

Stage B: Définition sujet + priorités

pb usuel en CV = trouver et classifier un

objet dans une grande image:



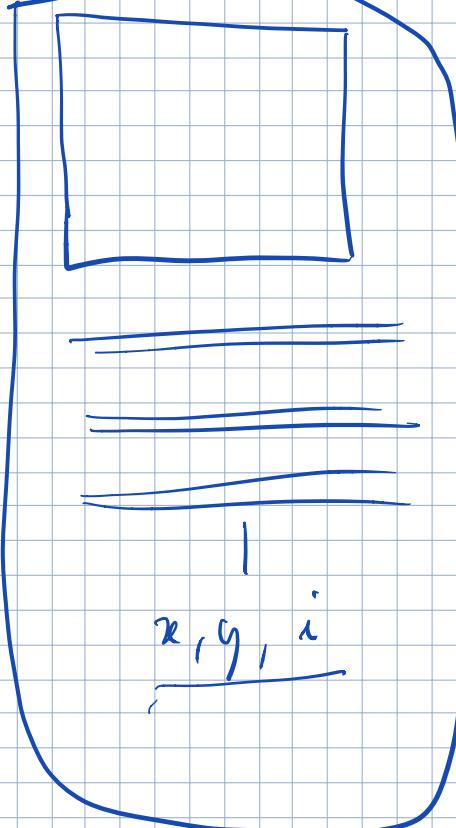
$x, y, i$

$x, y, i$

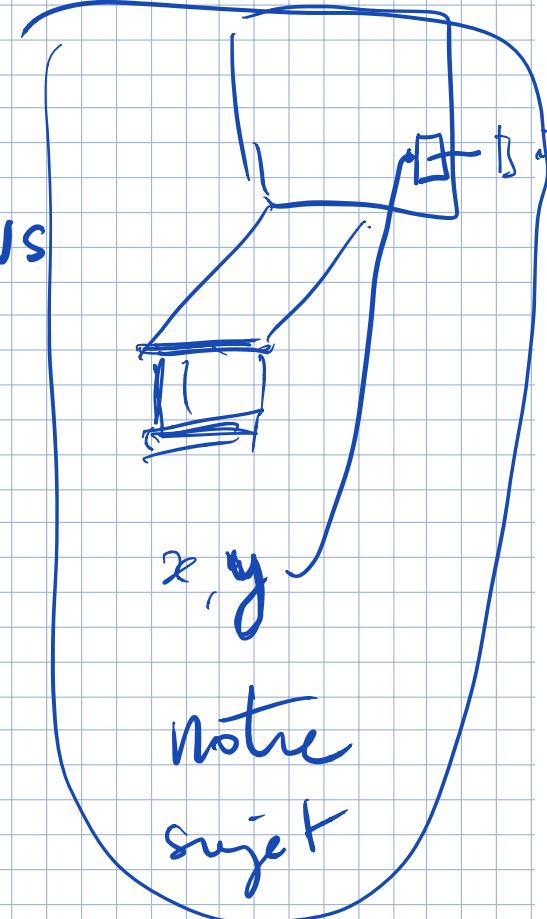
YOLO

BRUTE FORCE

2 hypothèses:

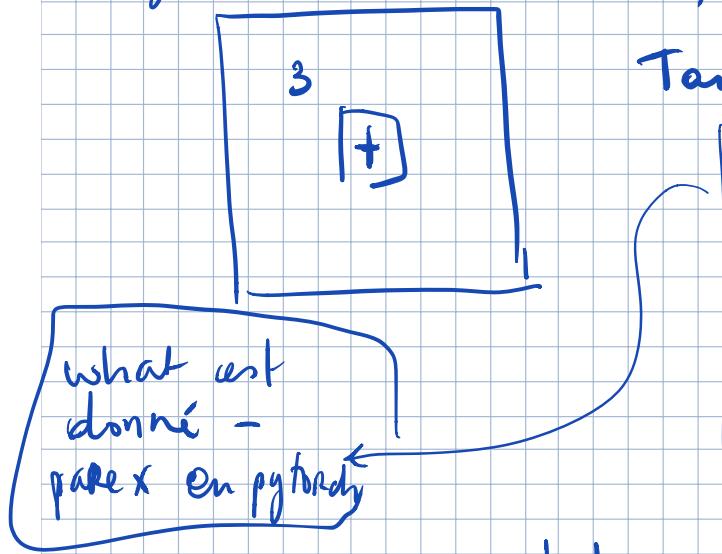


VS



classique  
(convolutif exhaustif)

## Algorithm



$$x, y \neq x_0, y_0$$

Tant que  $\Pi > \Pi_0$

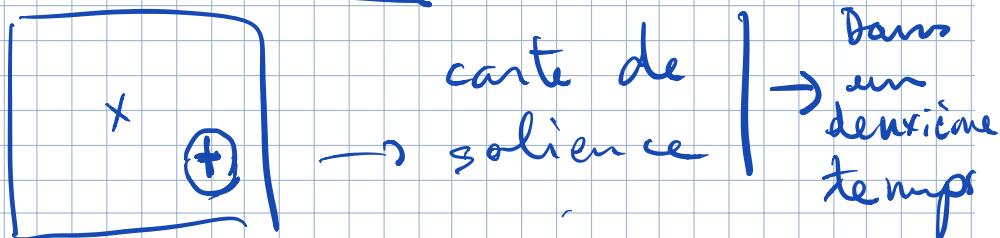
$i^*, \Pi = \text{what}(I(x, y))$

$$S' = LP(S, x, y)$$

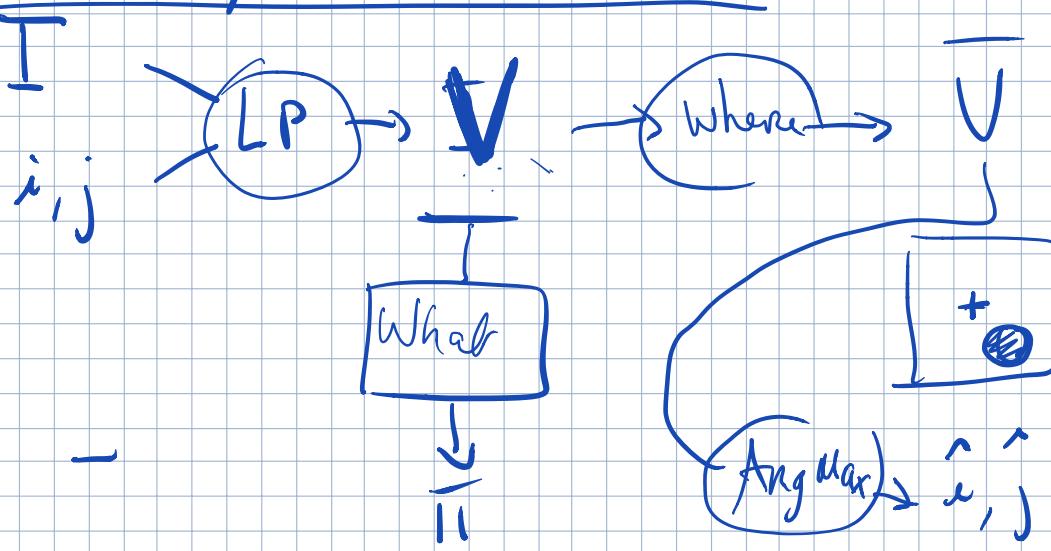
$$I' = LP(I, x, y)$$

$$x^*, y^* = \text{Whene}(I', S')$$

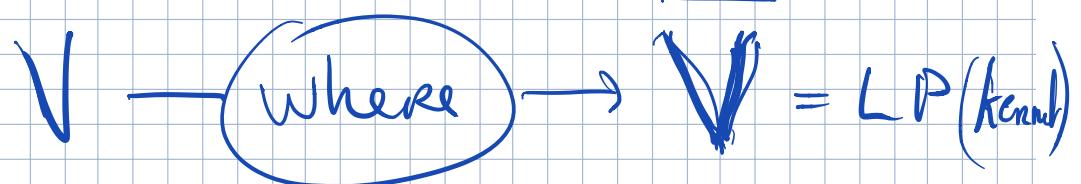
## return of inhibition



## Idee computationnelle :



Peut-être résolu simplement par  $\rightarrow$

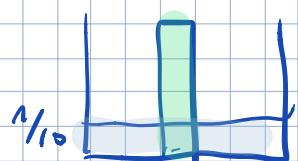


Traduit en mots :

Sachant une image  $I$  et un point de fixation  $i, j$

En utilisant le crop local autour du point de fixation ("macula") on peut classifier ce que l'on voit :

$$p_i = \text{What}(I_{\text{crop}, i, j}(I))$$



= mauvais  $\Rightarrow$  on continue  
= super!  $\Rightarrow$  on arrête

(Magn) sur la périph, nous avons accès à l'information visuelle sur la carte retinotique

$$V = LP_{i, j}(I)$$

Nous utilisons une fonction Where qui nous permet de ~~évaluer~~ la précision

proba de classification en fonction de  $V$  - elle opère sur un ensemble d'hypothèse de saccades  $H = \{(i, j)^{(k)}\}$

(l'amplitude de la saccade est  $a_{i,j}^{(k)} = |i^{(k)} - i|, |j^{(k)} - j|$ )

$$a_i^{(k)} = \text{Where}(V, (i, j)^{(k)})$$

↓

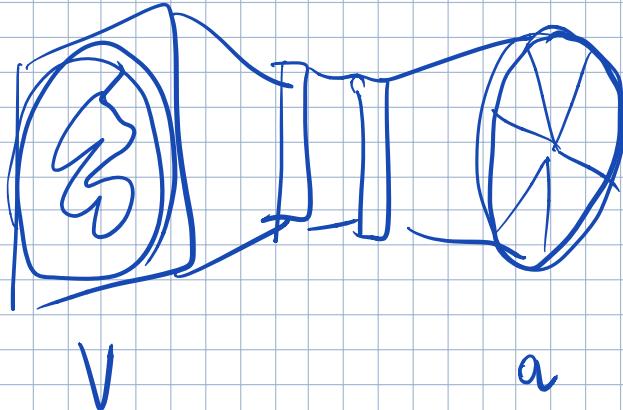
proba de classification évaluée sur une base de test

sachant qu'on a effectué un mvmt vers  $(i, j)^k$

Pour chaque hypothèse, (par exemple sur la grille des pixels, on mire une grille log-polaire centrée sur l'hypothèse courante soit  $(i, j)$ ).

Le but est d'apprendre la fonction qui génère les  $a_i^{(k)}$  sur les points  $(i, j)^{(k)}$  pour un  $V$  donné

→ c'est de l'apprentissage supervisé :

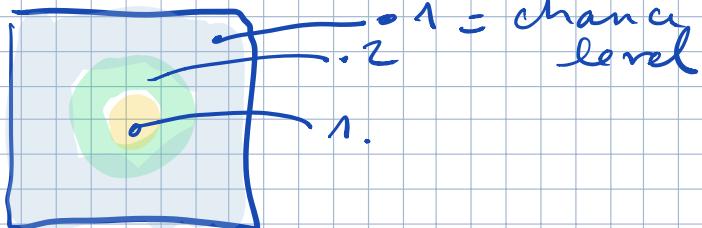


⇒ on peut l'apprendre par une descente de gradient en back prop

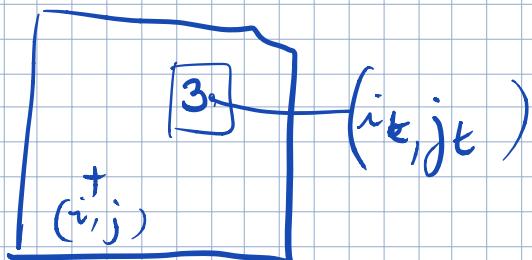
- Computationnellement, il peut paraître fastidieux de faire toutes les hypothèses à chaque saccade (même si c'est ce qu'on fait de notre vivant...) et on peut utiliser le kernel de 2018-02-16\_classifier.ipynb :

- On connaît pour une erreur de placement  $\Delta_i, \Delta_j$  la carte des  $a_i$ :

$$A(\Delta_i, \Delta_j) =$$



- Sur une image où on connaît la vraie position de l'objet  $(i_t, j_t)$



on a pas besoin de calculer  $a_i$  pour tous les points car ça dépend pas de  $(i, j)$  mais seulement de

$$- \Delta_i, \Delta_j = i^{(k)} - i, j^{(k)} - j$$

Et donc pour chaque hypothèse,

$$a^{(k)} = A \left( i^{(k)} - i, j^{(k)} - j \right)$$

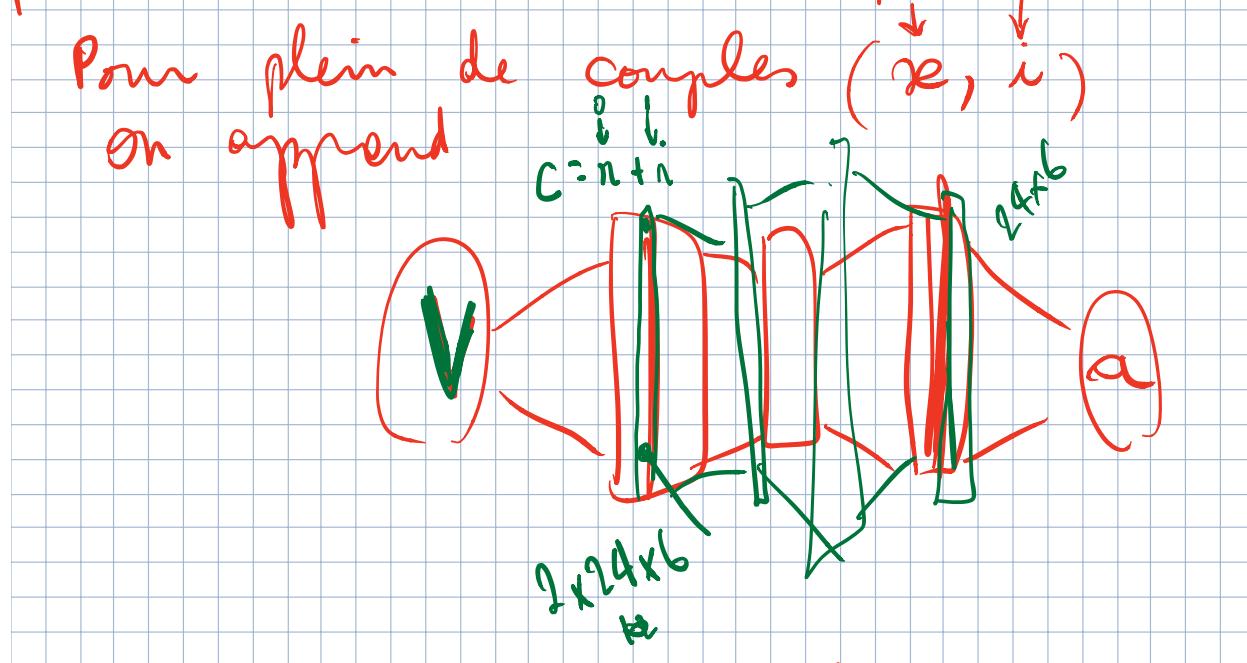
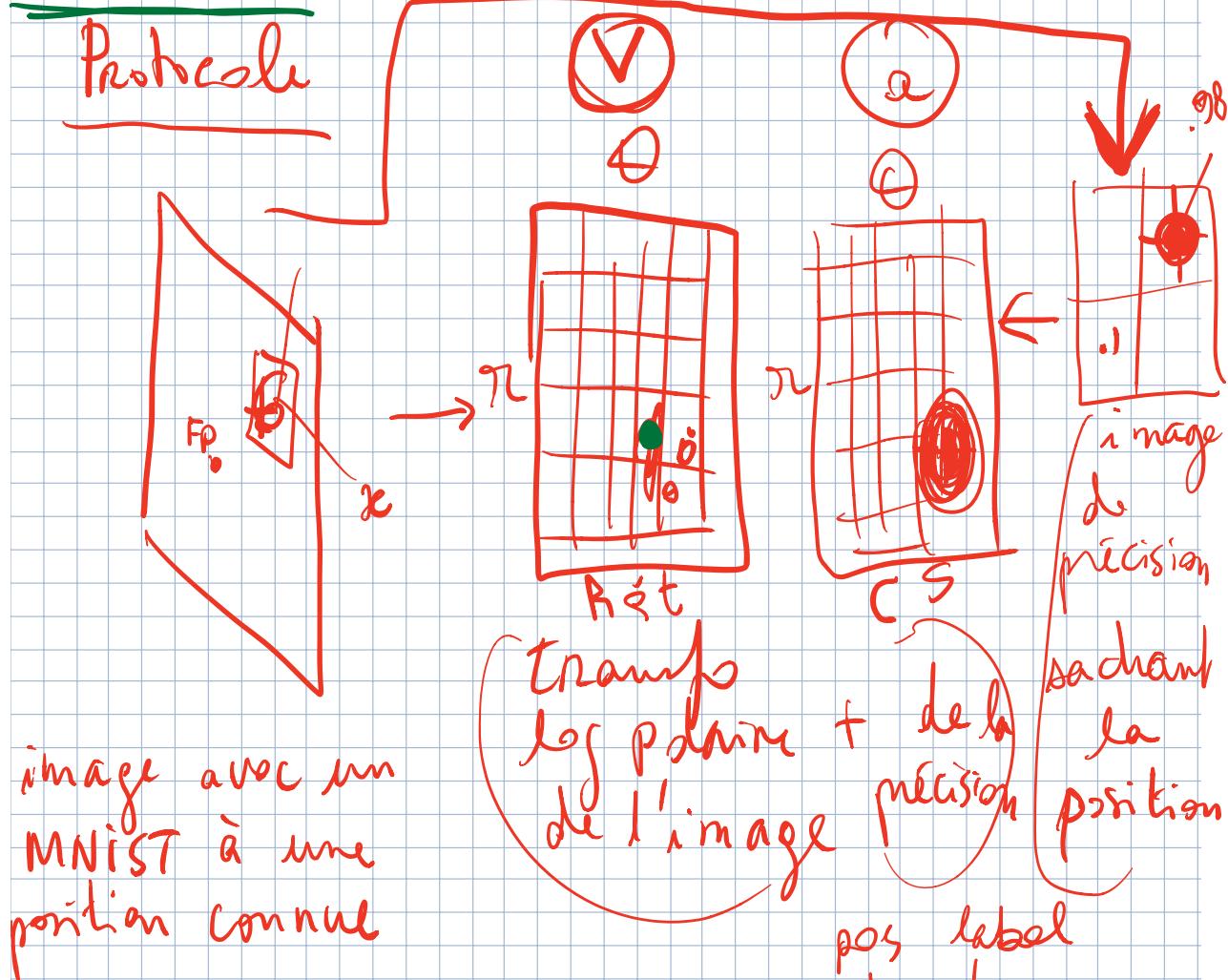
en particulier, pour une liste de points  $\mathcal{H}$  sur la log-polaire, ça ressemble à une transfo log-polaire

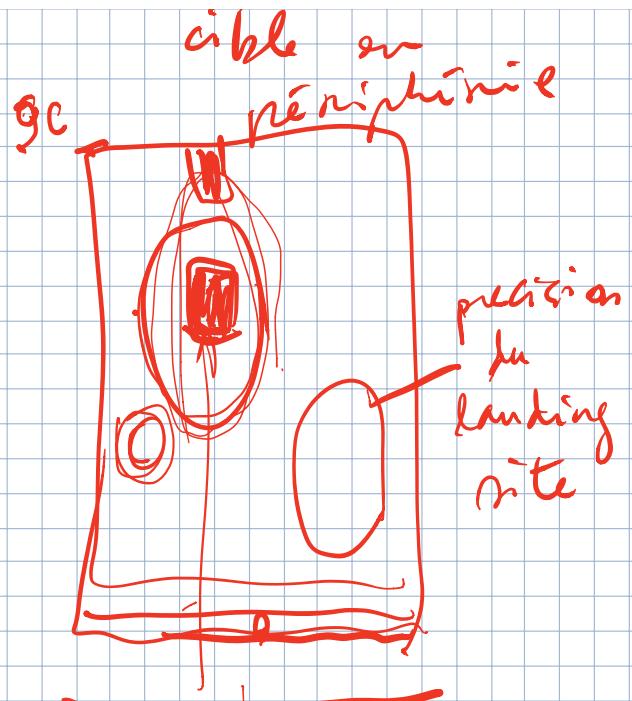
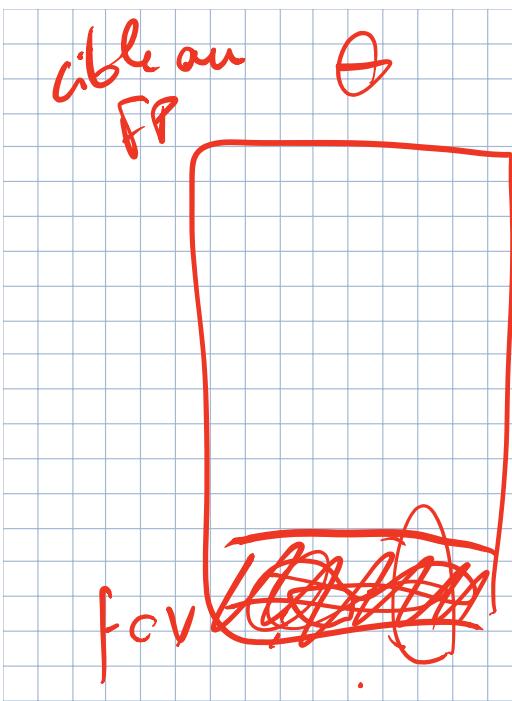
## Priorités

- collecter des paires (V, ~~A~~)
- apprendre à médire à sa chant V
- appliquer à un cas plus complexe avec du bruit.

2018 - 03 - 30

## Protocol





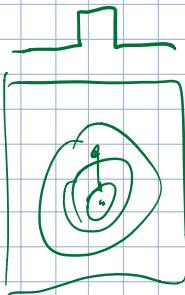
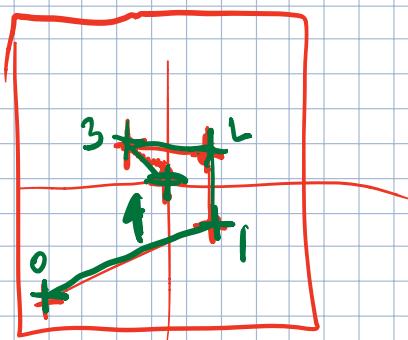
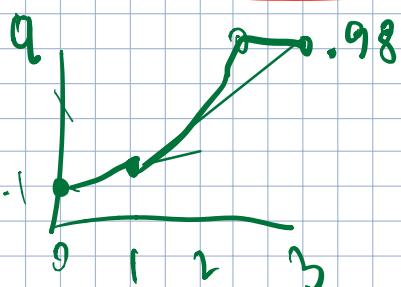
→ Fitt's Law ?

Retour sur what

on a défini le centre arbitraire -

$(x_0)$  en connaissant

Logo réelisé

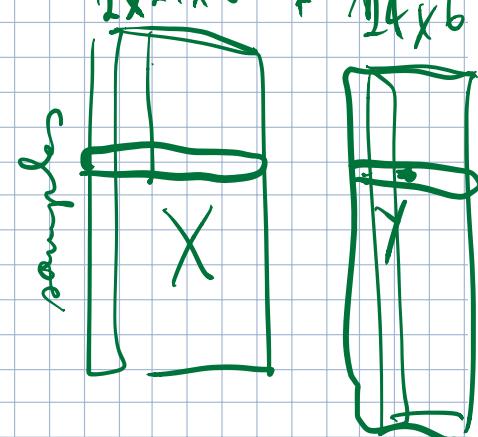


## Timeline:

- ipynb → markdown

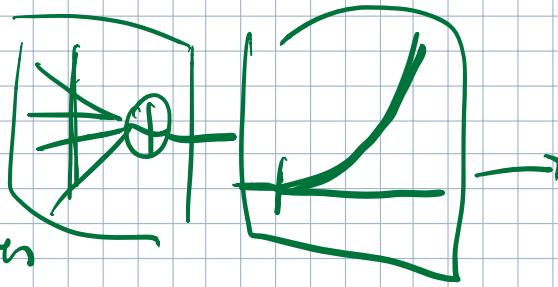
$2 \times 7 \times 6 + 1 \times 6$

→ base de données  
à partir de  $a_0$



- enlayer avec un réseau  
↑ couche convol.

- tester taille kernel
- tester différentes Non lnr.



→ 1 et 2 couches

Sujet de thèse: dans un pb

vision  
moteur  
langage

Comment réconcilier

WHAT

et

WHERE

↓  
deep-learning

↓  
stage de Master