

Guess the AI



Présenté par
Thomas Bignon & Laura Wang





Can a neural network learn to recognize doodling?

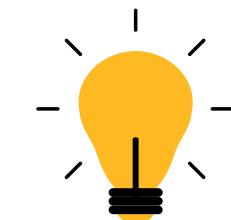
Help teach it by adding your drawings to the [world's largest doodling data set](#), shared publicly to help with machine learning research.

Objectifs



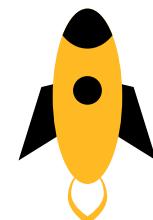
Génération

Générer des dessins par une IA.



Reconnaissance

Classifier des dessins d'humains.



Serveur web

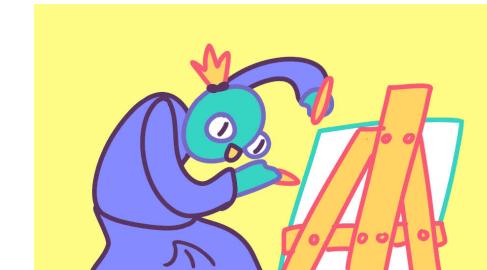
Création d'un serveur web pour une application implémentant les modèles.



Démonstration

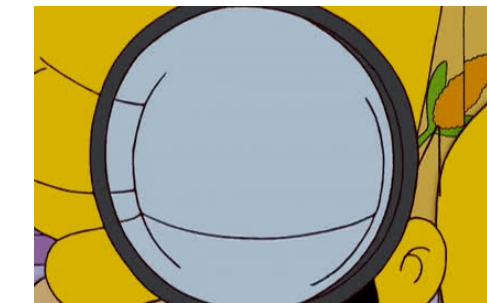
Connectez vous à :

bit.ly/bignonwang-plong



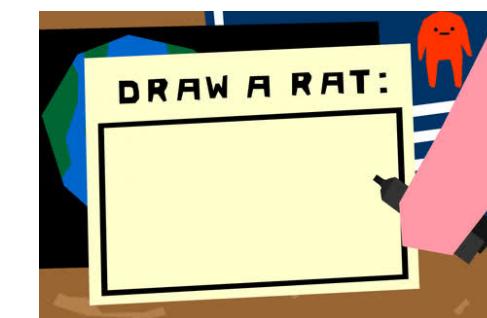
Faire une partie

/join



Tester la prédiction

/predict



Tester la génération

/generate/random

Conception & développement

- Création du modèle de classification CNN avec Keras
- Utilisation de Google Colab
- Serveur avec NodeJS
- Bootstrap (CSS)
- Dépendances NPM :
 - Express
 - Socket.io
 - Canvas
 - NumJS
 - Cli-progress
 - Colors.js
 - TensorFlow.js

Décomposition du problème



**Modèle de
classification
de dessins**

**Interface pour
dessiner et
traiter les
données**

**Modèles de
génération de
dessins**

**Construction du
serveur web et
du système de
salon**

```
notebooks/
  └── QuickDrawClassification_Thumbnails_Convert.ipynb
  └── QuickDrawClassification_Thumbnails_Training.ipynb
  └── download_decoder_models.py
```

```
public/
  └── html/
    ├── draw.html
    ├── guess.html
    ├── index.html
    ├── join.html
    ├── predict.html
    ├── room.html
    └── score.html
  └── js/
    ├── draw.js
    ├── guess.js
    ├── join.js
    ├── predict.js
    ├── room.js
    └── score.js
  └── style/
    └── style.css
```

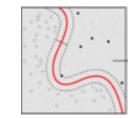
```
server/
  └── app.js
  └── img_processing.js
  └── generate.js
  └── classes.json
  └── sketch_rnn.js
  └── cnn_model/
    └── group1-shard1of2.bin
    └── group1-shard2of2.bin
    └── model.json
  └── key/
    └── cert.pem
    └── key.pem
    └── keytmp.pem
  └── sketch_rnn_model_data/
    └── bicycle.js
    └── butterfly.js
    └── castle.js
    └── cat.js
    └── eye.js
    └── face.js
    └── flower.js
    └── hand.js
    └── key.js
    └── mermaid.js
    └── pineapple.js
    └── rabbit.js
    └── snail.js
    └── spider.js
    └── truck.js
```



Architecture du projet



Compétences techniques



**Réseau neuronal convolutif &
Auto-encodeur**



TensorFlow & Keras

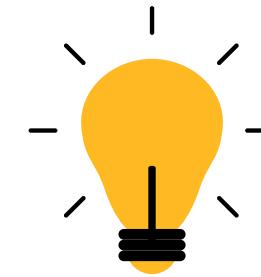


Les WebSockets (Socket.io)



PROBLÈMES

- Implémentation du modèle de classification avec en input les coordonnées des traits des dessins difficile.
- Les dessins des humains trop facilement différenciables des dessins générés.
- Création et entraînement d'un modèle de génération de dessins compliqué.
- Utilisation de serveur web pour jouer sur plusieurs ordinateurs



SOLUTIONS

- Création d'une miniature de l'image du dessin pour l'input du modèle de classification.
- Application d'un algorithme de simplification des lignes : Ramer-Douglas-Peucker.
- Utilisation de modèles pré-entraînés avec le même dataset.
- Utilisation de Ngrok pour faire un tunnel vers localhost.

Algorithme de traitements des coordonnées des points d'un dessin



```
fonction DouglasPeucker(points[1..n], ε)
    // Trouve l'indice du point le plus éloigné du segment
    dmax = 0
    index = 0
    pour i = 2 à n - 1
        d = distance du points[i] au segment [points[1], points[n])
        si d > dmax
            index = i
            dmax = d

    // Si la distance dmax est supérieure au seuil, appels récursifs
    si dmax > ε
        // Appel récursif de la fonction
        recPoints1[1..k] = DouglasPeucker(points[1..index], ε)
        recPoints2[1..m] = DouglasPeucker(points[index..n], ε)

        // Construit la liste des résultats à partir des résultats partiels
        renvoie la concaténation de recPoints1[1..k-1] et recResults2[1..m]

    sinon // Tous les points sont proches → renvoie un segment avec les extrémités
        renvoie [points[1], points[n]]

// Fonction appelée sur les données du dessin, à fois qu'un point est ajouté, quand le joueur dessine et que l'on doit mettre à jour l'affichage

fonction RenduDessin(lignes[1..n])
    // Trace les lignes une par une sauf la dernière
    pour i = 1 à n - 1
        trace la ligne lignes[i] sur la toile

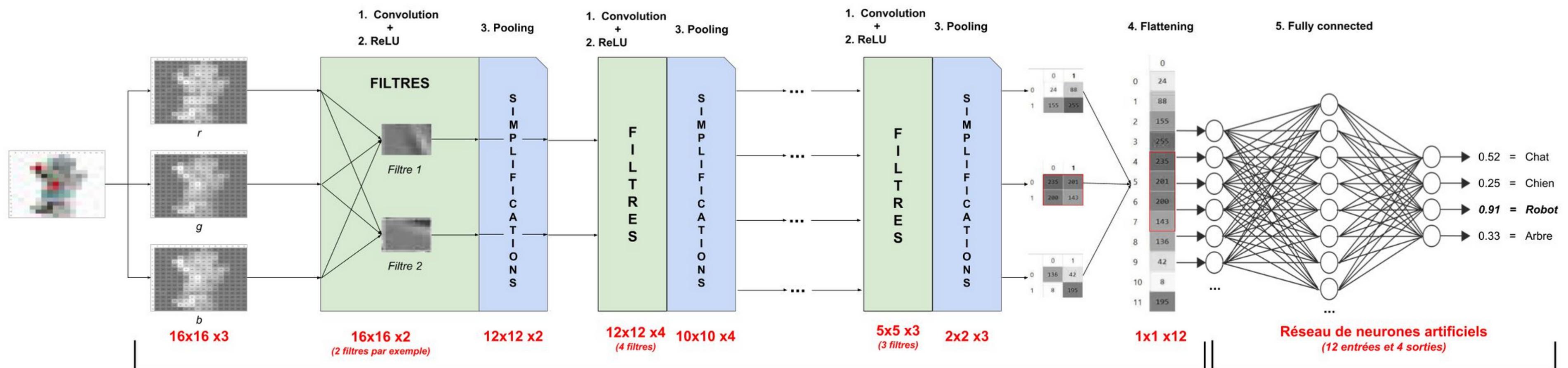
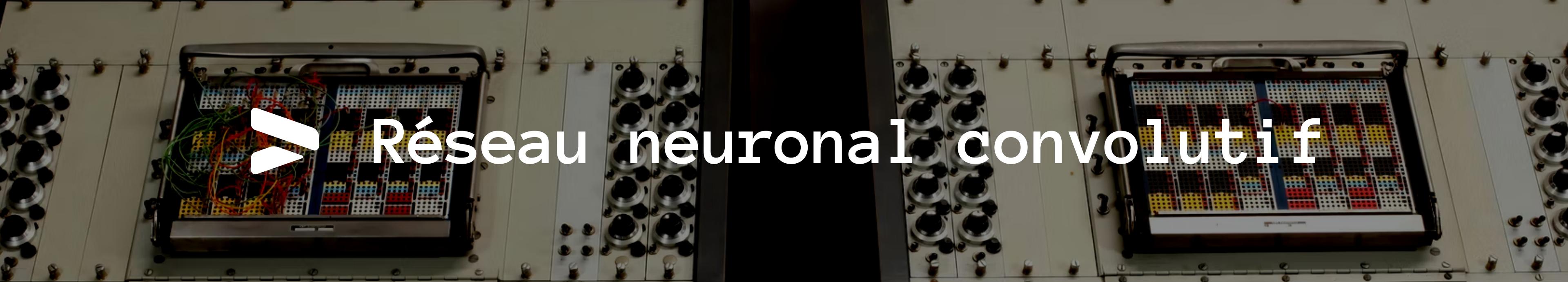
    // Trace la ligne qui est en train d'être dessinée
    trace la ligne DouglasPeucker(lignes[n], 1) sur la toile

// Fonction appelée sur les données du dessin quand le jour fini de dessiner une ligne
fonction finDeLigne(lignes[1..n])
    // Simplification de la dernière ligne
    lignes[n] = DouglasPeucker(lignes[n], 1)

    // Ajout d'une nouvelle ligne vide pour recevoir les prochains points
    lignes[n+1] = []
```



Réseau neuronal convolutif

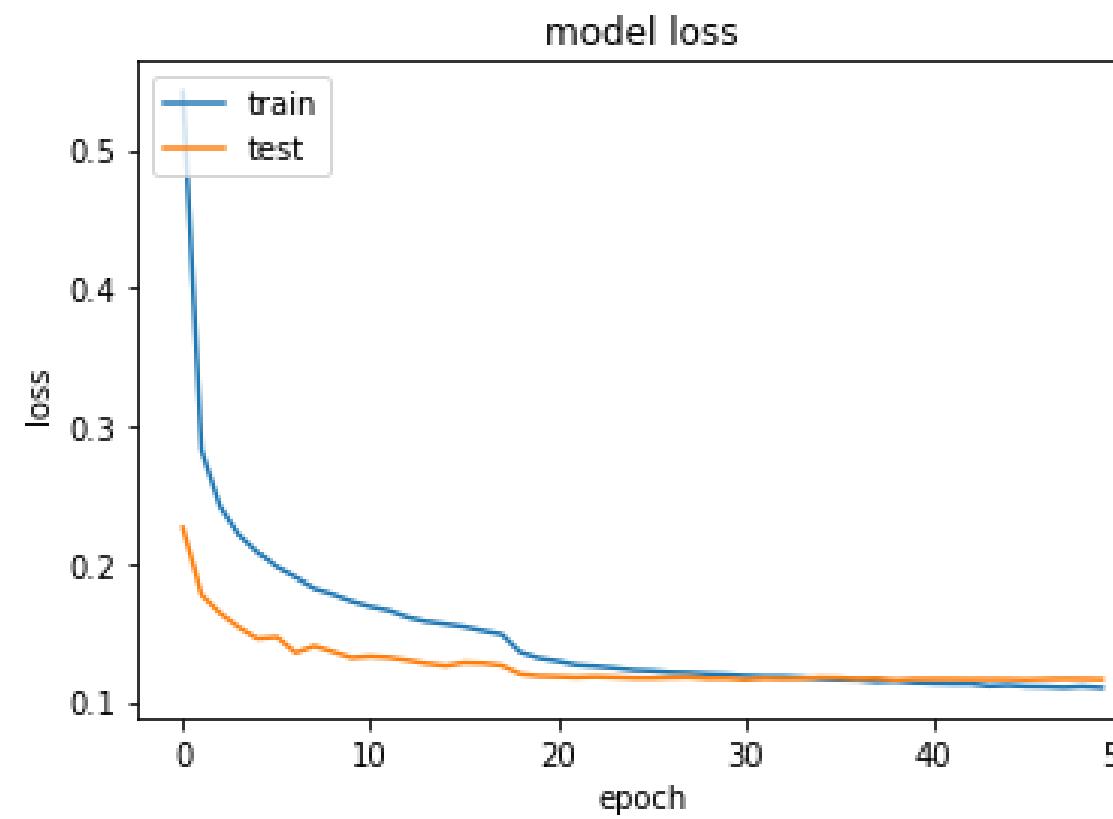
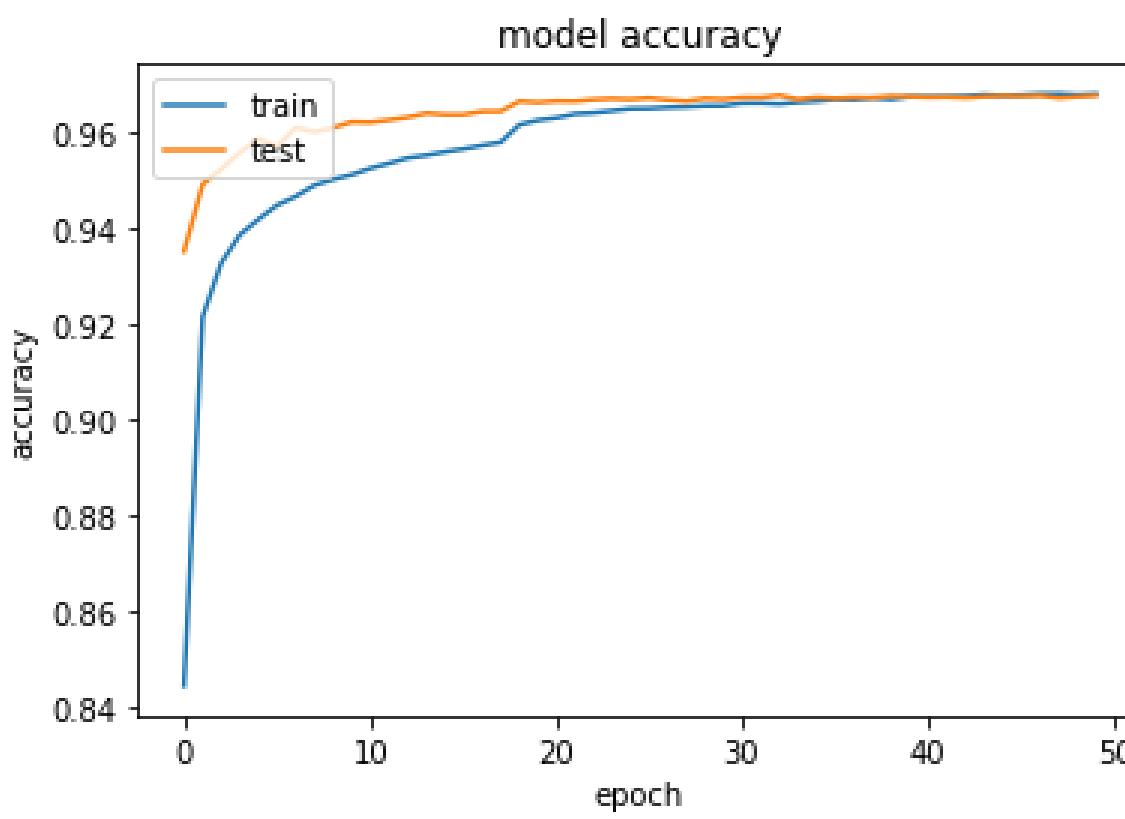


Extraction des informations de l'image grâce à un enchaînement de filtres (et de simplifications)

Prédiction de la classe de l'image

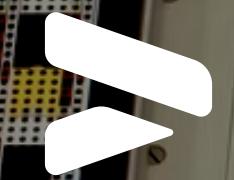


Réseau neuronal convolutif

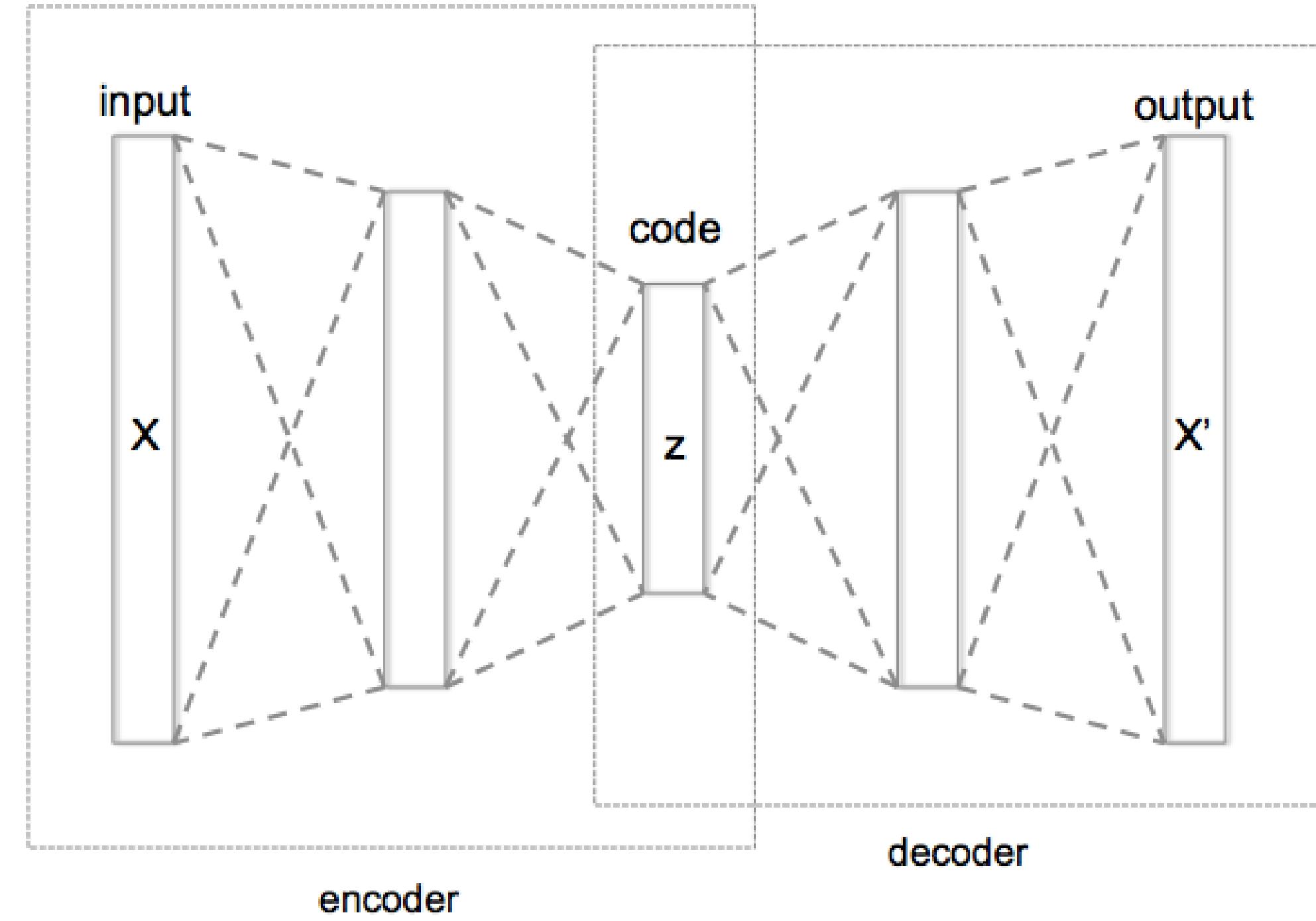


	precision	recall	support
bicycle	0.99	0.99	16500
butterfly	0.99	0.98	16500
castle	0.99	0.98	16500
cat	0.92	0.95	16500
eye	0.97	0.97	16500
face	0.97	0.98	16500
flower	0.97	0.98	16500
hand	0.97	0.96	16500
key	0.94	0.96	16500
mermaid	0.93	0.97	16500
pineapple	0.98	0.98	16500
rabbit	0.95	0.91	16500
snail	0.98	0.97	16500
spider	0.97	0.96	16500
truck	0.98	0.98	16500
accuracy			247500
macro avg	0.97	0.97	247500
weighted avg	0.97	0.97	247500

Top 5 accuracy: 0.991648
Top 3 accuracy: 0.987002



Auto-encodeur





Conclusion



VERSION 2.0

- Amélioration de la sécurité côté client
- Faire une version qui fonctionne sur mobile
- Ajouter des modes de jeu qui implémentent les fonctionnalités existantes



SI ON POUVAIT REFAIRE LE PROJET

- Faire le serveur web en Python avec Flask
- Meilleure répartition du temps
- Essayer de faire la reconnaissance de dessins en utilisant les coordonnées des points en input