Collecte de Données et Classification de Data Visualization

Soutenance finale

29/03/2018

Plan

- Equipe - Contexte - Objectifs
- Démarche - Planning - Livrables
- Les Réseaux de Neurones Convolutionnels
- Implémentation
- Résultats

L’équipe

Chef de projet

Alexis MARTIN - DELAHAYE

Fonctionnel

Marc ARNAL

Dev Team

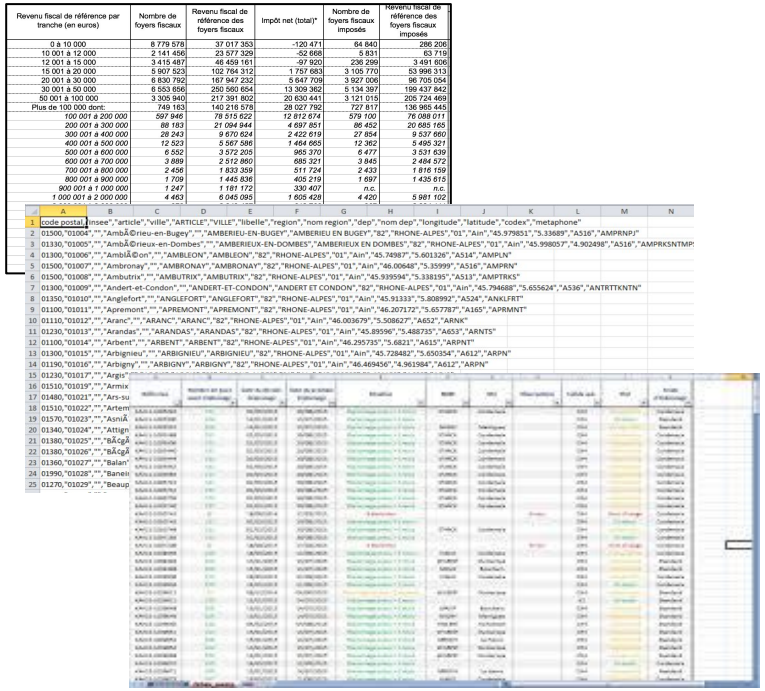
Luca GUÉRY

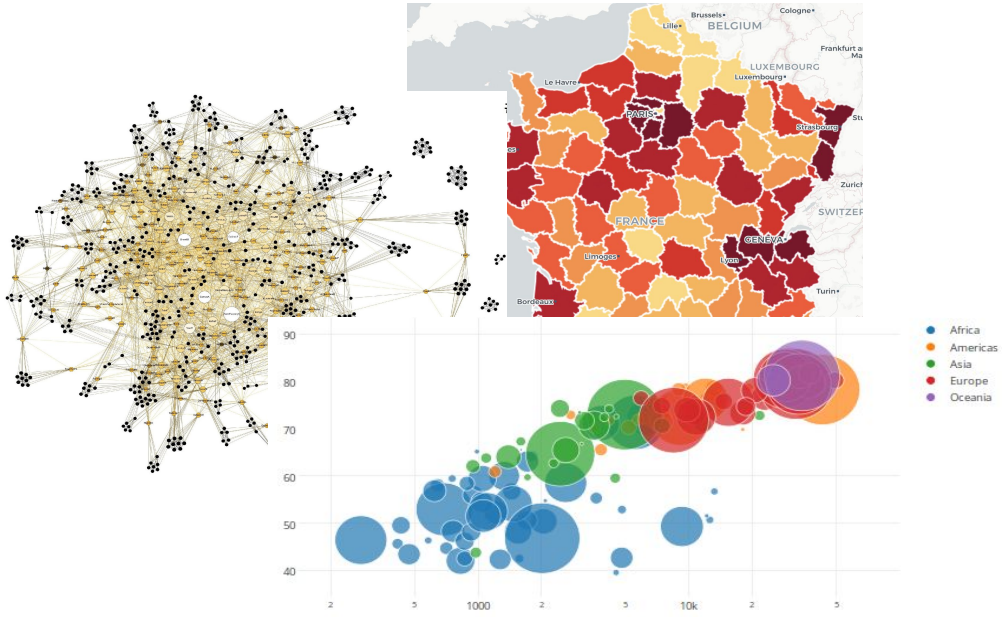
Louis KRAEMER

Arnaud BRUGIERE

Contexte & Objectifs

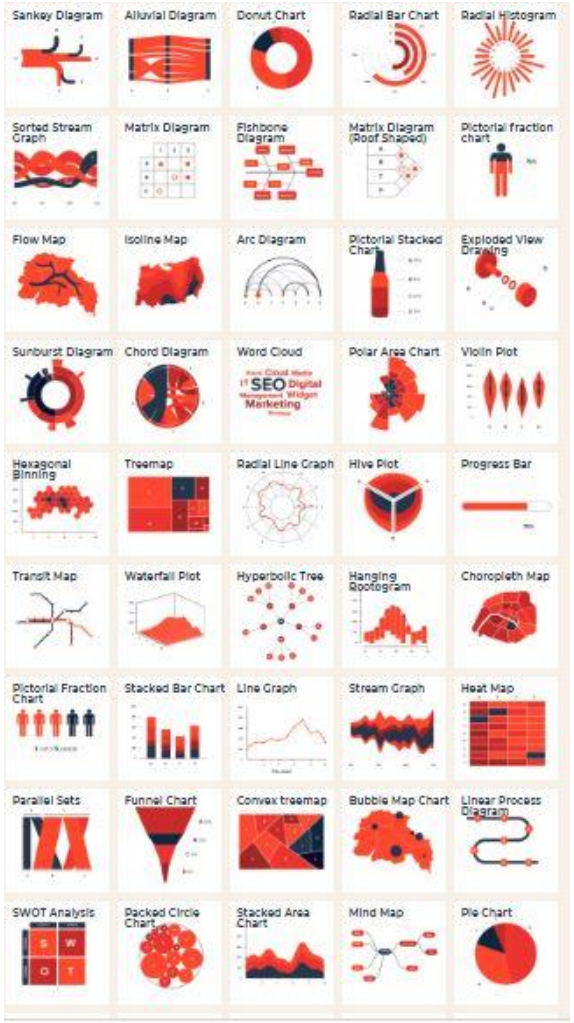
Data Visualization : représentation visuelle des données pour une **meilleure compréhension**

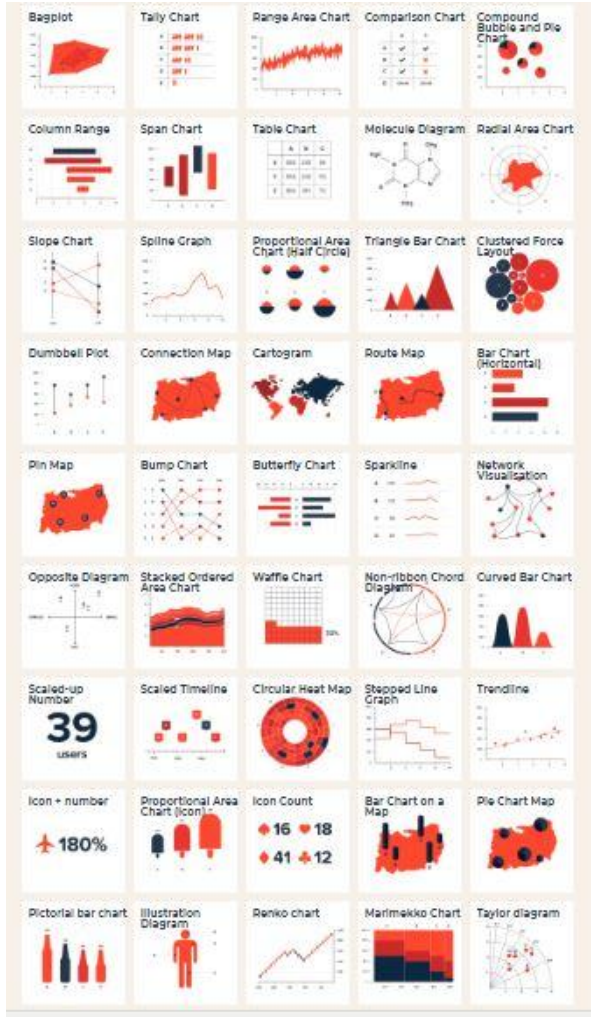




Objectif : labéliser et trier automatiquement les visuels de data-visualisation et construire une base de données complète de visuels

Tour d’horizon des visualisations de données





KEY INSIGHTS :

- Forte diversité
  - Modèles
  - Outils
- Forte concentration
- Forte évolution

Démarche - Planning par phase

Phase 1 : Cadrage

- Organisation équipe
- Fixation méthode et planning
- Etat de l'art

Phase 2 : Détermination et construction algorithme

Choix et architecture de notre premier CNN

Phase 3 : Détermination des visualisations de données adaptées et collecte

Nous nous sommes d'abord concentré sur 3 classes : les diagrammes à aires, à barres et à lignes

Phase 4 : Itération sur l'algorithme

Nous avons changé les couches et les paramétrages pour déterminer la meilleure architecture possible

Phase 5 : Généralisation

Extension jusqu'à 10 classes de visualisation de données

Livrables

LIVRABLES FINALISÉS:

- **GitHub** : code et documentation, tutoriels.
- **Rapport :** contexte, explication et analyse des résultats
- **Support de présentations :** présentation de cadrage, de mi-parcours, finale

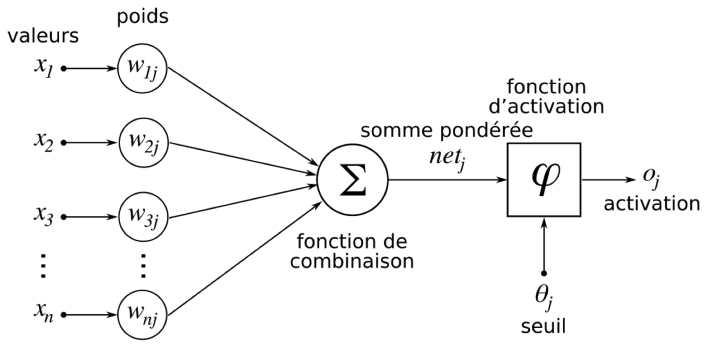
LIVRABLES POUVANT FAIRE L'OBJET DE PROCHAINES ÉTAPES :

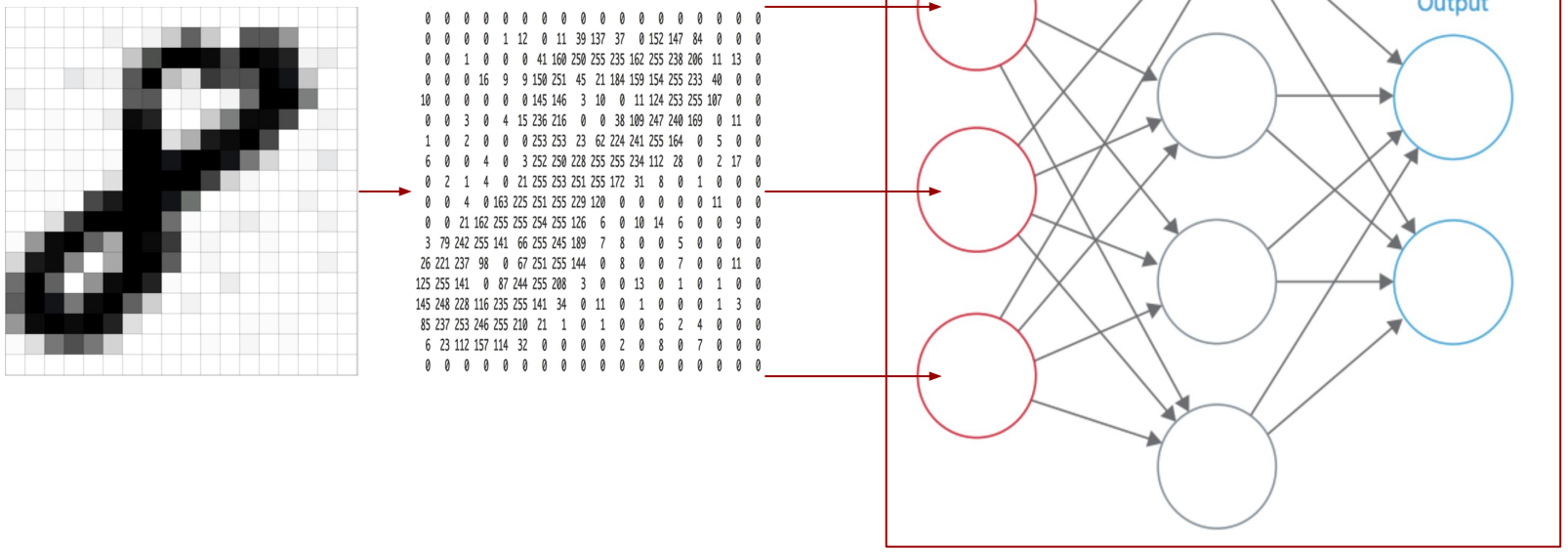
- **Site**: hébergement au-delà d'un fonctionnement en local.
- **Algorithme :** amélioration continue possible par des itérations supplémentaires

Les réseaux de neurones

Les Réseaux de Neurones

Fonctionnement d'un réseau de neurones :





- Chaque objet dans l'image est considéré différemment
- Apprend indépendamment à reconnaître des formes

Les Réseaux de Neurones Convolutionnels

Fonctionnement d'une couche de Convolution :

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
0	0	1
1	-1	1

w0[:,:,1]

-1	0	1
1	-1	1
0	1	0

w0[:,:,2]

-1	1	1
1	1	0
0	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	-1
0	-1	0
0	-1	1

w1[:,:,1]

-1	0	0
1	-1	0
1	-1	0

w1[:,:,2]

-1	1	-1
0	-1	-1
1	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

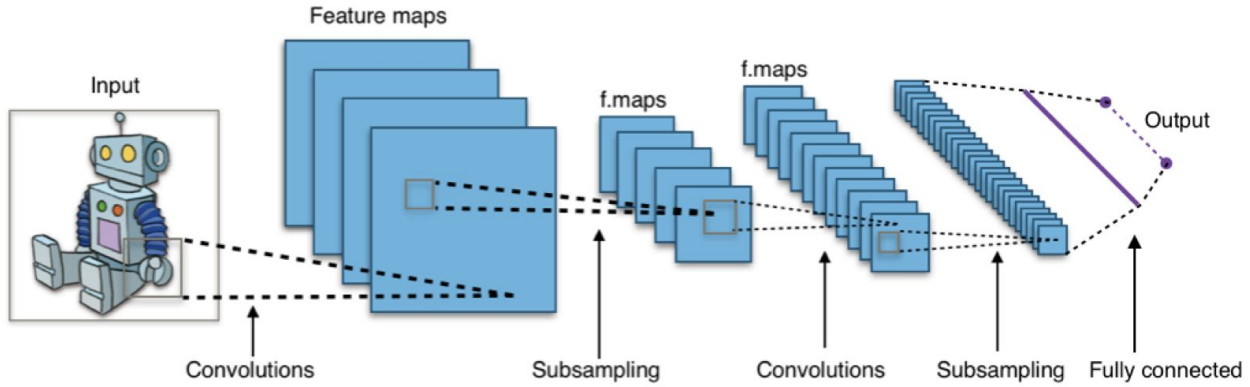
2	3	3
3	7	3
8	10	-3

o[:,:,1]

-8	-8	-3
-3	1	0
-3	-8	-5

toggle movement

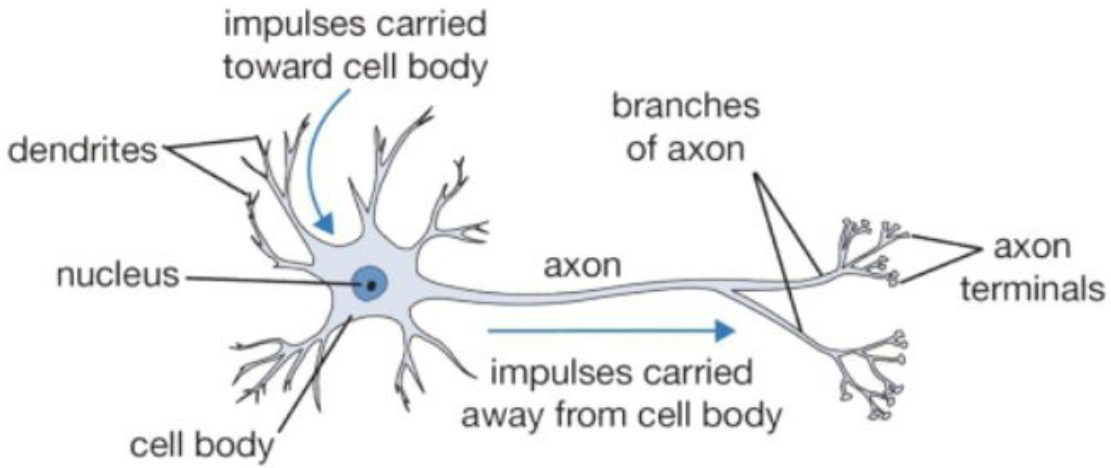
Architecture générale d'un CNN :



Les Réseaux de Neurones Convolutionnels

Avantages

- Demande moins de mémoire
- Demande moins de puissance de calcul
- Invariants sur les décalage d'image car tous les pixels ont le même poids dans le réseau neuronal
- Les réseaux de neurones sont adaptés via des boucles retour ce qui permet d'optimiser le résultat



Implémentation

Construction de la base de données exploitables

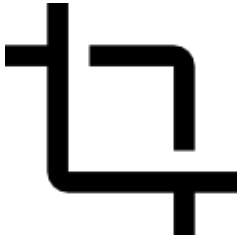
Scraping

Python
Google Images

Sélection

Type de fichier
Contraste important
Une classe unique

Formattage

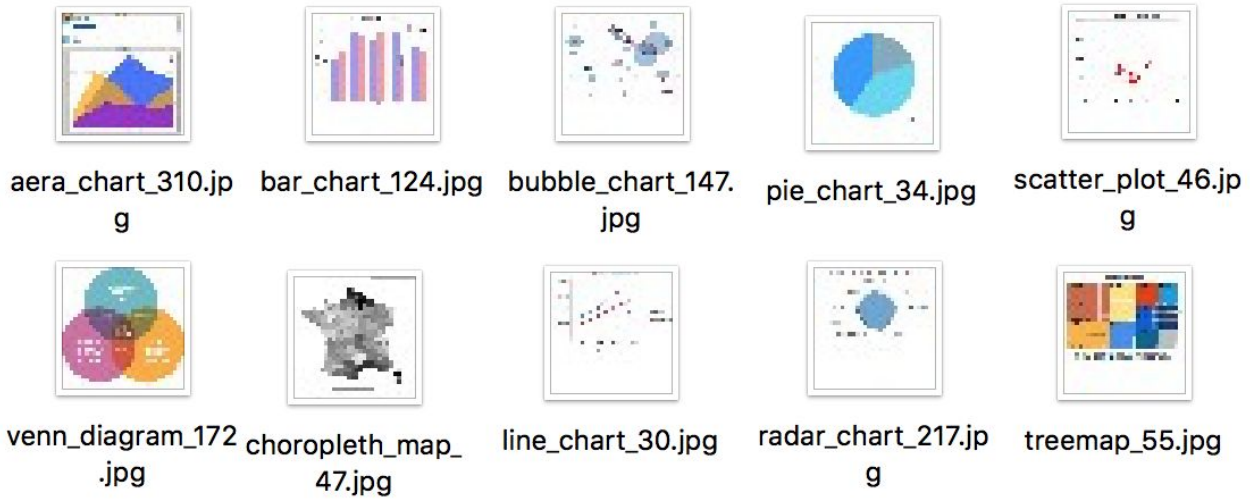


Complétion des images pour un format unique

Construction de la base de données exploitables

Caractéristiques de la BDD

- **3200 images** en format JPG
- **10 catégories** différentes : Line Chart, Bar Chart, Scatter Plot, Aera Chart, Bubble chart, Choropleth map, Pie Chart, Radar Chart, Treemap, Venn Diagram
- Nommage des images : nom_class_id.jpg
- Image au **même format** : 64x64 par défaut



- Divisées en deux après mélange: par défaut, **90% pour l'entraînement - cross-validation** du modèle et **10% images pour le test** du modèle

Construction de la base de données exploitables

Constitution et sérialisation de nos 2 tableaux d'images (entraînement, validation) associés à 2 tableaux pour les classes respectives des images à partir du script Python build_dataset.py

Par défaut, offset_train_val = 0.7

```
# Divide the data into offset% train, offset% validation
train_addrs = train_val_addrs[0:int(offset_train_val * len(train_val_addrs))]
train_labels = train_val_labels[0:int(offset_train_val * len(train_val_labels))]
validation_addrs = train_val_addrs[int(offset_train_val * len(train_val_addrs)):]
validation_labels = train_val_labels[int(offset_train_val * len(train_val_labels)):]

# Create a list of image array for the training dataset
for addr in train_addrs:
    img = cv2.imread(addr)
    X.append(img)
```

```
with open('dataset.pkl', 'wb') as f:
    #train_set, valid_set with images array and label array
    pickle.dump((X_train,Y_train,X_val_resized,Y_val_resized),f)
```

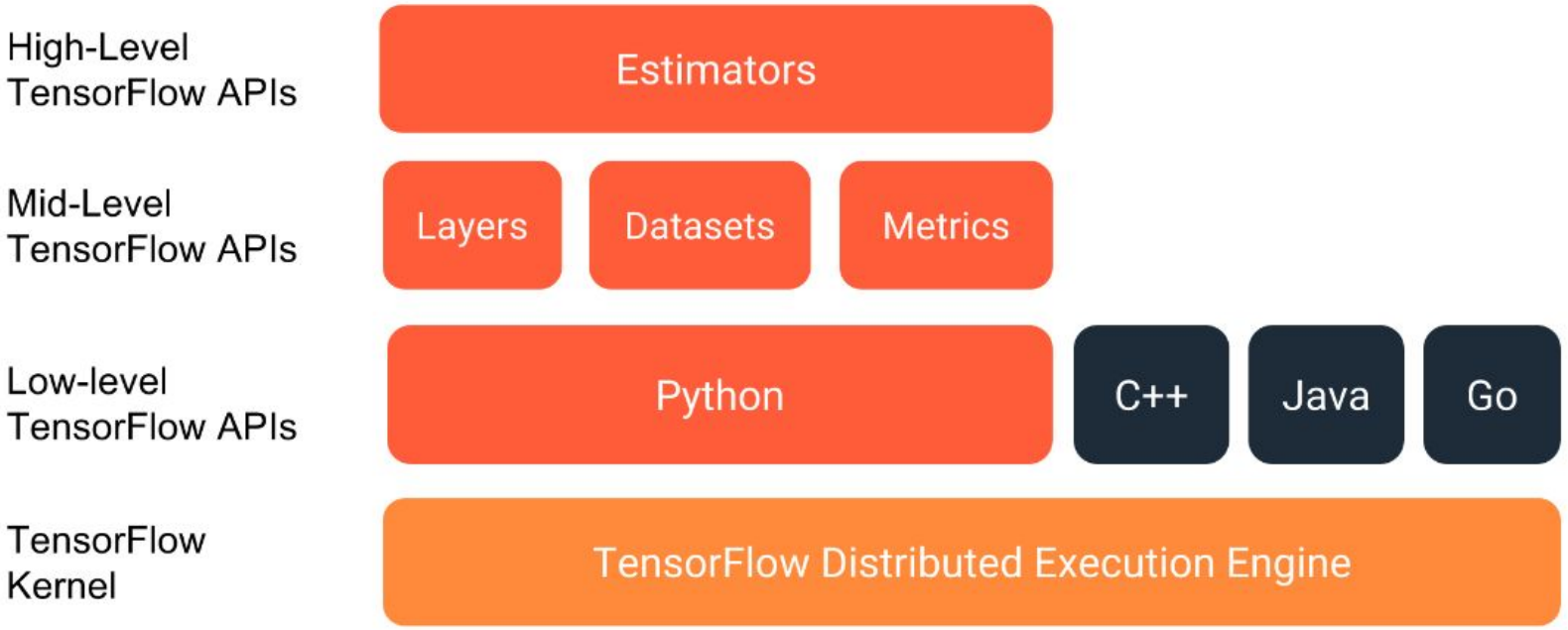
➔ **Fichier dataset.pkl prêt à être utilisé pour entraîner notre modèle de réseaux de neurones**

➔ **Dossier /test prêt à tester le modèle entraîné**

Présentation de Tensor Flow

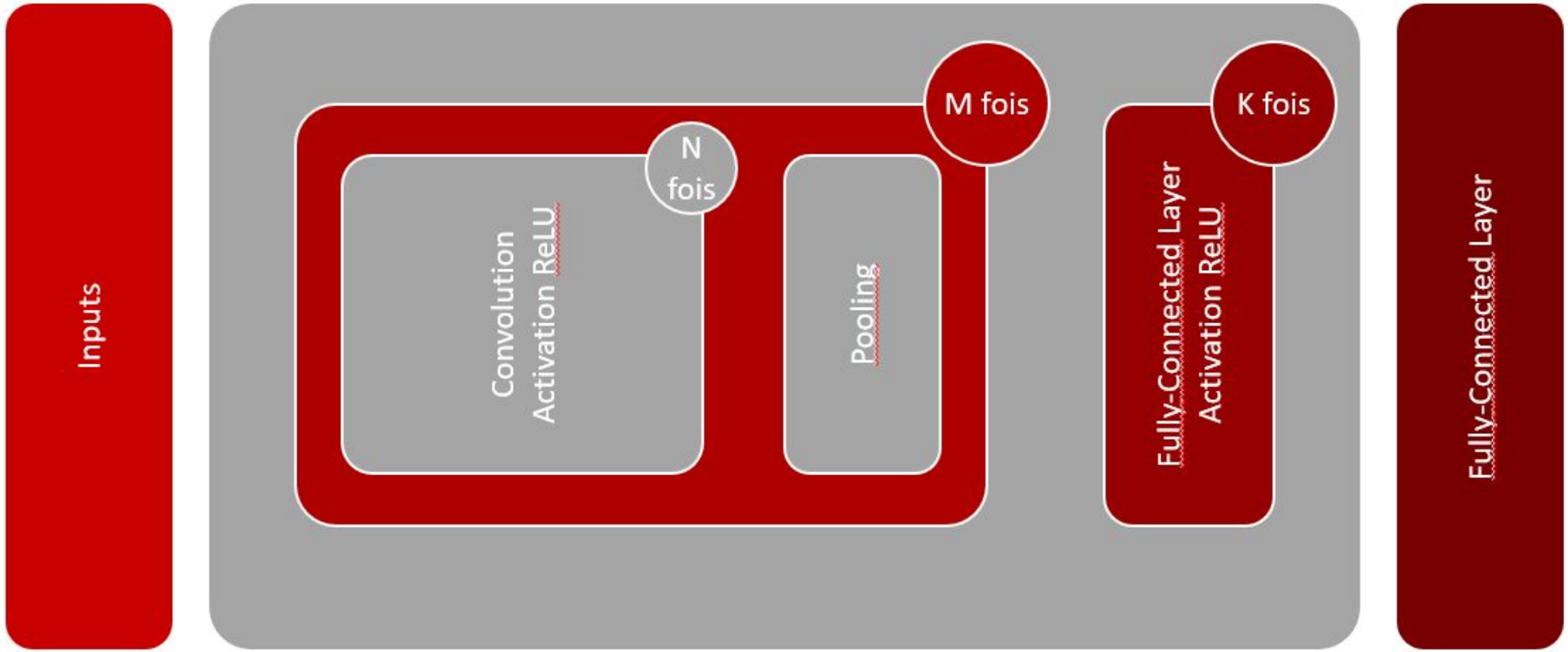
Framework open-source de Machine Learning

- Développé par Google et publié en novembre 2015
- Dispose de plusieurs APIs de plus en plus complexes permettant de faire évoluer notre algorithme
- Forte communauté et Framework reconnu (~ 90000 étoiles sur Github)



Architecture du réseau

Structure Globale

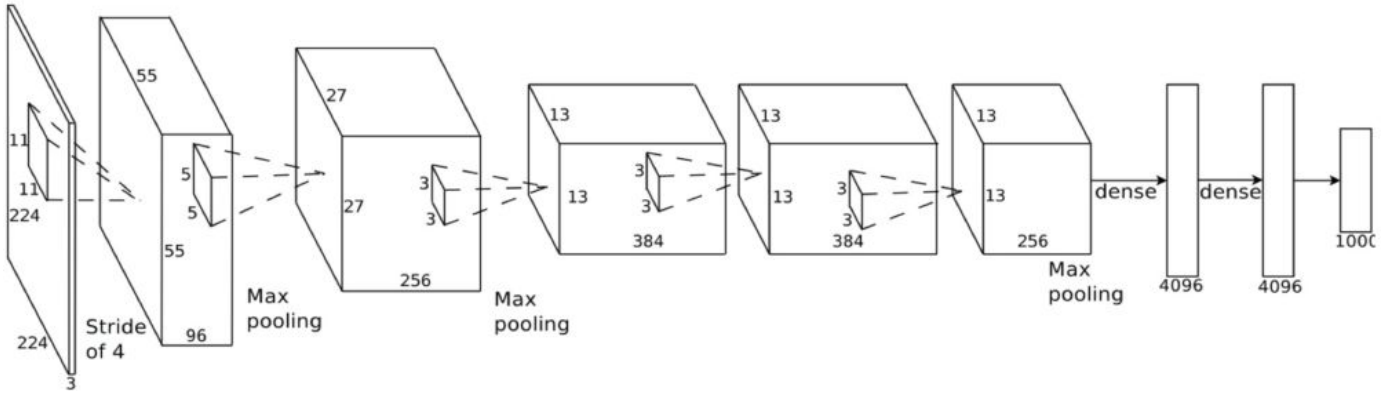


Architecture du réseau

Cinq réseaux

	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10
Réseau 1	Convol (32,3)	Max-pool (2)	Convol (64,3)	Convol (64,3)	Max-pool (2)	FC (256)	Dropout (50%)	FC		
Réseau 2	Convol (32,3)	Convol (32,3)	Max-pool (2)	Convol (32,3)	Convol (32,3)	Max-pool (2)	FC (512)	FC (512)	FC	
Réseau 3	Convol (32,3)	Average-pooling (2)	Convol (32,3)	Average-pooling (2)	Convol (32,3)	Average-pooling (2)	FC (512)	FC (512)	Dropout (50%)	FC
Réseau 4	Convol (32,3)	Convol (32,3)	Convol (32,5, 0 pad)	Convol (32,3)	Convol (32,3)	Convol (32,5, 0 pad)	FC (512)	Dropout (50%)	FC	
Réseau 5	Convol (64,3)	Convol (64,3)	Average-pooling (2)	Convol (32,3)	Convol (32,3)	Max-pool (2)	FC (512)	FC (512)	FC	

Entraînement du réseau de neurones



Caractéristiques communes :

- Shuffling : mélange aléatoire
- Data Preprocessing : normalisation des données
- Data Augmentation : rotation, flip, blur des images
- Dropout : neurones non pris en compte, meilleure généralisation du modèle

Éviter le sur-apprentissage

Entraînement du réseau de neurones

Paramètres :

- Nombre d'epoch : itérations sur le jeu de données en entraînement
- Batch size : nombre d'images prises pour chaque propagation
- Learning rate : paramètre gradient descendant
- Nombre de filtres : nombre de neurones
- Choix du réseau

Entraînement du réseau choisi :

```
model = re.getReseau()
model.fit(X, Y, n_epoch=settings.nb_epoch, shuffle=True, validation_set=(X_test, Y_test),
          show_metric=True, batch_size=settings.batch_size,
          snapshot_epoch=True)
model.save("dataviz-classifier.tfl")
```

Le modèle entraîné est enregistré

Résultats

Résultats - Les métriques

Script prediction.py pour tester le modèle sur l'ensemble test créé au préalable

```
for index, addr in enumerate(addrs):
    img = cv2.imread(addr).astype(np.float32, casting='unsafe')
    # Predict with the trained model
    prediction = model.predict([img])
```

➢ **prediction = Tableau contenant les probabilités d'appartenir à une classe : normalisation de la fonction Score**

Exemple avec 3 classes : **[0.81, 0.08, 0.11]**

➢ **Affectation de l'image à la première classe**

Matrice de confusion

```
"confusion": [
  [32.0, 2.0, 2.0, 0.0, 2.0, 2.0],
  [3.0, 32.0, 2.0, 0.0, 1.0, 0.0],
  [2.0, 1.0, 29.0, 0.0, 4.0, 4.0],
  [3.0, 0.0, 0.0, 34.0, 1.0, 1.0],
  [0.0, 1.0, 2.0, 2.0, 31.0, 3.0],
  [0.0, 2.0, 2.0, 3.0, 1.0, 30.0]
```

Résultats - Les métriques

Calcul de métriques pour analyser le modèle

➢ **Rappel & Précision pour chaque classe + Moyenne + Accuracy**

$$Précision = \frac{Le\ nombre\ d'images\ bien\ classées}{Le\ nombre\ total\ d'images\ prédites\ pour\ cette\ classe}$$

$$Recall = \frac{Le\ nombre\ d'images\ bien\ classées}{Le\ nombre\ total\ d'images\ de\ la\ classe\ dans\ la\ BDD}$$

$$Accuracy = \frac{Le\ nombre\ d'images\ bien\ classées\ toutes\ classes\ confondues}{Le\ nombre\ total\ d'images\ dans\ la\ BDD}$$

➢ **JSON créé, contient :**

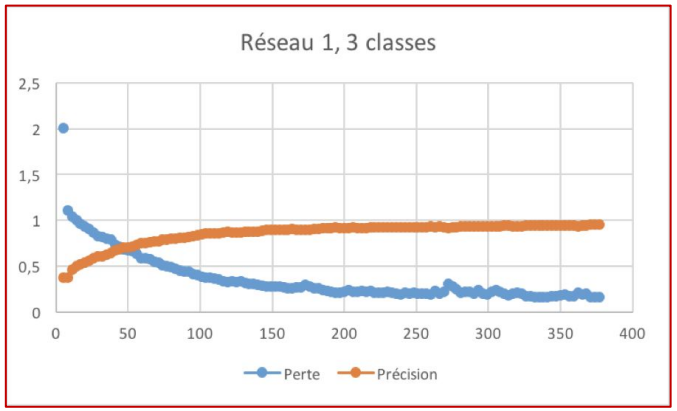

```
"pie_chart": [{
  "precision": 0.9523809523809523,
  "recall": 0.8888888888888888
```


- Les probabilités pour chaque image
- La matrice de confusion
- Les métriques pour chaque classe et globales
- L'ensemble des paramètres/hyperparamètres : structure réseau utilisé, epoch, batch_size, etc.

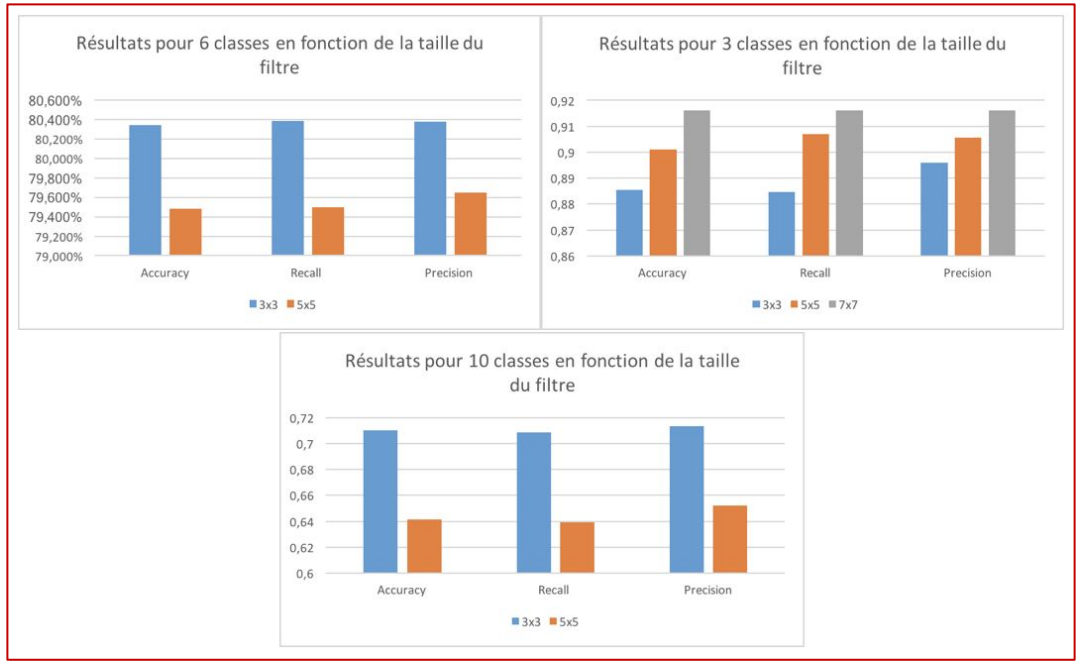
Résultats - Analyse

Choix des paramètres

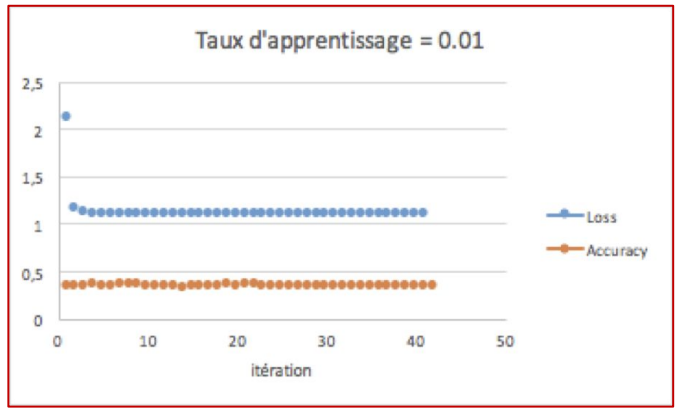
Epochs



Taille du filtre

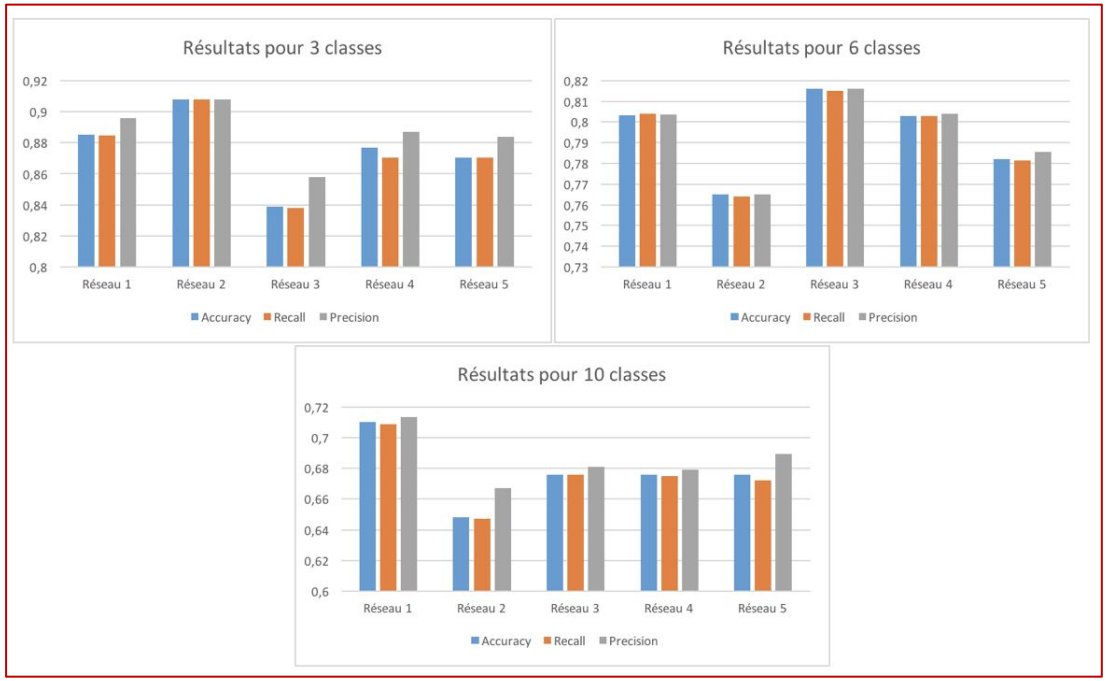


Learning Rate

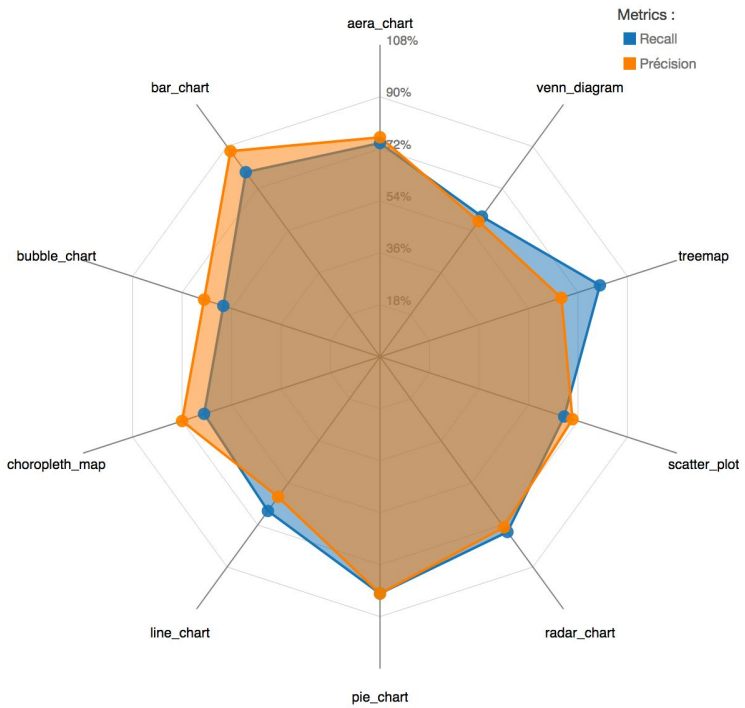


Résultats - Analyse

Choix du réseau

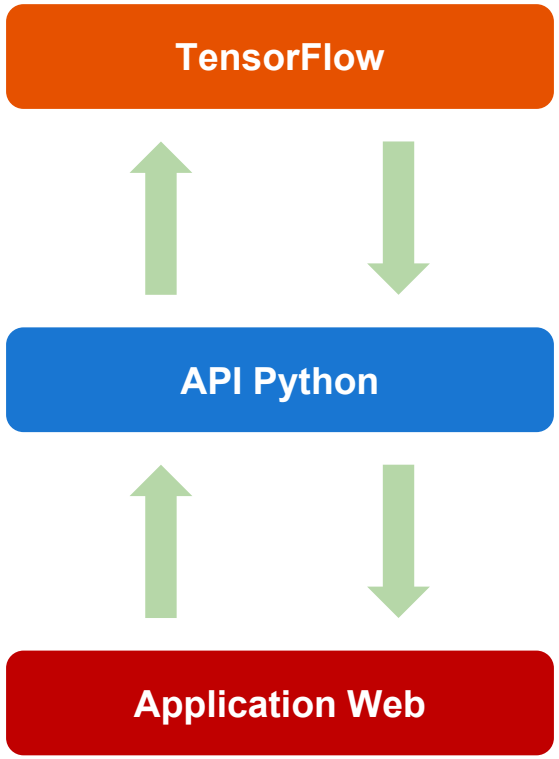


	3 classes	6 classes	10 classes
Réseau	**1**	**3**	**1**
Taille des filtres	7x7	3x3	3x3
Taille des images	64x64	64x64	64x64
Taux d'apprentissage	0.001	0.001	0.001
Nombre d'itérations	125	125	125
Accuracy	91,60%	81,60%	71,03%



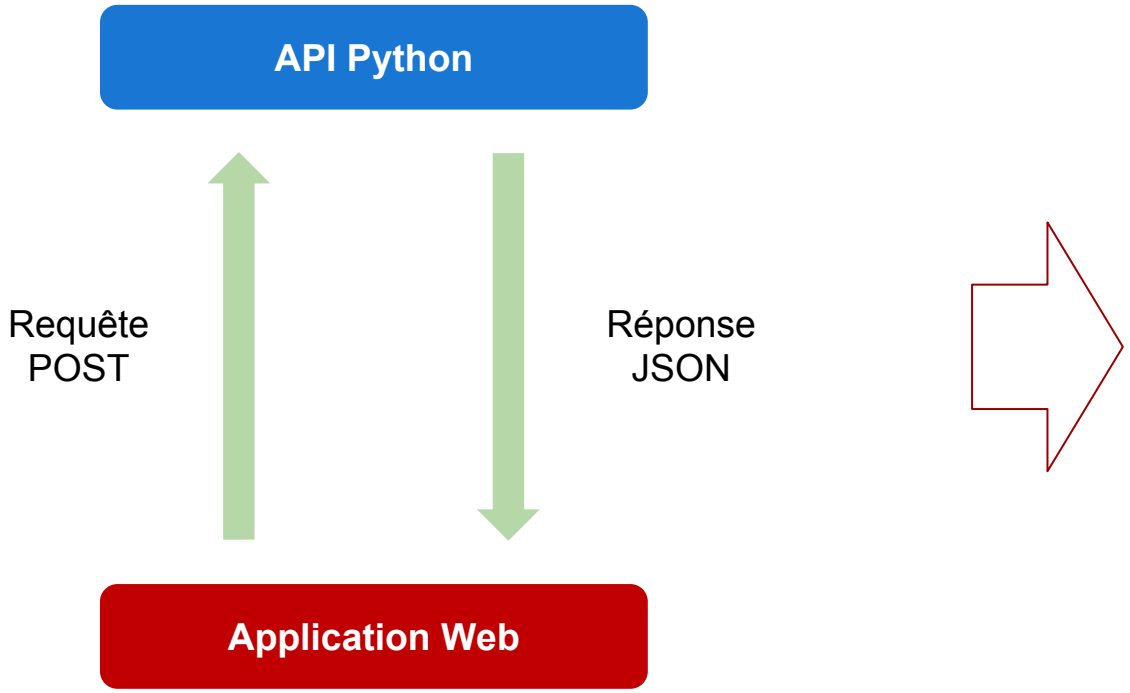
Visualisation des résultats

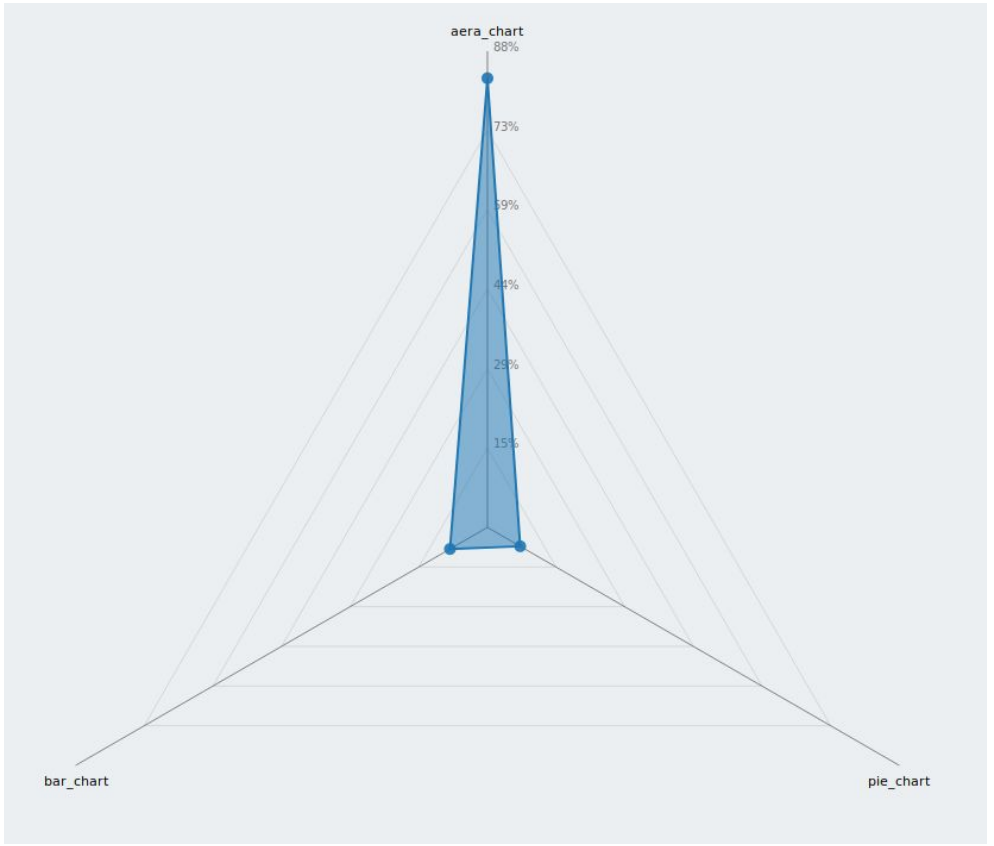
Architecture de l'application



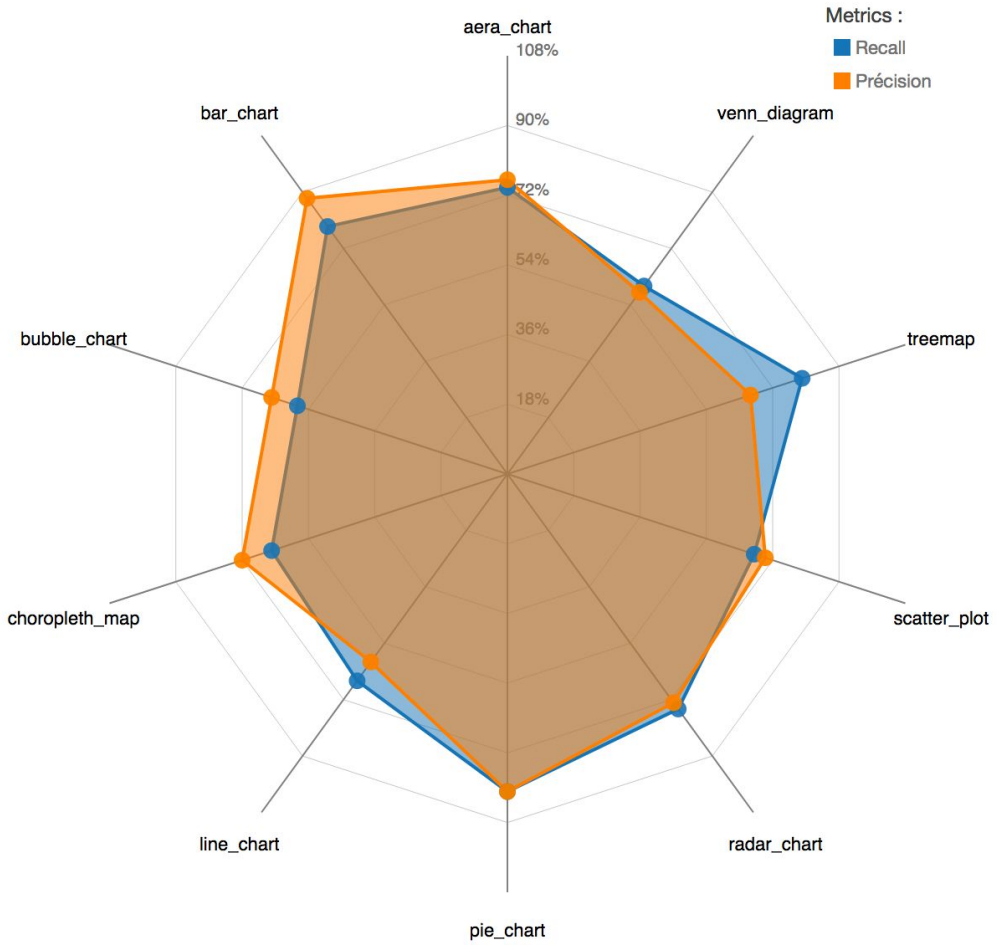
Visualisation des résultats

Fonctionnalité de l'application





Visualisation des résultats



Merci pour votre attention !