

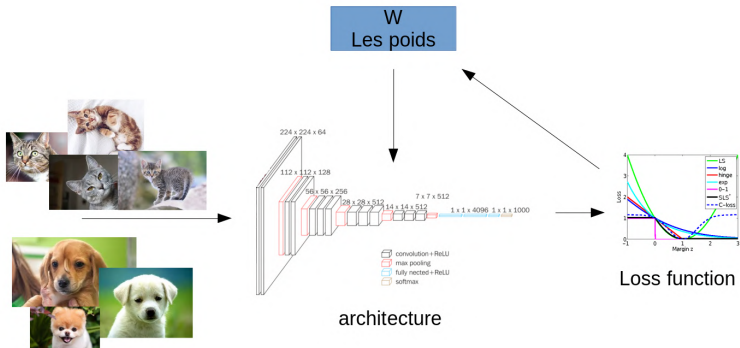
## Deep learning

mots clés : deep learning, convolution, pooling, pytorch, segmentation,  
exemples adversaires

Adrien CHAN-HON-TONG  
ONERA

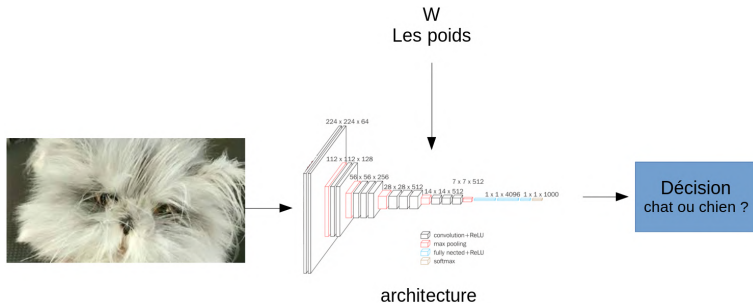
# Rappel : apprentissage vs test

## Apprentissage



# Rappel : apprentissage vs test

## Test et/ou production et/ou inférence



# Rappel

## SVM vs DL

Dans le cas du SVM, on a  $f(x, w) = w^T x + w_{\text{biais}}$  qui donne un signe

$$f(x, w) > 0 \text{ ou } f(x, w) < 0$$

pour dire de quel coté on est.

Dans un réseau de neurone c'est PAREIL sauf que

$$f(x, w) = w_Q \times \text{relu}(w_{Q-1} \times \text{relu}(\dots(\text{relu}(w_1 \times x))))$$

# Rappel

## L'apprentissage en pratique

L'apprentissage consiste à appliquer la méthode de la descente de gradient stochastique (optimiseur à choisir) à une fonction de perte (à choisir) qui approxime l'erreur d'apprentissage

Par exemple

$$partial\_loss(w) = \sum_{n \in Batch} relu(1 - y_n f(x_n, w))$$

$$w = w - \lambda_{iter} \nabla_w partial\_loss$$

# Plan

- ▶ Rappel
- ▶ Pytorch
- ▶ CNN
- ▶ Segmentation
- ▶ Exemples adversaires
- ▶ Perspective

## Forward - Backward

```
for t
  for i
    for j
       $A[t][i] += \text{relu}(A[t-1][j]) * w[t-1][i][j]$ 
DA[z][1] = partial_loss
for t from z to 1
  for j
    for i
       $DA[t][j] += DA[t+1][i] * w[t][i][j] * \text{relu}'(A[t][j])$ 
```

# Pytorch

## Forward backward

```
for t
  for i
    for j
       $A[t][i] += \text{relu}(A[t-1][j]) * w[t-1][i][j]$ 
DA[z][1] = partial_loss
for t from z to 1
  for j
    for i
       $DA[t][j] += DA[t+1][i] * w[t][i][j] * \text{relu}'(A[t][j])$ 
```

## Forward backward en pytorch

```
z = net(x)
loss = l(z,y)
loss.backward()
```



## Qu'est ce que c'est

- ▶ C'est un moteur de réseau de neurones : ça permet de programmer à l'aide de fonction haut niveau (`loss.backward()`), le reste est pris en charge par le moteur
- ▶ Cette abstraction est réalisé grâce à des objets *variable* qui stocke leur dépendance :  $c = a + b$ , la variable  $c$  stocke qu'elle dépend de  $a$  et  $b$
- ▶ En plus, la plupart des objets classiques sont précodés (couches de neurones, stratégie d'optimisation, loss function, activation)
- ▶ En particulier pour l'image, torchvision propose des fonctions de très haut niveau (ex téléchargement et chargement en mémoire le jeu de données MNIST)

## moindre carré

minimisons  $f(x) = (Ax - b) \cdot (Ax - b)$

### À la main

pour  $t$  de 1 à  $T$  :

$$\nabla_x f = 2A^T(Ax_t - b)$$

$$x_{t+1} = x_t - \rho \nabla_x f$$

### Pytorch

`optimizer = optim.SGD([x], lr= $\nabla$ )`

pour  $t$  de 1 à  $T$  :

$$f = (Ax - b) \cdot (Ax - b)$$

`optimizer.zero_grad()`

`f.backward()`

`optimizer.step()`

# TODO

TODO - on leur dit quoi d'autres sur pytorch ??

# Plan

- ▶ Rappel
- ▶ Pytorch
- ▶ CNN
- ▶ Segmentation
- ▶ Exemples adversaires
- ▶ Perspective

# Données structurées

## Avant le deep learning

Données annotées et structurées  
en dimension 100000000



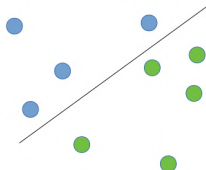
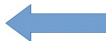
Descriptions des données (features)  
faites à la main  
(hand crafted)



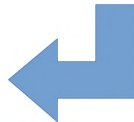
Nuage de points (D=10000)



chat/chien



Classification de points



# Données structurées

## Après

Données annotées et structurées  
en dimension 100000000

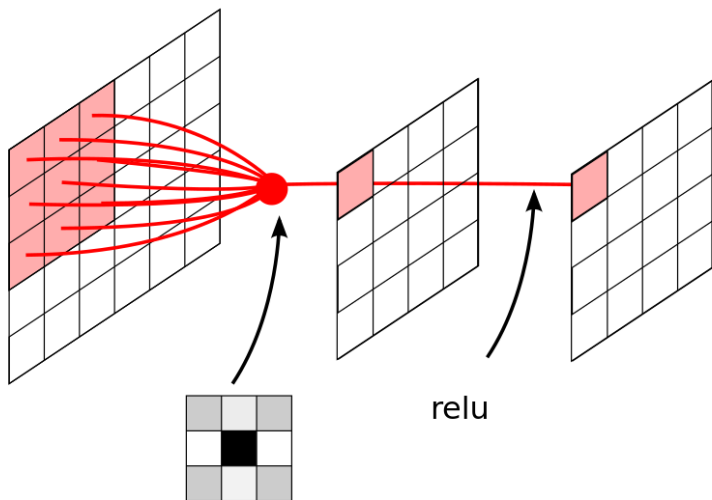


Le réseau de neurones fait TOUT

chat/chien

# Données structurées

## Le neurone convolutif



# Données structurées

## Le neurone convolutif

Si l'entrée est  $I \in \mathbb{R}^{C \times H \times L}$  :  $C$  canaux (3 pour Rouge Vert Bleu)  $H$  la hauteur et  $L$  la largeur. On peut considérer la convolution de  $I$  avec un noyau  $K \in \mathbb{R}^{C \times (2\delta_H+1) \times (2\delta_L+1)}$  noté  $I \star K \in \mathbb{R}^{H \times L}$  et défini par :

$$(I \star K)_{h,l} = \sum_{c=0}^C \sum_{\alpha=0}^{2\delta_H+1} \sum_{\beta=0}^{2\delta_L+1} I_{c,h-\delta_H+\alpha,l-\delta_L+\beta} \times K_{c,\alpha,\beta}$$

Comme pour les neurones, on pourra considérer un groupe de  $\mathcal{C}$  neurones convolutifs  $K_1, \dots, K_C$  dont on regroupe les sorties en une nouvelle image dans  $\mathbb{R}^{C \times H \times L}$ .



## Le neurone convolutif

Si on considère 1 valeur de  $I \star K$  typiquement

$$(I \star K)_{h,l} = \sum_{c=0}^C \sum_{\alpha=0}^{2\delta_H+1} \sum_{\beta=0}^{2\delta_L+1} I_{c,h-\delta_H+\alpha,l-\delta_L+\beta} \times K_{c,\alpha,\beta}$$

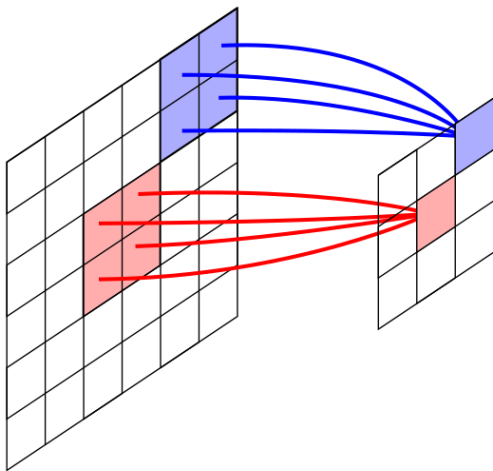
On voit que cette valeur peut tout à fait se coder avec un neurone classique  $I \cdot \mathcal{K}$  avec

$$\mathcal{K}_{i,j} = \begin{cases} K_{i-h+\delta_H,j-l+\delta_L} & \text{si } \begin{matrix} h - \delta_H \leq i \leq h + \delta_H \\ l - \delta_L \leq j \leq l + \delta_L \end{matrix} \\ 0 & \text{sinon} \end{cases}$$

**SAUF** que la convolution n'a que  $O(\delta_H \times \delta_L)$  paramètres pour générer  $I \star K$  contre  $O(H^2 \times L^2)$  avec une couche classique : 9 contre 4294967296 pour une image 256x256 et un noyau 3x3 ...

# CNN

## Le pooling



## Le pooling

Si l'entrée est  $I \in \mathbb{R}^{C \times H \times L}$ , alors  $\text{pool}(I) \in \mathbb{R}^{C \times \frac{H}{2} \times \frac{L}{2}}$

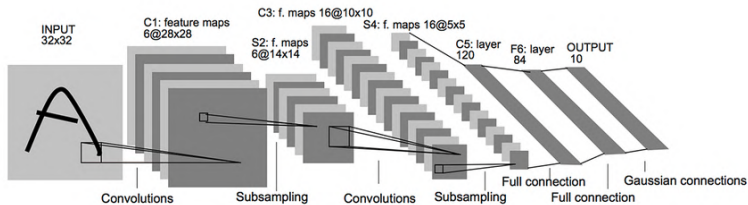
$$\text{pool}(I)_{c,h,l} = \max_{\alpha \in \{2h, 2h+1\}, \beta \in \{2l, 2l+1\}} I_{c,\alpha,\beta}$$

Il s'agit plus d'une activation car il n'y a pas de poids.

Notez que le max peut se coder avec un relu (comme vu en TD :  $\max(a, b) = \text{relu}(b - a) + a$ ) donc en théorie les réseaux relu peuvent déjà encoder le pooling. Mais en pratique cela force une certaine invariance...

# CNN

## Lenet

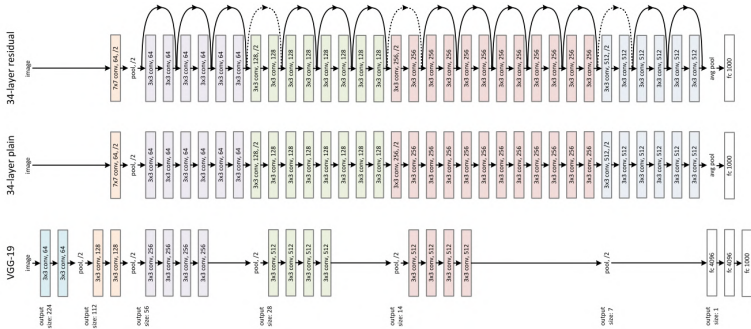


# CNN

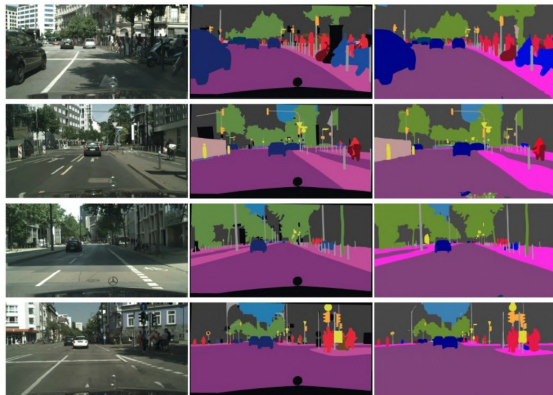
## Lenet, Alexnet, VGG



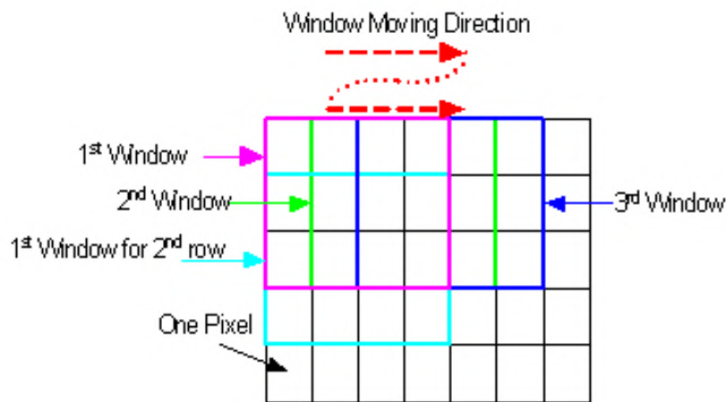
## VGG, Resnet



# Problèmes structurés



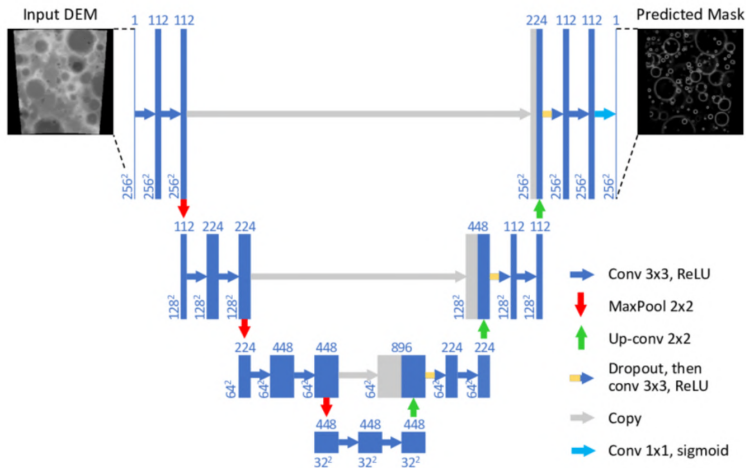
## segmentation sémantique



raisonner par fenêtre est une mauvaise idée : il faut raisonner par couche !



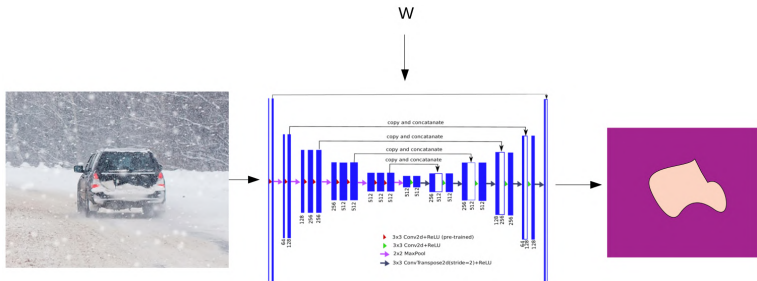
# segmentation sémantique





# segmentation sémantique : 2 phases aussi

Test



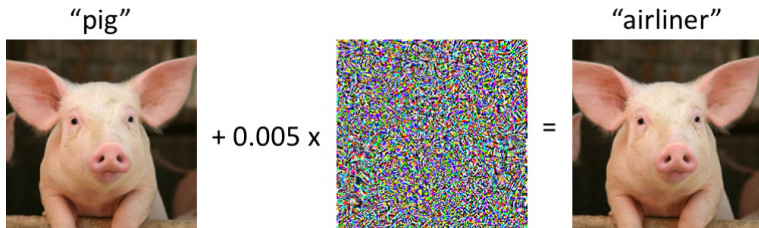
Le deep learning est incontestablement l'état de l'art sur les  
données/problèmes structurées  
(son, image, vidéo, texte, détection, segmentation, génération...)

# Plan

- ▶ Rappel
- ▶ Pytorch
- ▶ CNN
- ▶ Segmentation
- ▶ Exemples adversaires
- ▶ Perspective

# Grande dimension et instabilité

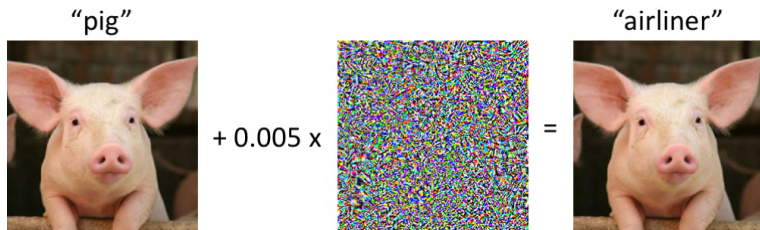
## Exemple adversaire



Les réseaux de neurones sont sensibles à des perturbations invisibles à l'oeil !

# Grande dimension et instabilité

## Exemple adversaire



Les réseaux de neurones sont sensibles à des perturbations invisibles à l'oeil !

En réalité, c'est pas clair que ce soit grave car ces perturbations ne pas forcément réalisable physiquement

# Grande dimension et instabilité

Pourquoi ?

L'apprentissage consiste à calculer

$$\nabla_w \text{loss}(y_n, f(x_n, w))$$

et à **actualiser**  $w$  de sorte que

$$\text{loss}(y_n, f(x_n, w)) \approx 0$$



# Grande dimension et instabilité

Pourquoi ?

Mais avec le même outils, on peut calculer

$$\nabla_x \text{loss}(y_n, f(x_n, w))$$

et **actualiser**  $x_n$  de sorte que

$$\text{loss}(y_n, f(x_n, w)) \gg 0$$

# Grande dimension et instabilité

## Pourquoi ?

Mais avec le même outils, on peut calculer

$$\nabla_x \text{loss}(y_n, f(x_n, w))$$

et **actualiser**  $x_n$  de sorte que

$$\text{loss}(y_n, f(x_n, w)) \gg 0$$

$\Rightarrow$  ce qui permet de construire une image  $x_n$  spécifiquement perturbée pour échapper au réseau : *adversarial exemple*.

# Grande dimension et instabilité

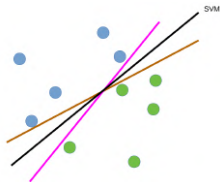
## Rendre les réseaux robustes

- ▶  $f$  un réseau binaire
- ▶ Ce qu'on ne veut pas c'est  $f(x) > 0$  et  $f(x + \textit{delta}) < 0$  (ou l'inverse) sur le testing set
- ▶ on veut donc apprendre au réseau à considérer que  $x$  est bien classé
  - ▶ non pas si  $f(x) > 0$
  - ▶ mais si  $f(x + \delta) > 0$  (avec  $\|\delta\| < \epsilon$ )

# Grande dimension et instabilité

## Rendre les réseaux robustes

- ▶  $f$  un réseau binaire
  - ▶ Ce qu'on ne veut pas c'est  $f(x) > 0$  et  $f(x + \textit{delta}) < 0$  (ou l'inverse) sur le testing set
  - ▶ on veut donc apprendre au réseau à considérer que  $x$  est bien classé
    - ▶ non pas si  $f(x) > 0$
    - ▶ mais si  $f(x + \delta) > 0$  (avec  $\|\delta\| < \epsilon$ )
- ⇒ comme ferait le SVM



# Grande dimension et instabilité

Rendre les réseaux robustes

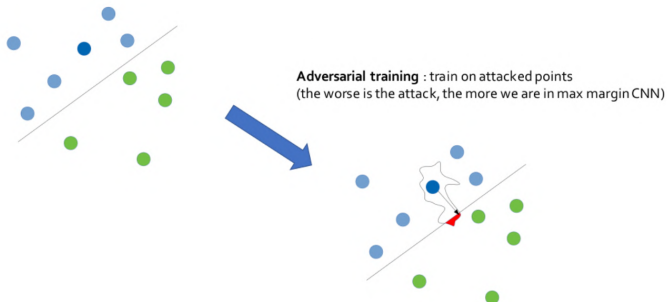
Sauf que calculer la marge c'est NP complet pour des réseaux relu !

# Grande dimension et instabilité

## Rendre les réseaux robustes

### 2 options

- ▶ adversarial training (sous estimé le déplacement maximal)
- ▶ construction d'une enveloppe convexe (sur estimation du déplacement maximal)



# Grande dimension et instabilité

## Rendre les réseaux robustes

2 options

- ▶ adversarial training (sous estimé le déplacement maximal)
- ▶ construction d'une enveloppe convexe (sur estimation du déplacement maximal)

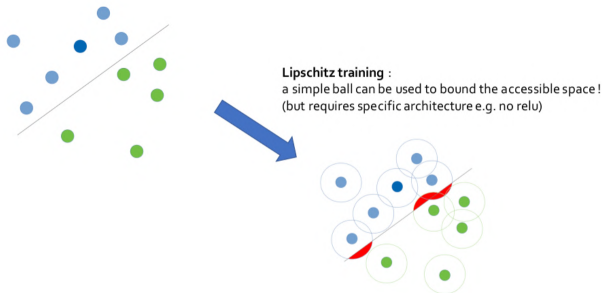


provable defenses against adversarial examples via the convex outer adversarial polytope

# Grande dimension et instabilité

## Rendre les réseaux robustes

Ou tenter des réseaux Lipschitz



Sorting Out Lipschitz Function Approximation



# Grande dimension et instabilité

## MinMax

$$\text{relu} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \max(0, a) \\ \max(0, b) \\ \max(0, c) \\ \max(0, d) \end{pmatrix}$$

(les valeurs n'augmentent pas mais elles peuvent diminuer)

$$\text{MinMax} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \max(a, b) \\ \max(c, d) \\ \min(a, b) \\ \min(c, d) \end{pmatrix}$$

(les valeurs ne changent pas, seul les places changent !)

# Grande dimension et instabilité

## MinMax

$$\text{MinMax} \begin{pmatrix} a \\ 0 \\ b \\ 0 \end{pmatrix} = \begin{pmatrix} \max(0, a) \\ \max(0, b) \\ \min(0, a) \\ \min(0, b) \end{pmatrix} = \begin{pmatrix} \text{relu} \begin{pmatrix} a \\ b \end{pmatrix} \\ -\text{relu} \begin{pmatrix} -a \\ -b \end{pmatrix} \end{pmatrix}$$

MinMax est aussi expressif que Relu en théorie

Mais l'activation est repoussée dans la partie linéaire pour permettre un meilleur contrôle du réseau !

## Les axes de recherche aujourd'hui

- ▶ frugal learning : apprendre avec peu de données
- ▶ incremental learning : Apprentissage de classes à la volé
- ▶ fairness, privacy preserving : Apprentissage éthique
- ▶ robustness : on en a parlé (tempête dans un verre d'eau)
- ▶ explainability : Apprentissage et langage
- ▶ physically informed neural network : Apprentissage hybride
- ▶ self supervised learning, representation learning : Apprentissage de représentation
- ▶ transfert learning : Adaptation de domaine

## Les axes de recherche aujourd'hui

- ▶ frugal learning : apprendre avec peu de données
- ▶ incremental learning : Apprentissage de classes à la volé
- ▶ fairness, privacy preserving : Apprentissage éthique
- ▶ robustness : on en a parlé (tempête dans un verre d'eau)
- ▶ explainability : Apprentissage et langage
- ▶ physically informed neural network : Apprentissage hybride
- ▶ self supervised learning, representation learning : Apprentissage de représentation
- ▶ transfert learning : Adaptation de domaine

**Mais le problème de la généralisation n'est PAS réglé !**

# Conclusion

## La généralisation

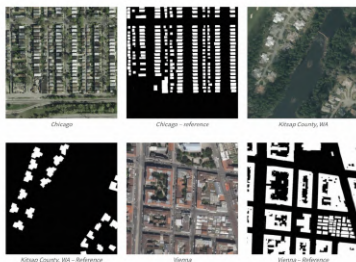
→ c'est un problème d'apprentissage mais aussi d'industrialisation de la collecte de données



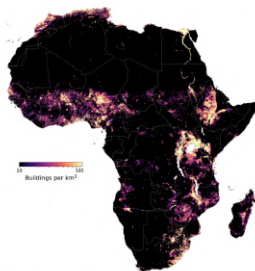
# Conclusion

## La généralisation

→ c'est un problème d'apprentissage mais aussi d'industrialisation de la collecte de données



Inria Aerial Dataset : 6 villes



Open building dataset : l'Afrique en entier

Questions ?