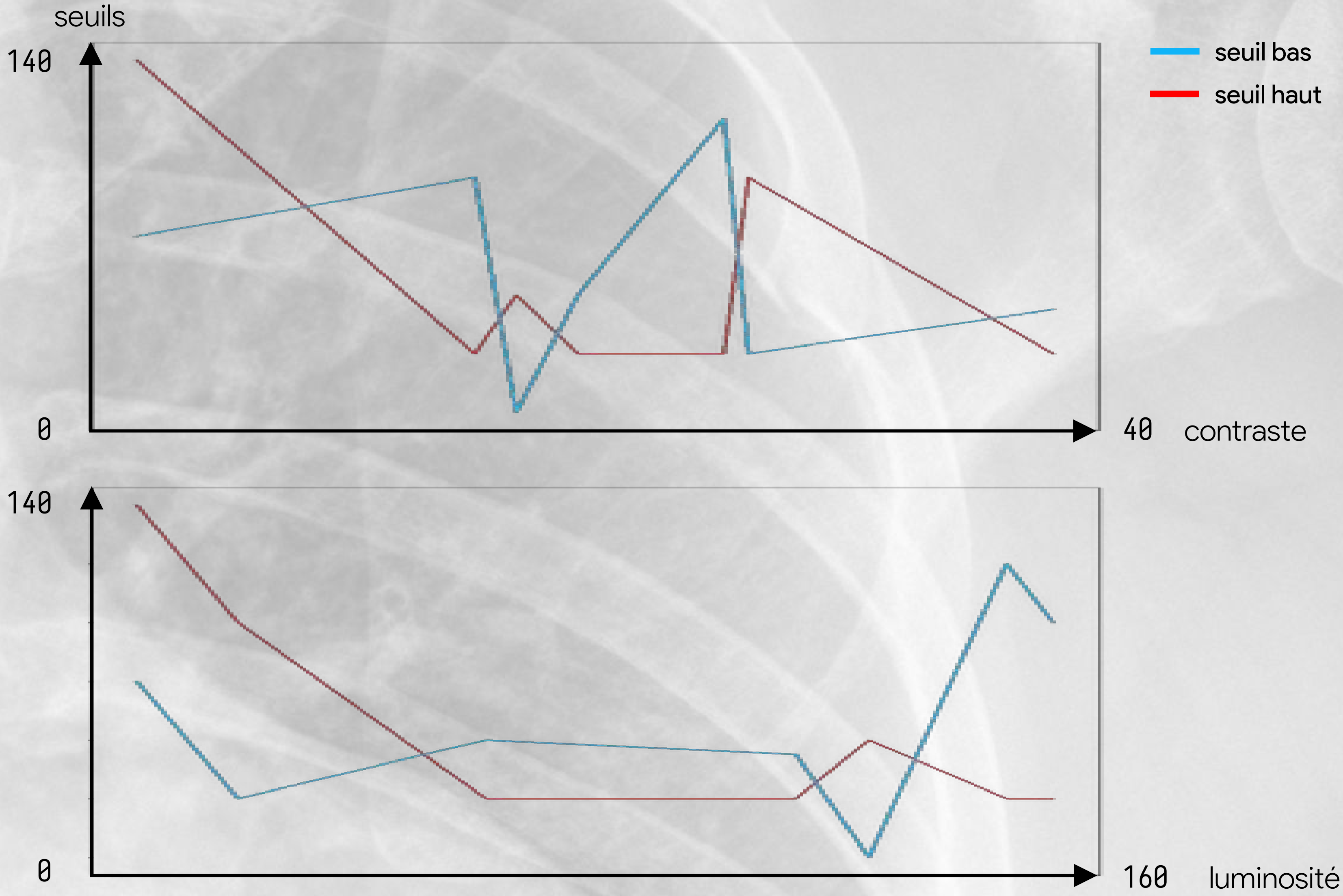
Noms des différentes *lignes de fracture* du fémur

Détection des contours

Un problème de texture

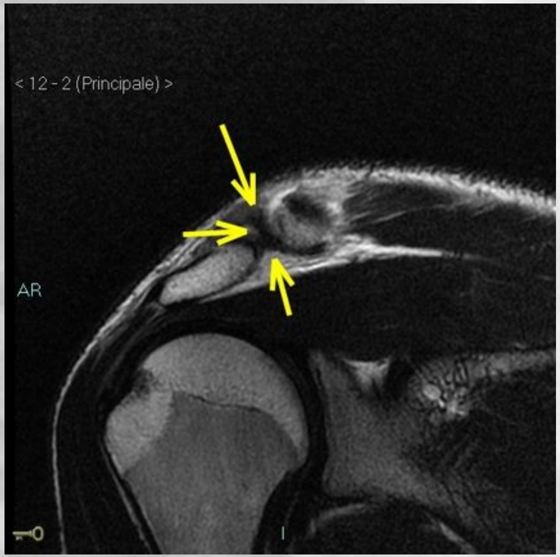


seuils(luminosité, contraste) ?



Seuils optimaux de détection de contours

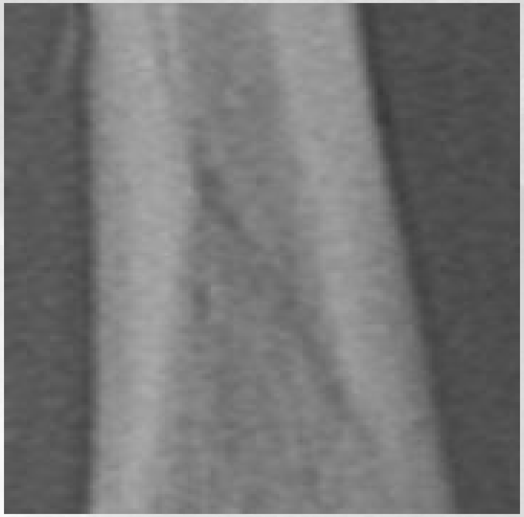
Recherche de sets de données



Recherche de sets de données

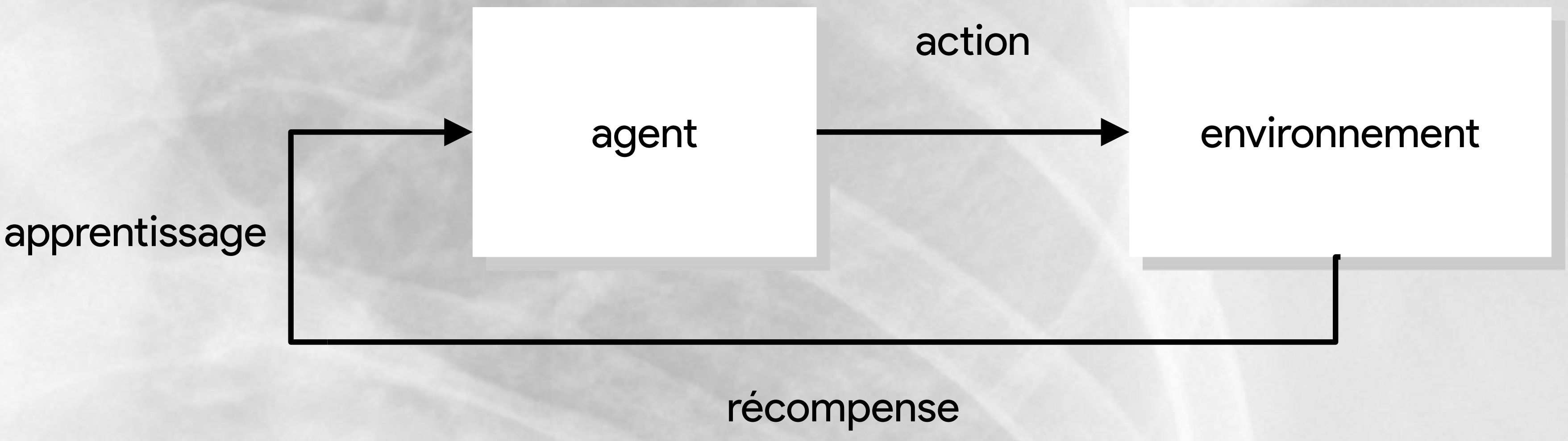
bas 40
haut 120

bas 27
haut 44

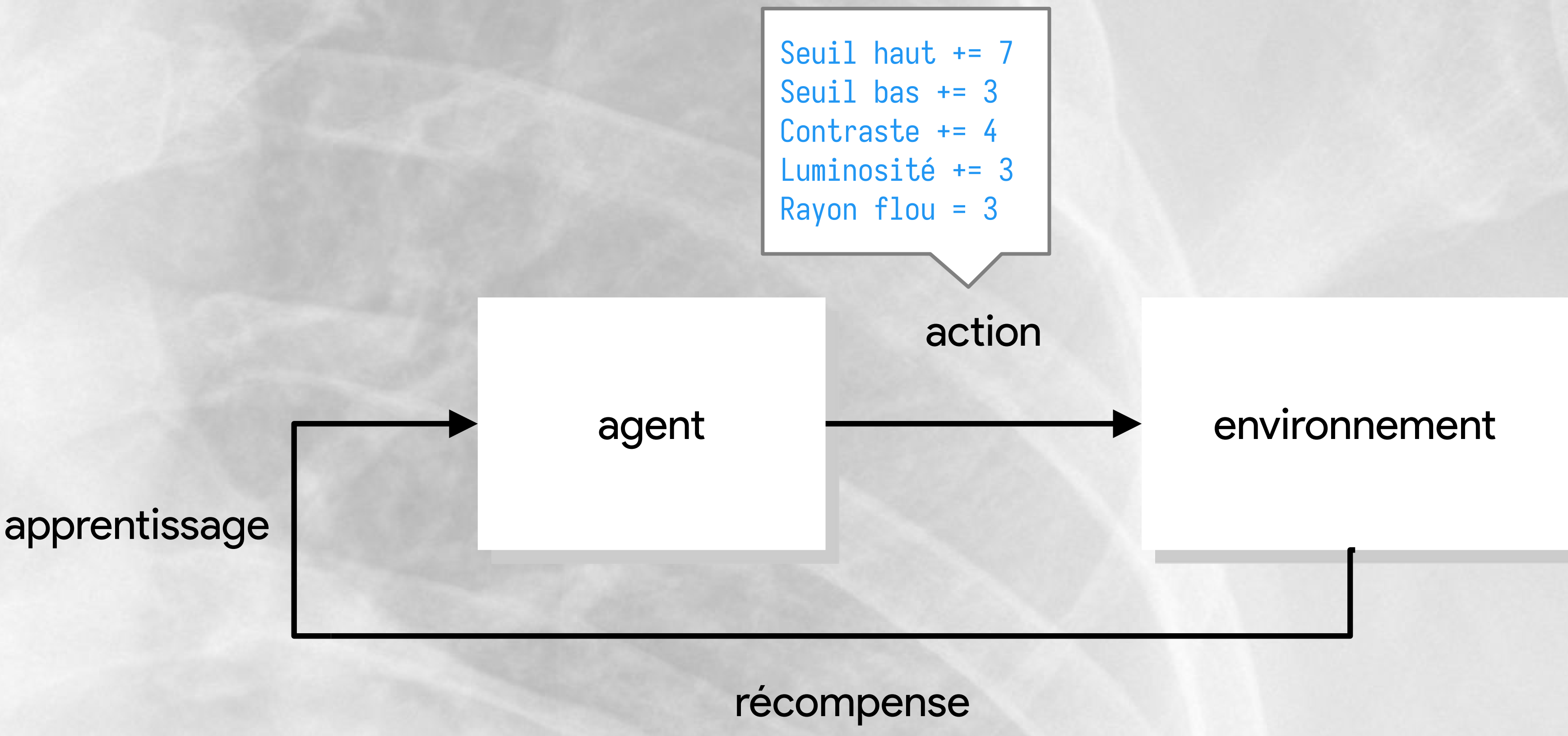


bas 20
haut 21

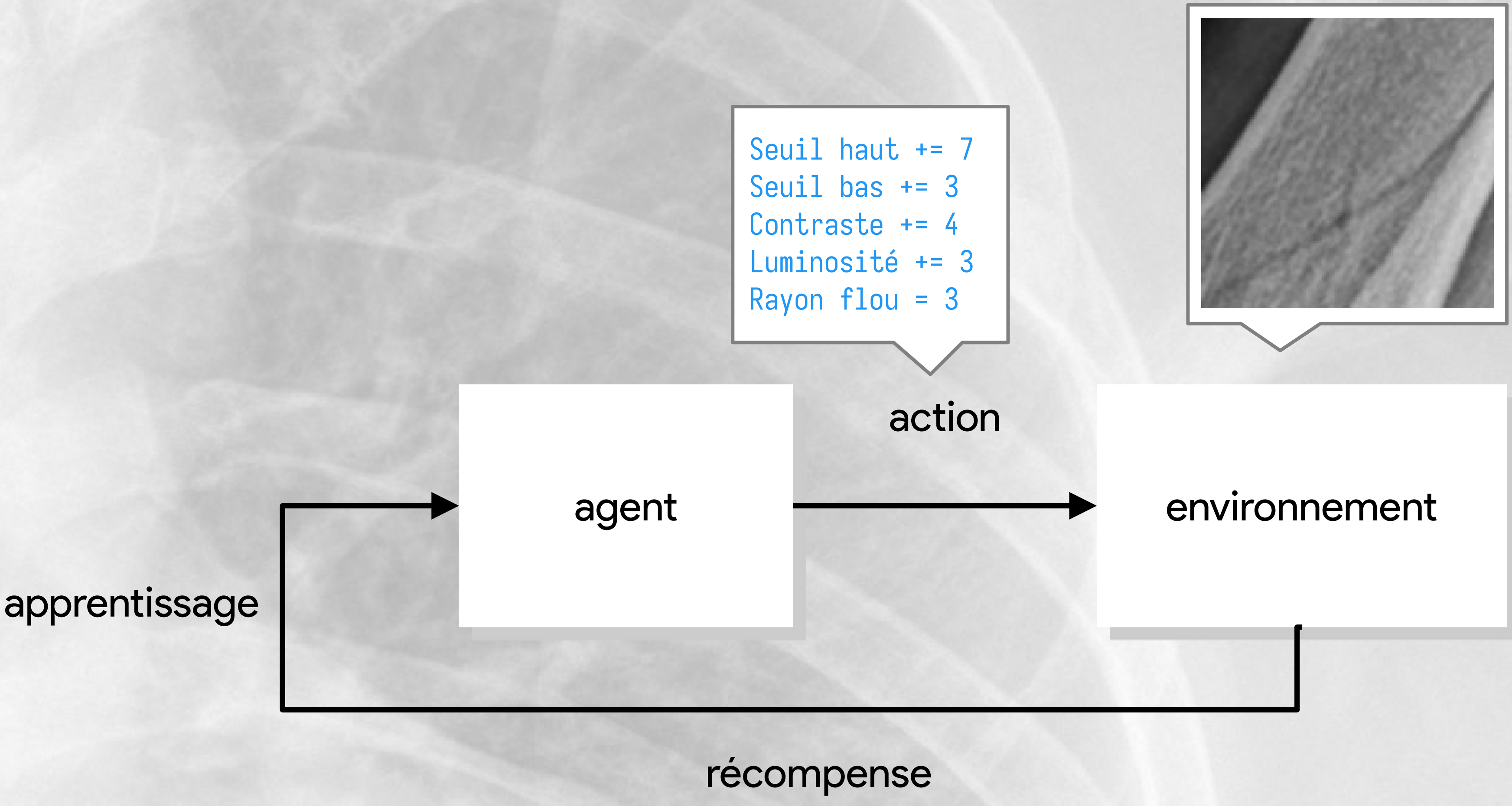
Apprentissage par renforcement



Apprentissage par renforcement



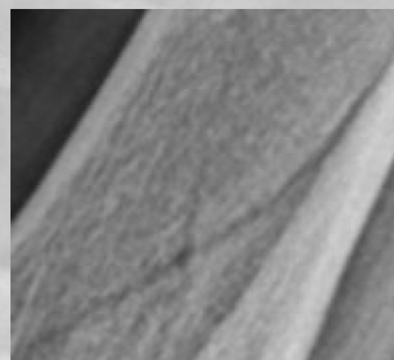
Apprentissage par renforcement



Apprentissage par renforcement

action

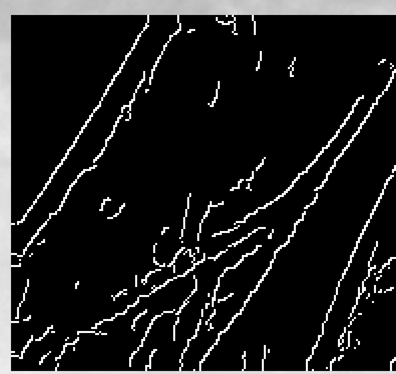
$\mathrm{cont}, \mathrm{lum}, r_{\mathrm{flou}}, \mathrm{seuil}_h, \mathrm{seuil}_b$



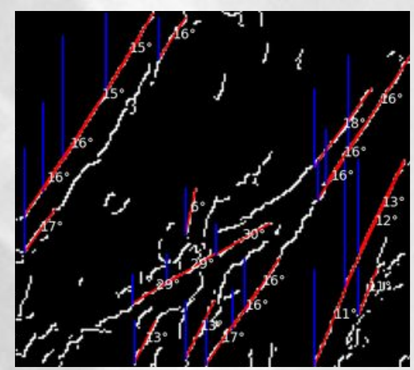
S

Environnement

$$T = \underset{r_{\mathrm{flou}}}{\mathrm{Flou}}\left(S \cdot \mathrm{cont} + \mathrm{lum}\right)$$

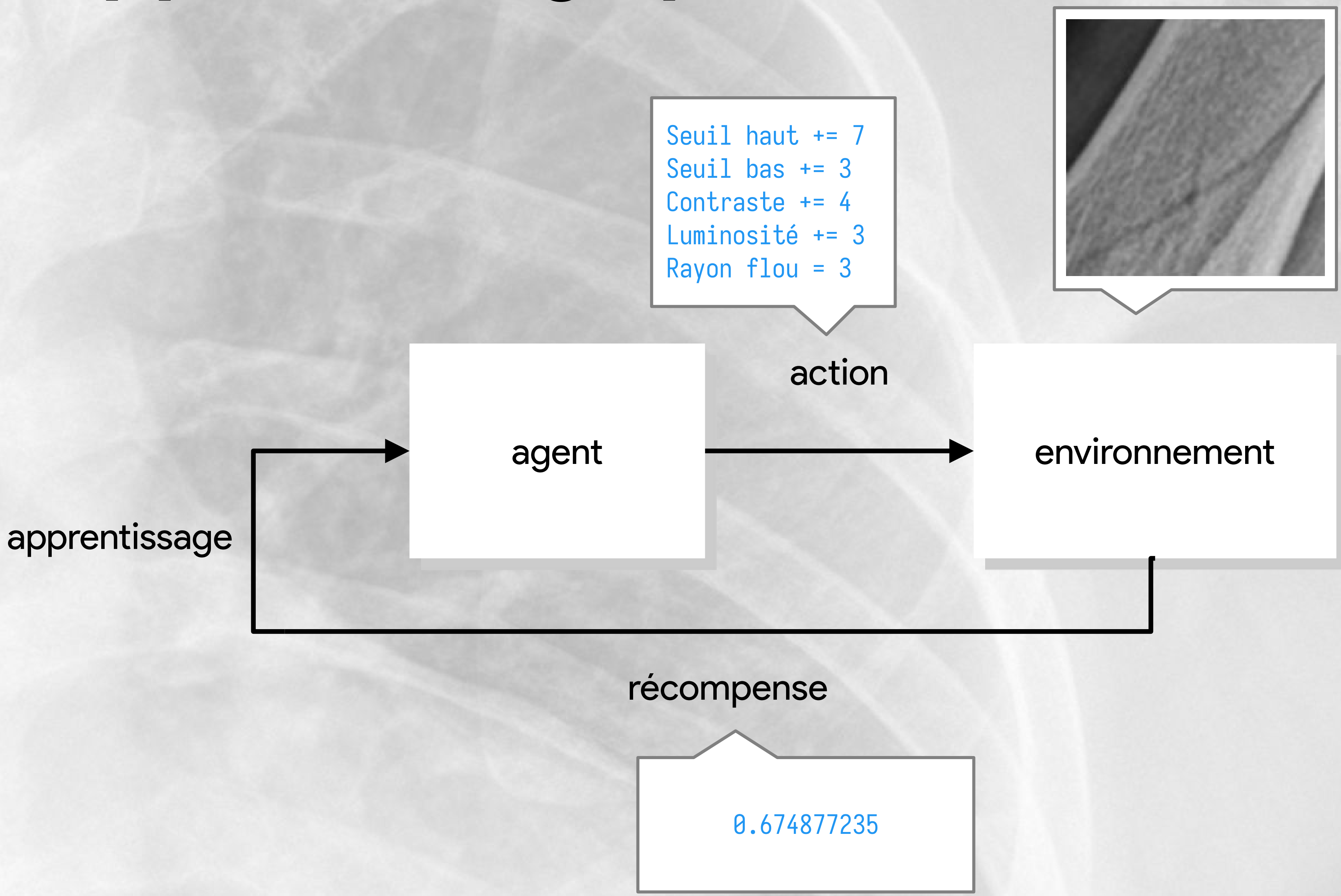


$$B = \underset{\mathrm{seuil}_h, \mathrm{seuil}_b}{\mathrm{Canny}} \; T$$



$$L = \{(i, f) \in \mathrm{Hough}(B),\ \|i - f\| \geq 20\}$$

Apprentissage par renforcement



Calcul de la récompense

$$\begin{cases} 1 - d(\operatorname{lum} B, 7, 15) & \text{si } d(\operatorname{lum} B, 7, 15) \neq 0 \\ 0.25 + 1 - d(|L|, 10, 25) & \text{sinon} \end{cases}$$

avec

$$d := (v, a, b) \mapsto \begin{cases} |v - a| & \text{si } v < a \\ |v - b| & \text{si } v > b \\ 0 & \text{sinon} \end{cases}$$

Apprentissage de l’agent
avec des *Q-Tables*

	État 1	État 2	État 3	…
Action 1	0.1244	0.3409	0.7574	0.7269
Action 2	0.8476	0.4427	0.3895	0.8374
Action 3	0.8479	0.7761	0.0762	0.7884
…	0.1121	0.4661	0.9433	0.1774

Apprentissage de l'agent

Le problème de dimension des *Q-Tables*

$$8^{200 \cdot 200}$$

$$20 \cdot 20 \cdot 6 \cdot 10 \cdot 5$$

	État 1	État 2	État 3	...
Action 1 `Seuil haut -= 10`	0.1244	0.3409	0.7574	0.7269
Action 2 `Seuil haut -= 9`	0.8476	0.4427	0.3895	0.8374
Action 3 `Seuil haut -= 8`	0.8479	0.7761	0.0762	0.7884
...	0.1121	0.4661	0.9433	0.1774

Apprentissage de l’agent
Le problème de dimension des *Q-Tables*

$$8^{200\cdot 200}\cdot 20\cdot 20\cdot 6\cdot 10\cdot 5\cdot 11 \quad \text{octets} \;\approx$$

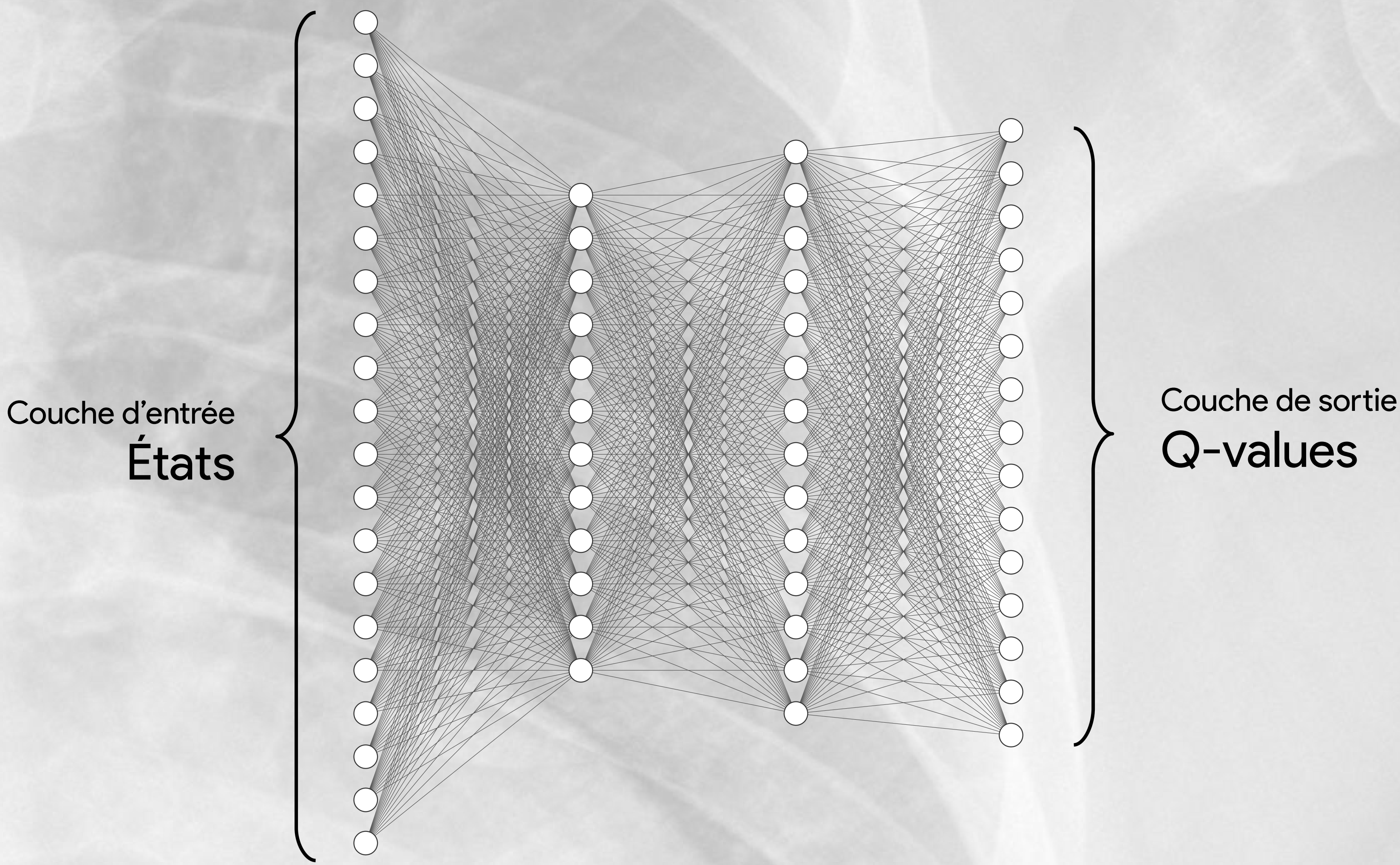
$$5.25\cdot 10^{36114}\ \text{Po}$$

Taille d’Internet (2014)

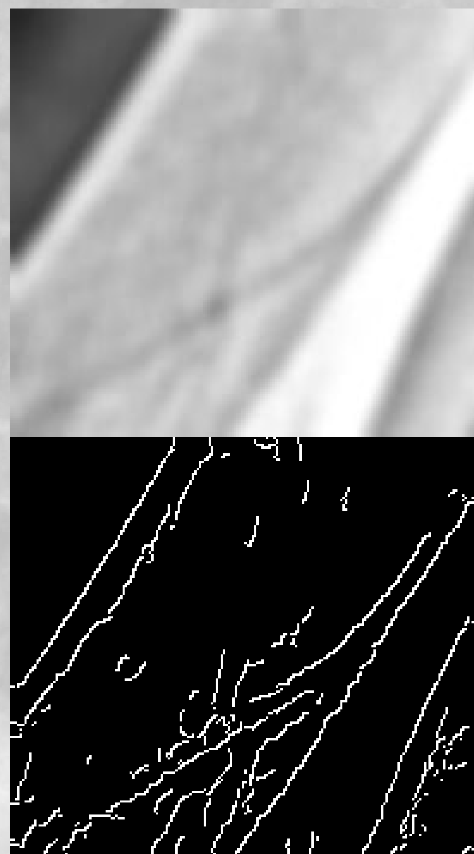
$$10^{9}\ \text{Po}$$

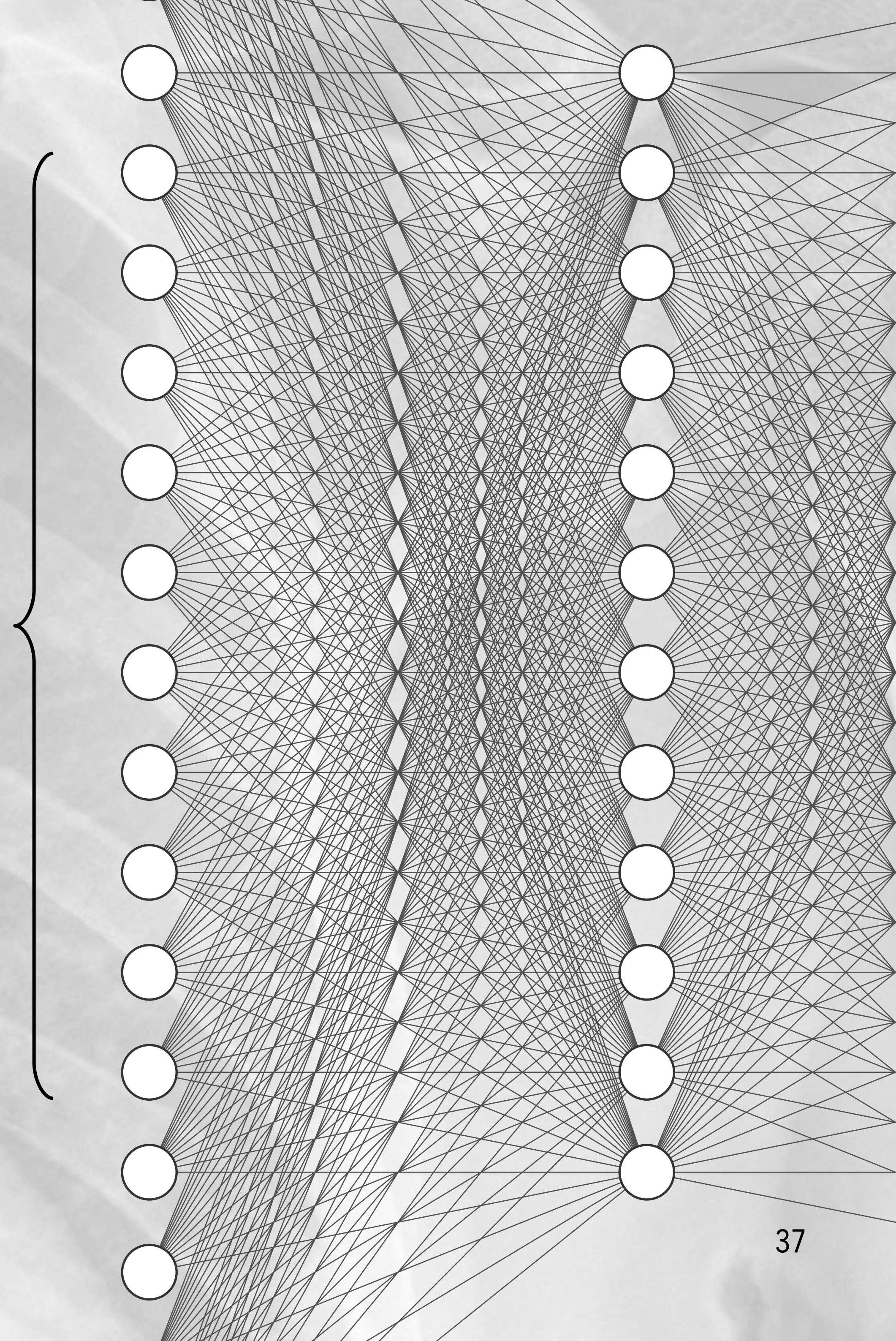
Apprentissage de l'agent

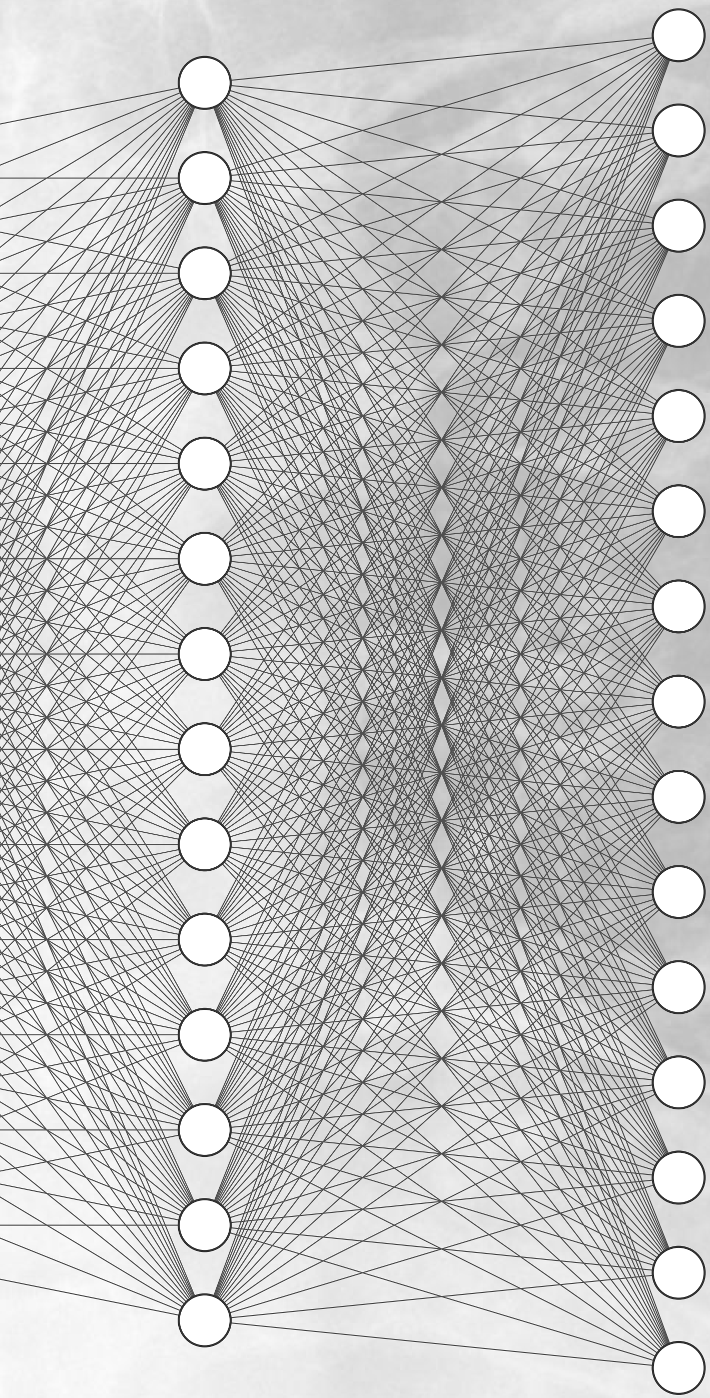
Avec des *Deep-Q Networks*



Un état







Incréments du seuil haut

$\text{seuil}_h \in \{-10, -9, \ldots, 8, 9, 10\}$

Incréments du seuil bas

$\text{seuil}_b \in \{-10, -9, \ldots, +8, +9, +10\}$

Incréments du contraste

$\text{cont} \in \{0, +0.1, +0.2, +0.3, +0.4, +0.5\}$

Incréments de la luminosité

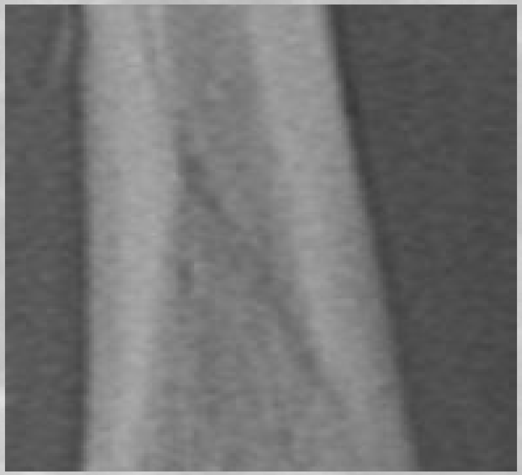
$\text{lum} \in \{-5, -4, \ldots, +3, +4, +5\}$

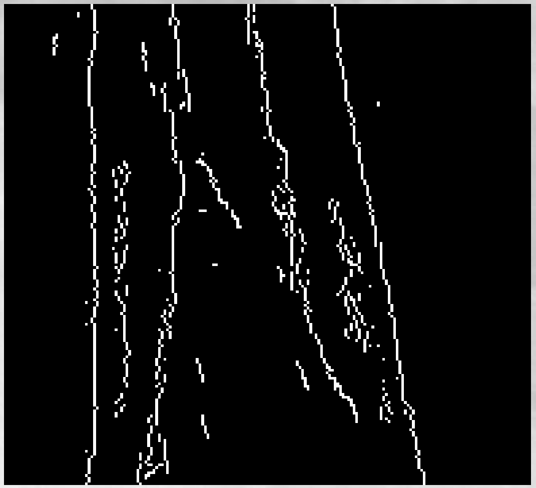
Réglage du rayon de flou

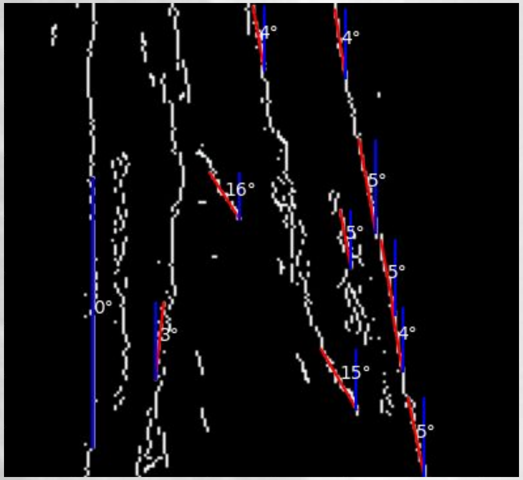
$r_{\text{flou}} \in \{0, 10, \ldots, 50\}$

TODO : ajouter des + pour les incréments partout

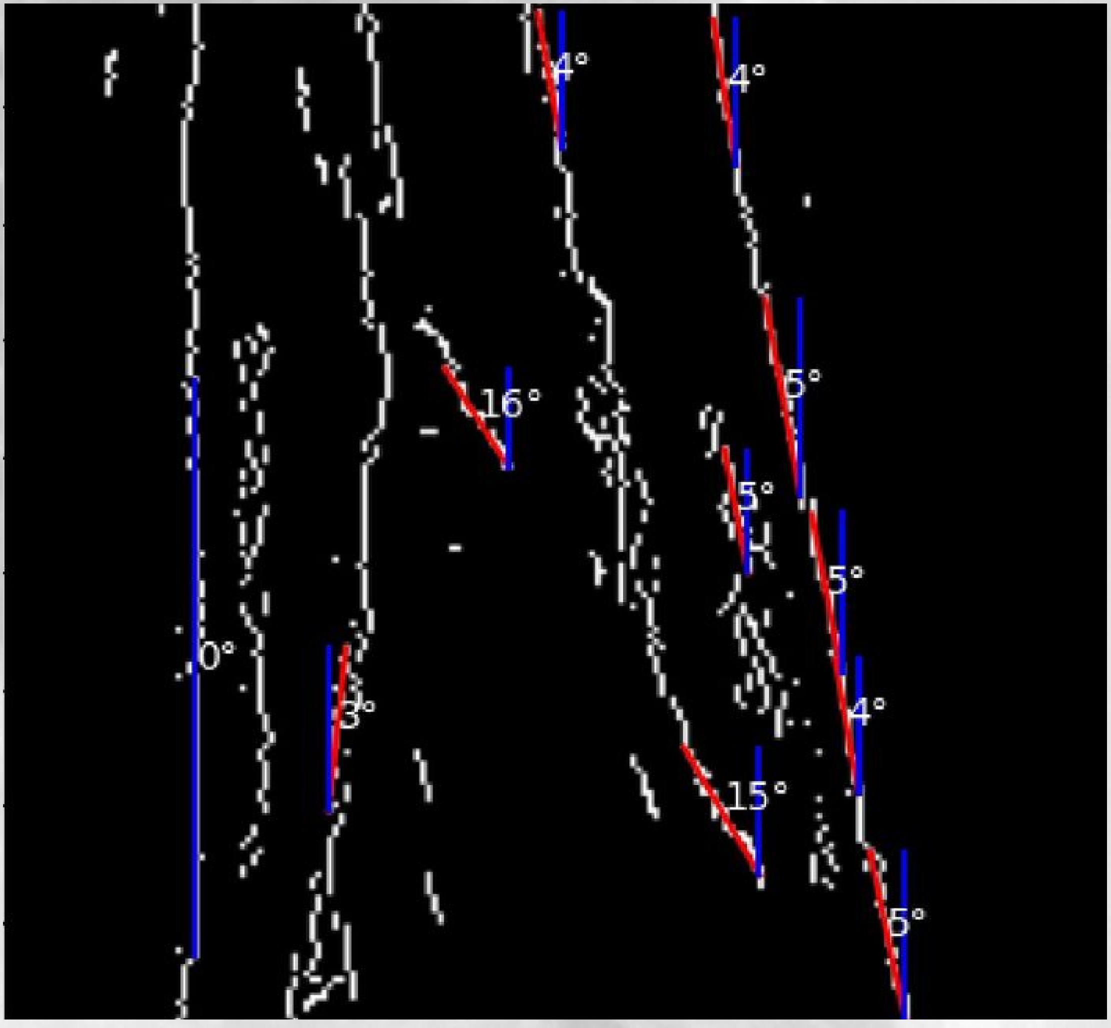
Résultats







Résultats



Annexe

detect.py
Traitements basiques et détection des bords

```python
import json
from pathlib import Path
from typing import Any, Optional, TypeVar

import cv2
import matplotlib.pyplot as plt
import numpy as np
from nptyping import NDArray
from rich.progress import Progress

from angles import display_lines, get_lines_probabilistic
from utils import *


def is_broken(angles: list[float], ε: float = 10) → bool:
    """
    If the maximum offset with a vertical angle is less than ε for all angles, the bone is not broken
    """
    τ = 2 * np.pi
    deg = lambda rad: rad / τ * 180
    print("[" + " ".join(f"{int(deg(angle))}°" for angle in angles) + "]")
    return max(map(deg, angles)) ≥ ε


def is_white(pixel: float) → bool:
    return pixel > 0.75


def center_of(image: np.ndarray) → np.ndarray:
    # return image[:, len(image[0])//4: -len(image[0])//4]
    return image


def contrast_of(image: np.ndarray) → float:
    if len(image.shape) == 2:
        return image.std()
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).std()


def boost_contrast(image: np.ndarray) → np.ndarray:
    return 4 * image


def grayscale_of(image: NDArray[Any, Any, 3]) → NDArray[Any, Any]:
    return cv2.cvtColor(image, cv2.COLOR_BGR2HSV)[:, :, 2]


def brightness_of(image: Union[NDArray[Any, Any, 3], NDArray[Any, Any]]) → float:
    # RGB
    if len(image.shape) == 3:
        image = grayscale_of(image)
    return mean(flatten_2D(image))


def detect_edges(
    image: Union[NDArray[Any, Any, 3], NDArray[Any, Any]],
    low: int,
    high: int,
    σ: int = 3,
    blur: float = 0,
) → tuple[NDArray[Any, Any, 3], NDArray[Any, Any]]:
    """
    Détecte les bords d'une image, en utilisant—si blur ≠ 0—un filtre bilatéral avec un σ_color =
σ_space = blur.
    """
    σ, low, high = map(int, (σ, low, high))

    if len(image.shape) == 2:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

    if blur:
        # image = cv2.bilateralFilter(image, d=5, sigmaColor=blur, sigmaSpace=blur)
        image = cv2.blur(image, (blur, blur))

    edges = cv2.Canny(image, low, high, apertureSize=σ, L2gradient=True)
    return image, edges


def save_figure(image_path: Path, save: Optional[Path] = None):
    image = cv2.imread(str(image_path))
    print(f"contrast is {contrast_of(image)}")
    original, edges = detect_edges(image, low=40, high=120, blur=3)
    lines = list(get_lines_probabilistic(center_of(edges), gap=5, length=20))
    if not lines:
        print(f"error: no lines detected for {image}")
        return
    broken = is_broken([angle for _, _, angle in lines])
    fig, ax = plt.subplots(1, 2, sharex=True, sharey=True)
    fig.suptitle(
        f"Détecté comme {'cassé' if broken else 'sain'}\n"
        f"cont: {contrast_of(image)} lum: {brightness_of(image)}\n"
        # f"tilt: {image_tilt(lines)/(2*np.pi)*180}° #segments: {len(lines)}\n"
        f"outlum: {brightness_of(center_of(edges))} lumratio:
{brightness_of(center_of(edges))/brightness_of(image)}"
    )
    ax[0].imshow(original)
    # ax[1].imshow(edges)
    display_lines(ax[1], center_of(edges), lines)

    if save:
        plt.savefig(str(save))
    else:
        plt.show()

    print(
        f'{image_path}: Detected as {"broken" if broken else "healthy"}',
        end="\n\n",
    )


if __name__ == "__main__":
    with Progress() as bar:
        files = list(Path("datasets/various").glob("*.png"))
        task = bar.add_task("[blue]Processing", total=len(files))
        for testfile in files:
            save_figure(testfile, save=Path("line-detection") / testfile.name)
            bar.advance(task)
```

angles.py
Détection des lignes et calcul des angles

```python
import pathlib
from typing import Iterable, Optional
from math import sqrt
import numpy as np

τ = 2 * np.pi

import matplotlib.pyplot as plt
from skimage.transform import (hough_line, hough_line_peaks,
                               probabilistic_hough_line)


def norm(a: tuple[int, int], b: tuple[int, int]) → float:
    return sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)

def get_lines_probabilistic(
    edges: np.ndarray, length: int = 5, gap: int = 3, minimum_length: float = 0
) → Iterable[tuple[tuple[int, int], tuple[int, int], float]]:
    """
    Return value:
    list of (start point, end point, angle with the vertical projection in
radians)
    """
    for beginning, end in probabilistic_hough_line(
        edges, threshold=10, line_length=length, line_gap=gap
    ):
        x0, y0 = beginning
        x1, y1 = end
        # CAH
        angle = np.arccos(abs(y1 - y0) / np.sqrt((x1 - x0) ** 2 + (y1 - y0) **
2))

        if norm(beginning, end) ≥ minimum_length:
            yield beginning, end, angle
```

```python
def unique_angles(ε: float, lines: Iterable[tuple[tuple[int, int], tuple[int,
int], float]]) → set[float]:
    """
    Return list of unique angles. Two angles are considered equal if they are
less than ε appart from each other.
    """
    angles = set()
    for _, _, angle1 in lines:
        for _, _, angle1 in lines:
            if abs(angle1 - angle2) > ε:
                angles.add(angle1)
    return angles

def display_lines(
    ax,
    image: np.ndarray,
    lines: list[tuple[tuple[int, int], tuple[int, int], float]],
    probabilistic: bool = True,
    save: Optional[str] = None,
):
    """
    Display lines on top of image with matplotlib
    """
    plt.imshow(image, cmap="gray")
    midway = lambda p1, p2: ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)
    if probabilistic:
        counter = 0
        # for beginning, end, angle in lines[5:6]:
        for (x0, y0), (x1, y1), angle in lines:
            ax.plot([x0, x1], [y0, y1], color="red")
            ax.plot([x0, x0], [y0, y1], color="blue")
            ax.text(
                *midway((x0, y0), (x1, y1)), f"{int(angle*180/tau)}°",
color="white"
            )
            counter += 1
    else:
        for *point, angle in lines:
            ax.axline(point, slope=angle)
            ax.scatter(*point)

    ax.set_xlim(0, image.shape[1])
    ax.set_ylim(image.shape[0], 0)
    if save:
        plt.savefig(save)
```

utils.py

Fonctions diverses utiles à l'ensemble du programme

```python
from math import sqrt
from abc import ABCMeta, abstractmethod
from typing import Iterable, Union, TypeVar, Callable, Any

def mean(o: Iterable[Union[float, int]]) -> float:
    values = list(o)
    return sum(values) / len(values)


def flatten_2D(o: Iterable):
    flat = []
    for row in o:
        for item in row:
            flat.append(item)
    return flat


# needed to remove pylint(cell-var-from-loop), see
https://stackoverflow.com/a/67928238/9943464
def access(o, key):
    return o.get(key)


def norm(vector):
    return sqrt(sum(map(lambda x: x ** 2, vector)))


def roughly_equals(ε=0.01) -> Callable[..., bool]:
    def _(a, *b):
        return any(abs(a - b_) < ε for b_ in b)

    return _

T = TypeVar("T", bound=Union[float, int])
def clip(minimum: T, maximum: T, o: T) -> T:
    return max(minimum, min(maximum, o))
```

```python
T = TypeVar("T")
def partition(o: Iterable[T], layout: Union[list[int], tuple[int]]) ->
list[list[T]]:
    """
    Chunks o into chunks of sizes given by layout.

    >>> partition([1, 2, 3, 4, 5, 6, 7, 8, 9], (3, 3, 3))
    [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

    >>> partition([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], (4, 4, 4))
    [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

    >>> partition([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], (6, 2, 4))
    [[1, 2, 3, 4, 5, 6], [7, 8], [9, 10, 11, 12]]
    """
    # can't use [[]] * len(layout) as all sublists will be views of the
same sublist
    partitions = []
    for _ in layout:
        partitions.append([])

    index_in_partition = 0
    current_partition = 0

    for item in o:
        partitions[current_partition].append(item)
        if index_in_partition == layout[current_partition] - 1:
            current_partition += 1
            index_in_partition = 0
        else:
            index_in_partition += 1

    return partitions
```

rl_environment.py 1/2

Environnement pour l'apprentissage par renforcement

```python
from datetime import datetime
import matplotlib.pyplot as plt
import random
import inspect
import json
from pathlib import Path
from typing import *
import textwrap

import cv2
import gym
import numpy as np
from gym import spaces
from nptyping import NDArray
from numpy import array
from rich import print
from angles import get_lines_probabilistic, unique_angles, display_lines

from detect import brightness_of, contrast_of, detect_edges, grayscale_of
from utils import roughly_equals, clip


class EdgeDetectionEnv(gym.Env):
    metadata = {"render_modes": ["human", "rgb_array"], "render_fps": 4}

    def pick_from_dataset(self) -> NDArray:
        self.current_image_name = random.choice(self.dataset)
        print(f"picking {self.current_image_name}")
        return self.preprocess(cv2.imread(self.current_image_name))

    @property
    def saw_everything(self) -> bool:
        return len(self.seen_images) == len(self.dataset)

    @property
    def dataset_size(self) -> int:
        return len(self.dataset)

    def preprocess(self, image: NDArray) -> NDArray:
        """
        Resize the image to the biggest dimensions of the dataset
        """
        width, height = image.shape[:2]
        if width == self.image_dimensions[0] and height ==
self.image_dimensions[1]:
            return image

        print(f"preprocess: resizing: {width}x{height} ->
{self.image_dimensions}")
        return cv2.resize(image, self.image_dimensions)

    def biggest_dimensions(self, dataset: list[str]) -> Tuple[int, int]:
        biggest_width = biggest_height = 0
        for image in dataset:
            width, height = cv2.imread(image).shape[:2]
            if width > biggest_width:
                biggest_width = width
            if height > biggest_height:
                biggest_height = height
        return biggest_width, biggest_height

    def done(self) -> bool:
        return self.steps_count_for_current_image >
self.max_steps_for_single_image or (int(brightness_of(self.edges)) in
range(*self.acceptable_brightness_range) and self.segments_count in
range(*self.acceptable_segments_count_range))

    def save_settings(self, agent_name: str, into: Path):
        # Used to encode int64 and other numpy number types
        def numpy_encoder(object):
            if isinstance(object, np.generic):
                return object.item()

        assert self.current_image_name is not None
        save_as = into / agent_name / Path(self.current_image_name).stem
        save_as.parent.mkdir(parents=True, exist_ok=True)
        Path(f"{save_as}--info.json").write_text(json.dumps(self.info,
default=numpy_encoder, indent=2))
        cv2.imwrite(f"{save_as}--source.png", self.source)
        cv2.imwrite(f"{save_as}--edges.png", self.edges)
        cv2.imwrite(f"{save_as}--original-source.png", self.original_source)
        _, ax = plt.subplots()
        display_lines(ax, self.edges, self.segments, probabilistic=True,
save=f"{save_as}--lines.png")

    @property
    def action_space_shape(self) -> int:
        return sum(v[0] for v in self.action_space_layout.values())

    def __init__(
        self,
        render_mode: Union[str, None],
        acceptable_brightness_range: Tuple[int, int],
        acceptable_segments_count_range: Tuple[int, int],
        dataset: Path,
        max_thresholds_increment: int = 5,
        max_brightness_increment: int = 3,
        max_blur_value: int = 30,
        step_blur_value: int =1,
        max_steps_for_single_image: int = 10_000,
    ):
        assert render_mode is None or render_mode in
self.metadata["render_modes"]

        self.dataset = [str(f) for f in dataset.iterdir() if f.is_file()]
        self.seen_images = set()
        self.unique_segment_angles = set()
        self.current_image_name = None
        self.thresholds = [100, 100]
        self.brightness_boost = 0
        self.segments = []
        self.blur = 0
        self.max_steps_for_single_image = max_steps_for_single_image
        self.segments_count = None
        self.contrast_multiplier = 1
        self.acceptable_brightness_range = acceptable_brightness_range
        self.acceptable_segments_count_range = acceptable_segments_count_range
        self.image_dimensions = self.biggest_dimensions(self.dataset)
        self.max_increment = max_thresholds_increment
        self.max_contrast_increment = 1
        self.max_brightness_increment = max_brightness_increment
        self.max_blur_value = max_blur_value // step_blur_value
        self.step_blur_value = step_blur_value
        self.steps_count_for_current_image = 0
        self.last_winning_edges = array([])
        self.last_winning_thresholds = [None, None]

        pixels_space = lambda width, height: spaces.Box(low=array([0, 0]),
high=array([width, height]), dtype=np.int16)

        # we stick the two images horizontally instead of adding a third
dimension (2*width, height) instead of (2, width, height)
        self.observation_space_shape = (
self.image_dimensions[0],
            self.image_dimensions[1],
        )
        # self.observation_space.shape gives (2,) instead of (width, height)
        self.observation_space = pixels_space(*self.observation_space_shape)

        # key: [size, offset]
        self.action_space_layout = {
            "high_threshold": [2 * max_thresholds_increment, -
max_thresholds_increment],
            "low_threshold": [2 * max_thresholds_increment, -
max_thresholds_increment],
            "contrast": [2*self.max_contrast_increment, -
self.max_contrast_increment],
            "brightness": [
2 * self.max_brightness_increment,
                -self.max_brightness_increment,
            ],
            "blur": [self.max_blur_value, 0],
        }

        self.action_space = spaces.Dict(
            {k: spaces.Discrete(size, start=offset) for k, (size, offset) in
self.action_space_layout.items()}
        )

        print(f"Initialzed action space with layout {self.action_space_layout}")

        if render_mode == "human":
            import pygame

            pygame.init()
            pygame.display.init()
            self.window = pygame.display.set_mode(self.image_dimensions)
            self.clock = pygame.time.Clock()

        # self.renderer = Renderer(render_mode, self._render_frame)

    @property
    def observation(self) -> NDArray:
        return array([self.source,
self.edges]).reshape(*self.observation_space_shape)

    def reward(self, brightness: float) -> float:
        lo, hi = self.acceptable_brightness_range

        if brightness in self.acceptable_brightness_range:
            return 1

        offset = abs(brightness - (lo if brightness < lo else hi))
        width = abs((0 - lo) if brightness < lo else (255 - hi))

        reward = 1 - (offset / width)

        if reward == 1 and self.segments_count is not None:
            lo, hi = self.acceptable_segments_count_range
            offset = abs(self.segments_count - (lo if self.segments_count < lo
else hi))
            # en supposant segments count ∈ [0, 10_000[
            width = abs((0 - lo) if self.segments_count < lo else (10_000 - hi))
            return clip(0, 1, 0.25 + (1 - offset / width))

        return reward
```

rl_environment.py 2/2

Environnement pour l'apprentissage par renforcement

```
    @property
    def info(self) -> Dict[str, Any]:
        return {
            "at": f"{datetime.now():%Y-%m-%dT%H:%M:%S}",
            "source": {
                "brightness": brightness_of(self.source),
                "contrast": contrast_of(self.source),
                # "original": self.original_source,
                "name": self.current_image_name,
            },
            "edges": {
                "brightness": brightness_of(self.edges),
                "contrast": contrast_of(self.edges),
            },
            "segments": {
                "count": self.segments_count,
                "angles": list(self.unique_segment_angles),
            },
            "settings": {
                "high_threshold": self.thresholds[0],
                "low_threshold": self.thresholds[1],
                "contrast_multiplier": self.contrast_multiplier,
                "brightness_boost": self.brightness_boost,
                "bilateral_blur_sigmas": self.blur,
            },
        }

    def reset(self, seed=None, return_info=False):
        super().reset(seed=seed)
        # Pick a random bone radio image from the set
        self.source, self.edges = detect_edges(self.pick_from_dataset(), low=seed
or 50, high=seed or 50)
        self.source = grayscale_of(self.source)
        self.original_source = self.source.copy()
        if self.steps_count_for_current_image <= self.max_steps_for_single_image:
            self.seen_images.add(self.current_image_name)
        self.steps_count_for_current_image = 0

        return (self.observation, self.info) if return_info else self.observation

    def step(self, action: OrderedDict, ε):
        print(f"with {dict(**action)}", end=" ")

        self.thresholds[0] = clip(20, 150, self.thresholds[0] +
action["high_threshold"])
        self.steps_count_for_current_image += 1
        self.thresholds[1] = clip(20, 150, self.thresholds[1] +
action["low_threshold"])
        self.blur = action["blur"]
        self.ε = ε
        self.contrast_multiplier = 1 + clip(0, 5, self.contrast_multiplier*10 - 1
+ action["contrast"]) / 10
        self.brightness_boost = clip(0, 30, self.brightness_boost +
action["brightness"])
        self.source = np.clip(
            self.original_source.astype("int16") * self.contrast_multiplier +
self.brightness_boost,
            # self.source.astype("int16") + action["brightness"],


        ).astype("uint8")
        blurred_source, self.edges = detect_edges(self.source, *self.thresholds,
blur=self.blur * self.step_blur_value)
        self.source = grayscale_of(blurred_source)
        edges_brightness = brightness_of(self.edges)

        if roughly_equals(0.001)(edges_brightness, 0, 255):
            print("pullup", end=" ")
            self.source = self.original_source.copy()
            self.contrast_multiplier = 1
            self.brightness_boost = 0
            return (self.observation, -1, False, self.info)

        self.segments = list(get_lines_probabilistic(self.edges,
minimum_length=20))
        self.segments_count = len(self.segments)
        # 50 mrad ≈ 3°
        self.unique_segment_angles = unique_angles(50e-3, self.segments)

        print(f"bright {edges_brightness}, #seg {self.segments_count}", end=" =>
")


        if (done := self.done()):
            self.last_winning_edges = self.edges.copy()
            self.last_winning_thresholds = self.thresholds.copy()

        return (
            self.observation,
            self.reward(edges_brightness),
            done,
            self.info,
        )

    def render(self, window):
        import pygame

        window.fill((255, 255, 255))
        self._draw_image(self.original_source, window, (0, 0))
        self._draw_text("original", window, 0, self.image_dimensions[1] + 20)
        self._draw_image(self.source, window, (self.image_dimensions[0], 0))
        self._draw_text(f"original * {self.contrast_multiplier} +
{self.brightness_boost}\nblur {self.blur * self.step_blur_value}", window,
self.image_dimensions[0], self.image_dimensions[1] + 20)
        self._draw_image(self.edges, window, (self.image_dimensions[0]*2, 0))
        self._draw_text(
            f"""
            thresh lo {self.thresholds[0]} hi {self.thresholds[1]}
            bright {brightness_of(self.edges):.2f}
            segments count {self.segments_count}
            """, window, self.image_dimensions[0]*2, self.image_dimensions[1] +
20)
        self._draw_image(self.last_winning_edges, window,
(self.image_dimensions[0]*3, 0))
        self._draw_text(
            f"""
            tresh were lo {self.last_winning_thresholds[0]} hi
{self.last_winning_thresholds[1]}
            in {self.acceptable_brightness_range}
            in {self.acceptable_segments_count_range}
            """,
            window,
            self.image_dimensions[0]*3,
            self.image_dimensions[1] + 20,
        )

        self._draw_text(
            f"""
            {self.current_image_name}
            {self.ε*100:.1f}% eXploration {(1-self.ε)*100:.1f}% Exploitation
            """,
            window,
            int(self.image_dimensions[0]*1.5),
            self.image_dimensions[1] + 75,
        )
        pygame.display.update()

    def _draw_text(self, text, window, x, y, width=None):
        import pygame

        text = inspect.cleandoc(text)
        if width is not None:
            text = textwrap.fill(text, width=width)
        pygame.init()
        font = pygame.font.SysFont("monospace", 12)
        for i, line in enumerate(text.splitlines()):
            text_surface = font.render(line, True, (0, 0, 0))
            window.blit(text_surface, (x, y + i * 12))

    def _draw_image(self, image, window, *at):
        import pygame

        if len(image.shape) < 2:
            return
        size = image.shape[1::-1]
        cv2_image = np.repeat(image.reshape(size[1], size[0], 1), 3, axis=2)
        surface = pygame.image.frombuffer(cv2_image.flatten(), size, "RGB")
        surface = surface.convert()
        window.blit(surface, at)

    def close(self):
        if self.window is not None:
            import pygame

            pygame.display.quit()
            pygame.quit()
```

rl_agent.py

Agent pour l'apprentissage par renforcement

```python
import random
from collections import deque
from pathlib import Path
from datetime import datetime
import numpy as np
from numpy import array
from tensorflow.keras.layers import (
    BatchNormalization,
    Conv2D,
    Conv3D,
    Dense,
    Dropout,
    Flatten,
    Input,
    MaxPooling2D,
)
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from rich import print
from rl_environment import EdgeDetectionEnv
from utils import partition

REPLAY_MEMORY_SIZE = 1_000
MIN_REPLAY_MEMORY_SIZE = 333

class NeuralNetwork:
    # Creates a convolutional block given (filters) number of filters, (dropout) dropout rate,
    # (bn) a boolean variable indecating the use of BatchNormalization,
    # (pool) a boolean variable indecating the use of MaxPooling2D
    def conv_block(self, inp, filters=64, bn=True, pool=True, dropout=0.2):
        _ = Conv2D(filters=filters, kernel_size=3, activation="relu")(inp)
        if bn:
            _ = BatchNormalization()(_)
        if pool:
            _ = MaxPooling2D(pool_size=(2, 2))(_)
        if dropout > 0:
            _ = Dropout(0.2)(_)
        return _

    def __init__(self, conv_list, dense_list, input_shape, dense_shape):
        print(f"neural: init: conv: {conv_list}; dense: {dense_list}; in: {input_shape}; out: {dense_shape}")
        # Defines the input layer with shape = ENVIRONMENT_SHAPE
        input_layer = Input(shape=(*input_shape, 1))
        print(input_shape, input_layer.shape)
        # Defines the first convolutional block:
        print(f"Constructing block #1")
        _ = self.conv_block(input_layer, filters=conv_list[0], bn=False, pool=False)
        # If number of convolutional layers is 2 or more, use a loop to create them.
        if len(conv_list) > 1:
            for i, c in enumerate(conv_list[1:]):
                print(f"Constructing block #{i + 2}")
                _ = self.conv_block(_, filters=c)
        # Flatten the output of the last convolutional layer.
        _ = Flatten()(_)

        # Creating the dense layers:
        for d in dense_list:
            _ = Dense(units=d, activation="relu")(_)
        # The output layer has 4 nodes (one node per action)
        output = Dense(units=dense_shape, activation="linear", name="output")(_)

        # Put it all together:
        self.model = Model(inputs=input_layer, outputs=[output])
        self.model.compile(
            optimizer=Adam(lr=0.001),
            loss={"output": "mse"},
            metrics={"output": "accuracy"},
        )


class EdgeDetectionAgent:
    ACTION_NAMES = ["high_threshold", "low_threshold", "contrast", "brightness", "blur"]

    def __init__(
        self,
        name,
        env: EdgeDetectionEnv,
        conv_list,
        dense_list,
        memory_sample_size,
        discount_rate,
        update_target_model_every,
    ) -> None:
        self.env = env
        self.conv_list = conv_list
        self.dense_list = dense_list
        self.memory_sample_size = memory_sample_size
        self.last_save = None
        self.discount_rate = discount_rate
        self.update_target_model_every = update_target_model_every
        self.name = f"{name}_conv:{'+'.join(map(str, conv_list))}_dense:{'+'.join(map(str, dense_list))}_mem:{memory_sample_size}_y:{discount_rate}"

        print(env.observation_space_shape)

        self.model = NeuralNetwork(
            conv_list,
            dense_list,
            input_shape=env.observation_space_shape,
            dense_shape=env.action_space_shape,
        ).model
        self.target_model = NeuralNetwork(
            conv_list,
            dense_list,
            input_shape=env.observation_space_shape,
            dense_shape=env.action_space_shape,
        ).model
        self.target_model.set_weights(self.model.get_weights())
        self.replay_memory = deque(maxlen=REPLAY_MEMORY_SIZE)
        self.current_step_count = 0
        print(f"agent {self.name} initialized")

    def save_model(self, inside: Path):
        now = datetime.now()
        self.last_save = now
        timestamp = now.strftime("%Y-%m-%dT%H_%M_%S")
        self.model.save(inside / timestamp / "model.h5")
        self.target_model.save(inside / timestamp/ "target_model.h5")

    def remember(self, transition):
        self.replay_memory.append(transition)

    def train(self, terminal_state, step):
        if len(self.replay_memory) < max(self.memory_sample_size, MIN_REPLAY_MEMORY_SIZE):
            return

        print("sample", end=" ")
        memory_sample = random.sample(self.replay_memory, self.memory_sample_size)
        current_states = array([end for _, _, _, end, _ in memory_sample])
        future_states = array([start for start, _, _, _, _ in memory_sample])
        print("predict", end=" ")
        current_q_values = self.model.predict(current_states.reshape(-1, *self.env.observation_space_shape))
        future_q_values = self.target_model.predict(future_states.reshape(-1, *self.env.observation_space_shape))
        training_states = []
        training_q_values = []

        print(f"apply memory", end=" ")
        for index, (current_state, action, reward, future_state, done) in enumerate(memory_sample):
            for action_name, action_idx in self.neural_indices_of(action).items():
                new_q = reward + (
                    self.discount_rate * np.max(self.q_values_of_action(future_q_values[index], action_name))
                    if not done
                    else 0
                )
                try:
                    current_q_values[index, action_idx] = new_q
                except IndexError as e:
                    print(f"Tried indexing {index, action_idx} ({action_name}+={action[action_name]}) in {current_q_values.shape} Q-values")
                    raise e

            training_states.append(current_state)
            training_q_values.append(current_q_values[index])

        print(f"fit", end=" ")
        self.model.fit(
            x=array(training_states).reshape(-1, *self.env.observation_space_shape),
            y=array(training_q_values),
            batch_size=self.memory_sample_size,
            verbose=0,
            shuffle=False,
            callbacks=[],
        )

        if terminal_state:
            self.current_step_count += 1

        print("set weights")
        if self.current_step_count % self.update_target_model_every == 0:
            self.target_model.set_weights(self.model.get_weights())

    def get_q_values(self, state):
        return self.model.predict(state.reshape(-1, *self.env.observation_space_shape))

    def what_do_you_want_to_do(self, state):
        q_values = self.get_q_values(state).flatten()
        keys = self.ACTION_NAMES
        sizes = [self.env.action_space_layout[k][0] for k in keys]
        offsets = [self.env.action_space_layout[k][1] for k in keys]
        return {
            keys[i]: np.argmax(q_values_for_key) + offsets[i]
            for i, q_values_for_key in enumerate(partition(q_values, sizes))
        }

    def neural_indices_of(self, action: dict) -> dict[str, int]:
        """
        Returns a map of action names to the index of their values in the neural network's output layer.
        """
        return {name: self.neural_index_of(name, nudge) for name, nudge in action.items()}

    def neural_index_of(self, name: str, nudge: int) -> int:
        """
        Returns the index of the value of the action named `name` in the neural network's output layer.
        """
        cursor = 0
        for action_name in self.ACTION_NAMES:
            size, offset = self.env.action_space_layout[action_name]
            if action_name == name:
                return cursor + (nudge - offset)
            cursor += size

    def q_values_of_action(self, q_values, action_name: str) -> list[float]:
        size, offset = self.env.action_space_layout[action_name]
        max_nudge = size + offset
        start = self.neural_index_of(action_name, 0)
        end = self.neural_index_of(action_name, max_nudge)
        return q_values[start : end + 1]
```

rl_training.py
Processus d'entraînement de l'agent

```python
import random
from datetime import datetime, timedelta
import numpy as np
from typing import NamedTuple, Tuple, Union
import gym
from rl_environment import EdgeDetectionEnv
from rl_agent import EdgeDetectionAgent
from tqdm import trange
from pathlib import Path
import pygame
from rich import print, traceback

traceback.install()

env = EdgeDetectionEnv(
    render_mode=None,
    acceptable_brightness_range=(7, 15),
    acceptable_segments_count_range=(10, 25),
    dataset=Path("datasets/radiopaedia/cropped"),
    max_thresholds_increment=5,
    max_blur_value=50,
    step_blur_value=10,
    max_brightness_increment=3,
)

# WINDOW = pygame.display.set_mode((1000, 300))
# clock = pygame.time.Clock()

class Params(NamedTuple):
    memory_sample_size: int
    ε_fluctuations: int  # 0 to disable ε fluctuation
    max_episodes_count_without_progress: int = 0  # where
progress means an increment in reward. Use 0 to disable ECC
    ε_bounds: Tuple[Union[int, float], Union[int, float]] =
(0.001, 1)


def run(env: EdgeDetectionEnv, agent: EdgeDetectionAgent,
params: Params):
    episode_reward = 0
    ε = 1
    # ε_values = [ε]
    # rewards_history = []
    # epsiodes_without_progress_count = 0
    ε_decay = params.ε_bounds[1] - (
        params.ε_bounds[1] / int(env.dataset_size /
(params.ε_fluctuations or 0.8 * env.dataset_size))
    )

    episode = 0
    while not env.saw_everything:
        reward = 0
        step = 1
        action = 0

        current_state = env.reset()


        while not env.done():
            print(f"{datetime.now():%H:%M:%S}", end=" ")
            print("choose", end=" ")
            if random.random() > ε:
                print(f"[bold][magenta]E[/bold][/magenta]
{ε*100:.1f}%", end=" ")
                print("network", end=" ")
                action =
agent.what_do_you_want_to_do(current_state)
            else:
                print(f"[bold][cyan]X[/bold][/cyan]
{ε*100:.1f}%", end=" ")
                print("random", end=" ")
                action = env.action_space.sample()

            print("step", end=" ")
            new_state, reward, done, info =
env.step(action, ε)
            print(f"rewarded with {reward}", end=" ")
            episode_reward += reward

            print("remember", end=" ")
            agent.remember((current_state, action, reward,
new_state, done))
            print("train", end=" ")
            agent.train(done, step)

            current_state = new_state
            step += 1

            if agent.last_save is None or
abs(agent.last_save - datetime.now()) >
timedelta(minutes=15):
                agent.save_model(Path(__file__).parent /
"rl_models" / agent.name)

            # print("render", end=" ")
            # env.render(WINDOW)
            print("")

        print("== episode done! ==")

        if ε > params.ε_bounds[0]:
            print(f"decaying {ε = }")
            ε = max(ε * ε_decay, params.ε_bounds[0])

        if params.ε_fluctuations and episode %
int(env.dataset_size / params.ε_fluctuations) == 0:
            print(f"flucuating {ε = }")
            ε = params.ε_bounds[1]

        env.save_settings(agent.name, Path(__file__).parent
/ "rl_reports")

        print("====================")
        episode += 1

        # if episode_reward == rewards_history[-1]:
        #     epsiodes_without_progress_count += 1

        # rewards_history.append(episode_reward)

        # if  epsiodes_without_progress_count >
params.max_episodes_count_without_progress:
        #     ε = ε_values[-1]


if __name__ == "__main__":
    params = Params(memory_sample_size=128,
ε_fluctuations=2)
    agent = EdgeDetectionAgent(
        "perseverance",
        env,
        conv_list=[32],
        dense_list=[32, 32],
        discount_rate=0.99,
        memory_sample_size=params.memory_sample_size,
        update_target_model_every=5,
    )

    run(env, agent, params)
```