
Collecte de Données et Classification de Data Visualization

Soutenance finale

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



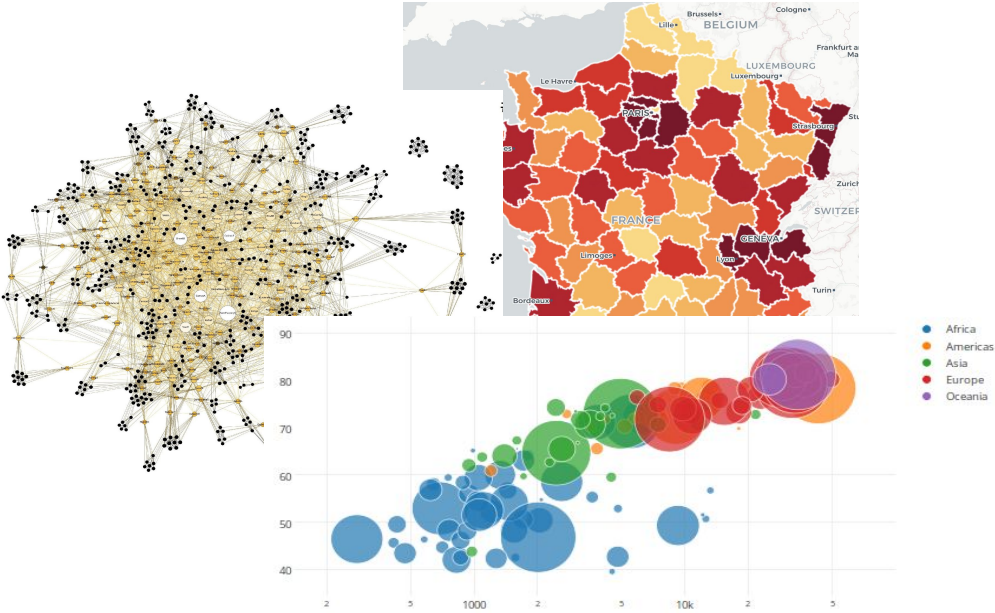
Arnaud BRUGIERE

Contexte & Objectifs

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

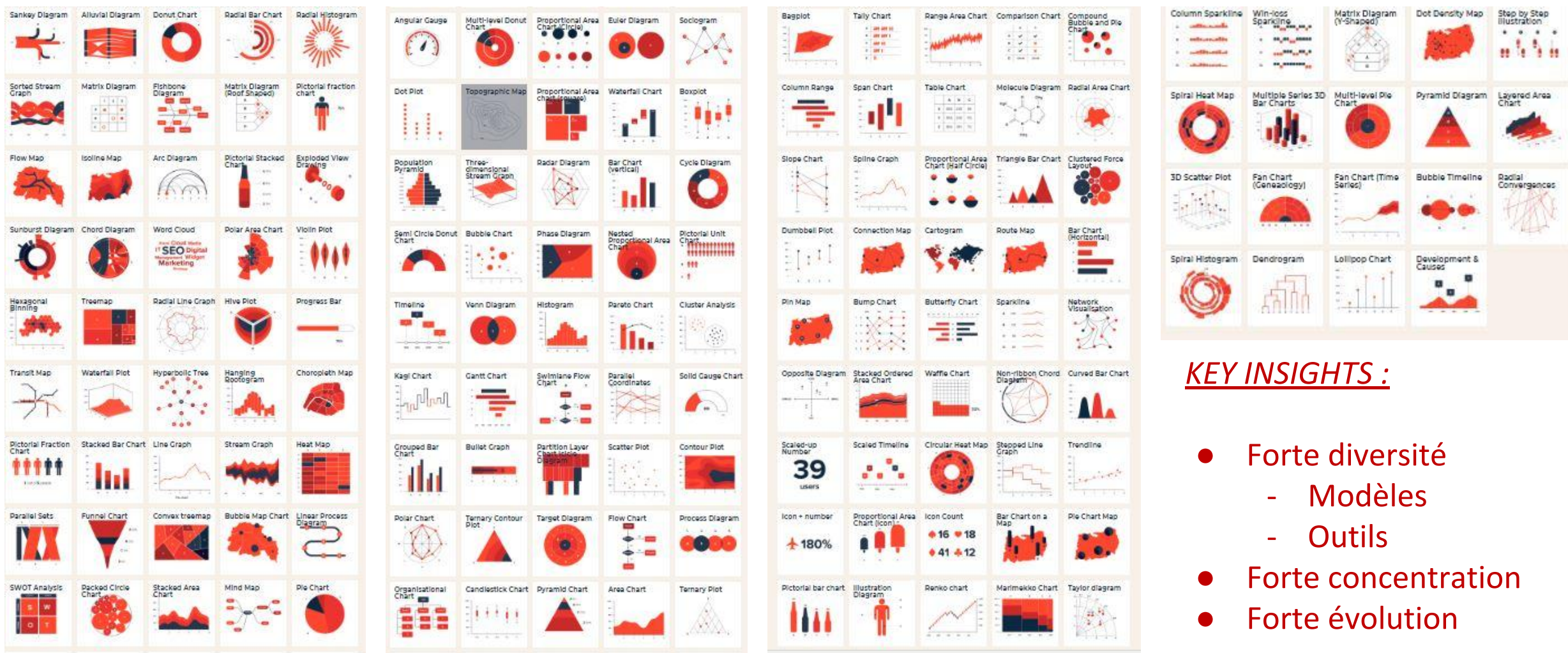
Revenu fiscal de référence par tranche (en euros)	Nombre de références des foyers fiscaux	Revenu fiscal de référence des foyers fiscaux	Impôt net (total)*	Nombre de références des foyers fiscaux imposés	Revenu fiscal de référence des foyers fiscaux imposés
0 à 10 000	6 778 578	37 017 353	-120 471	64 840	286 206
10 001 à 12 000	2 141 456	23 577 329	-52 668	8 831	83 719
12 001 à 15 000	3 415 487	48 459 161	-67 900	236 299	3 491 606
15 001 à 20 000	5 907 525	102 764 512	1 757 605	3 105 770	53 996 313
20 001 à 30 000	6 830 752	187 947 232	5 647 709	3 927 006	96 705 054
30 001 à 50 000	5 553 696	250 560 854	13 303 362	5 134 391	189 437 862
50 001 à 100 000	3 305 940	217 391 602	20 630 441	3 121 015	205 724 469
Plus de 100 000 euros	749 163	140 216 576	28 027 792	727 817	136 965 443
100 001 à 200 000	597 846	78 515 622	12 812 674	579 100	76 088 011
200 001 à 300 000	89 163	21 094 844	4 697 661	86 462	20 585 166
300 001 à 400 000	28 243	6 670 624	2 422 619	27 854	6 537 660
400 001 à 500 000	12 523	3 567 586	1 464 605	12 362	3 495 371
500 001 à 600 000	6 582	3 572 205	865 370	6 477	3 531 639
600 001 à 700 000	3 889	2 512 860	685 321	3 845	2 484 572
700 001 à 800 000	2 456	1 632 359	511 034	2 433	1 616 159
800 001 à 900 000	1 709	1 445 636	405 219	1 697	1 435 610
900 001 à 1 000 000	1 247	1 181 172	339 407	n.c.	n.c.
1 000 001 à 2 000 000	4 463	6 045 093	1 605 428	4 420	5 981 102

A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	code postal	adresse	article	ville	article	ville	region	nom region	dep	nom dep	longitude	latitude	code	metaphone
2	01000	'01004'	'Bugey'	'AMBERIEU'	'Bugey'	'AMBERIEU EN BUGY'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.97881'	'5.33689'	'A516'	'AMPRNPJ'
3	01330	'01003'	'Ambâ'	'Grieu-en-Dombes'	'AMBERIEUX-EN-DOBES'	'AMBERIEUX EN DOBES'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.99803'	'4.30249'	'A510'	'AMPRKNTMP'
4	01300	'01008'	'AMBI&'	'Con'	'AMBLEON'	'AMBLEON'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.74967'	'5.60132'	'A514'	'AMPLFV'
5	01500	'01007'	'Ambérieux'	'AMBERIEUX'	'AMBERIEUX'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.00648'	'5.35099'	'A513'	'AMPRKX'	
6	01500	'01008'	'Ambutrix'	'AMBITRIX'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.93594'	'5.338195'	'A513'	'AMPRKX'		
7	01300	'01009'	'Andert-et-Condor'	'ANDERT ET CONDON'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.79468'	'5.655624'	'A536'	'ANTRTKNTN'		
8	01500	'01010'	'Angletfort'	'ANGLETFORT'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.81331'	'5.80899'	'A534'	'ANKUFT'		
9	01100	'01011'	'Apremont'	'APREMONT'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.207172'	'5.657787'	'A165'	'APREMT'		
10	01110	'01012'	'Aranc'	'ARANC'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.003679'	'5.508627'	'A632'	'ARNK'		
11	01210	'01013'	'Arandaz'	'ARANDAZ'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.85096'	'5.485735'	'A633'	'ARNTS'		
12	01100	'01014'	'Arbent'	'ARBENT'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.295735'	'5.6821'	'A615'	'ARPN'		
13	01300	'01015'	'Arbigneu'	'ARBIGNEUX'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'45.72482'	'5.650341'	'A612'	'ARPN'		
14	01180	'01016'	'Arbigny'	'ARBIGNY'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
15	01230	'01017'	'Argis'	'ARGIS'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
16	01310	'01019'	'Armix'	'ARMIX'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
17	01480	'01021'	'Ary-su'	'ARY-SU'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
18	01510	'01022'	'Artem'	'ARTEM'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
19	01570	'01023'	'Asna'	'ASNA'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
20	01340	'01024'	'Atinge'	'ATINGE'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		
21	01380	'01025'	'B&C&'	'B&C&'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.295735'	'5.6821'	'A615'	'ARPN'		
22	01380	'01026'	'B&C&'	'B&C&'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.295735'	'5.6821'	'A615'	'ARPN'		
23	01380	'01027'	'B&C&'	'B&C&'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.295735'	'5.6821'	'A615'	'ARPN'		
24	01390	'01028'	'B&C&'	'B&C&'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.295735'	'5.6821'	'A615'	'ARPN'		
25	01270	'01029'	'Beaup'	'BEAUP'	'B2'	'RHONE-ALPES'	'01'	'Ain'	'46.49366'	'4.361984'	'A612'	'ARPN'		



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 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 5 : Généralisation

Extension jusqu'à 10 classes de visualisation de données

Phase 2 : Détermination et construction algorithme

Choix et architecture de notre premier CNN

Phase 4 : Itération sur l'algorithme

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



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

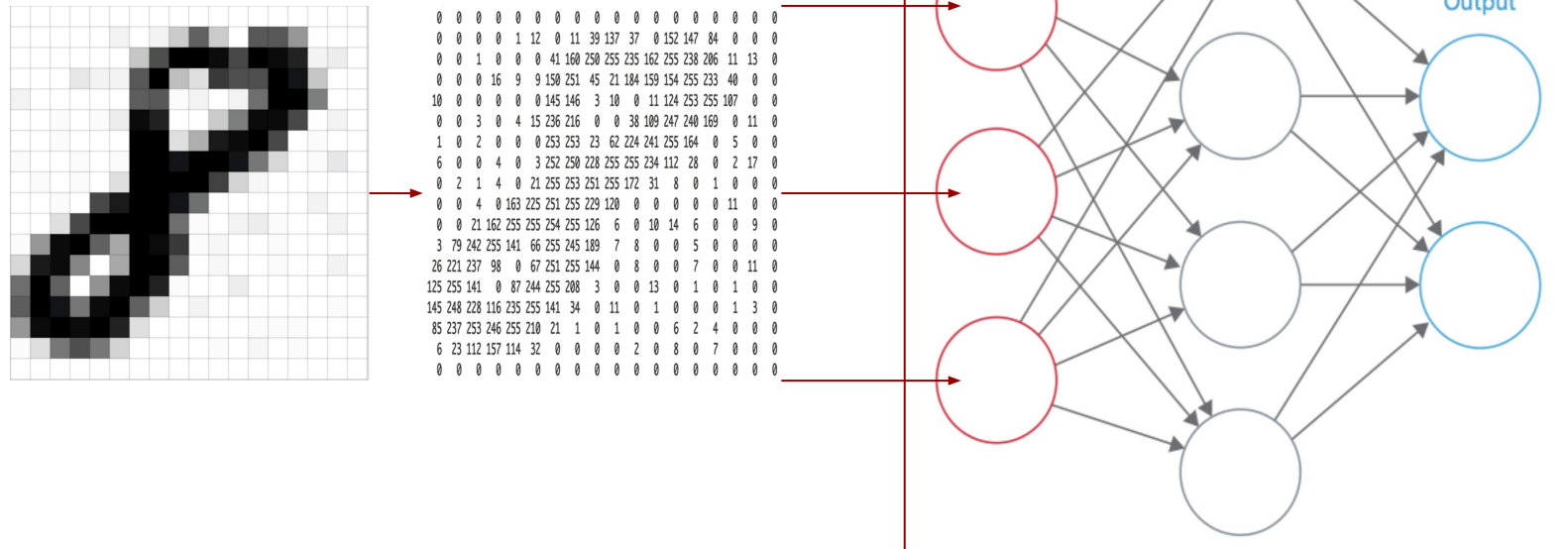
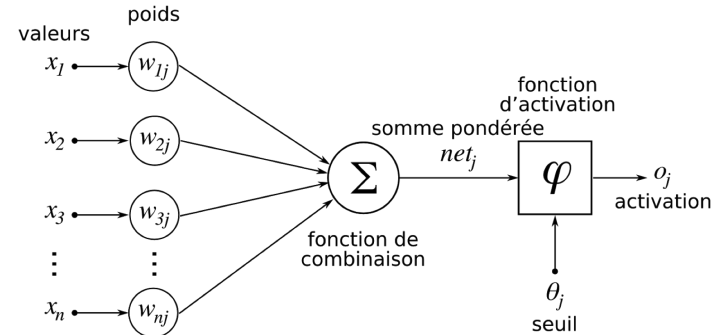
LIVRABLES POUVANT FAIRE L'OBJET DE PROCHAINES ÉTAPES :

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

Les réseaux de neurones

Les Réseaux de Neurons

Fonctionnement d'un réseau de neurones :



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

Les Réseaux de Neurons Convolutionnels

Fonctionnement d'une couche de Convolution :

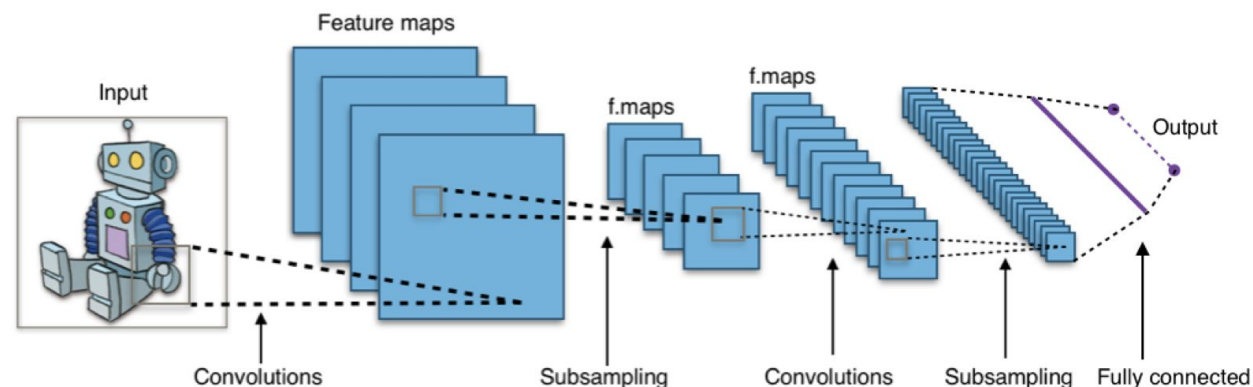
Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
$x[:, :, 0]$	$w0[:, :, 0]$	$w1[:, :, 0]$	$o[:, :, 0]$
0 0 0 0 0 0 0	-1 0 1	0 1 -1	2 3 3
0 0 0 1 0 2 0	0 0 1	0 -1 0	3 7 3
0 1 0 2 0 1 0	1 -1 1	0 -1 1	8 10 -3
0 1 0 2 2 0 0	$w0[:, :, 1]$	$w1[:, :, 1]$	$o[:, :, 1]$
0 2 0 0 2 0 0	-1 0 1	-1 0 0	-8 -8 -3
0 2 1 2 2 0 0	1 -1 1	1 -1 0	-3 1 0
0 0 0 0 0 0 0	0 1 0	1 -1 0	-3 -8 -5
$x[:, :, 1]$	$w0[:, :, 2]$	$w1[:, :, 2]$	
0 0 0 0 0 0 0	-1 1 1	-1 1 -1	
0 2 1 2 1 1 0	1 1 0	0 -1 -1	
0 2 1 2 0 1 0	0 -1 0	1 0 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			

Bias $b0$ (1x1x1)
 $b0[:, :, 0]$
 1

Bias $b1$ (1x1x1)
 $b1[:, :, 0]$
 0

toggle movement

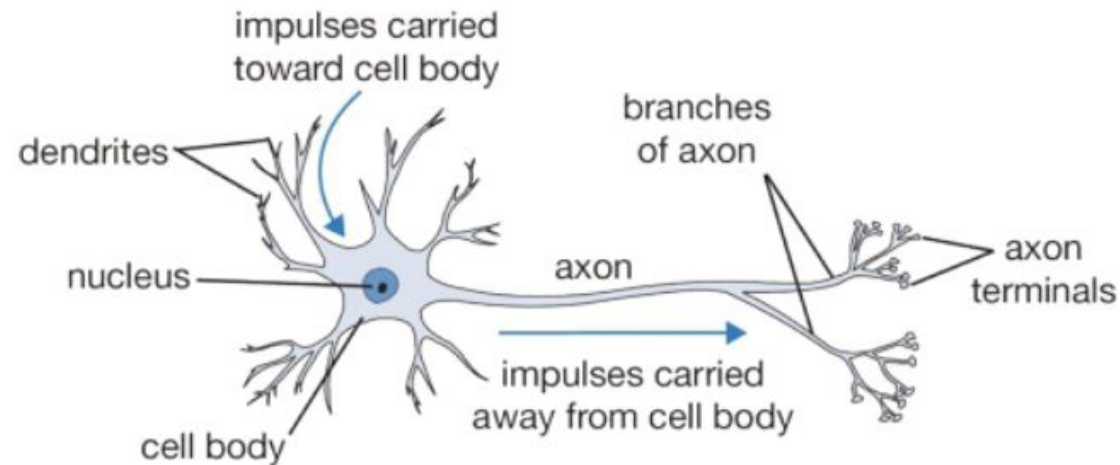
Architecture générale d'un CNN :



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



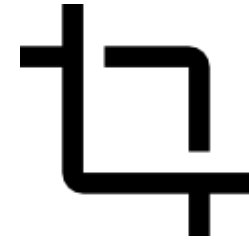
Python
Google Images

Sélection



Type de fichier
Contraste important
Une classe unique

Formattage

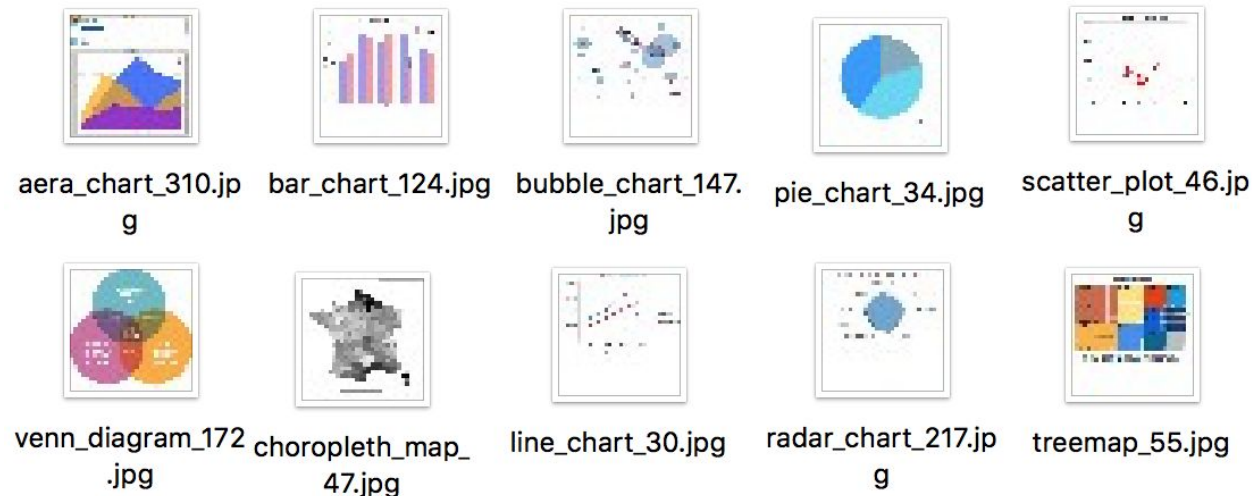


Complétion des
images pour un
format unique

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, Area 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)
```

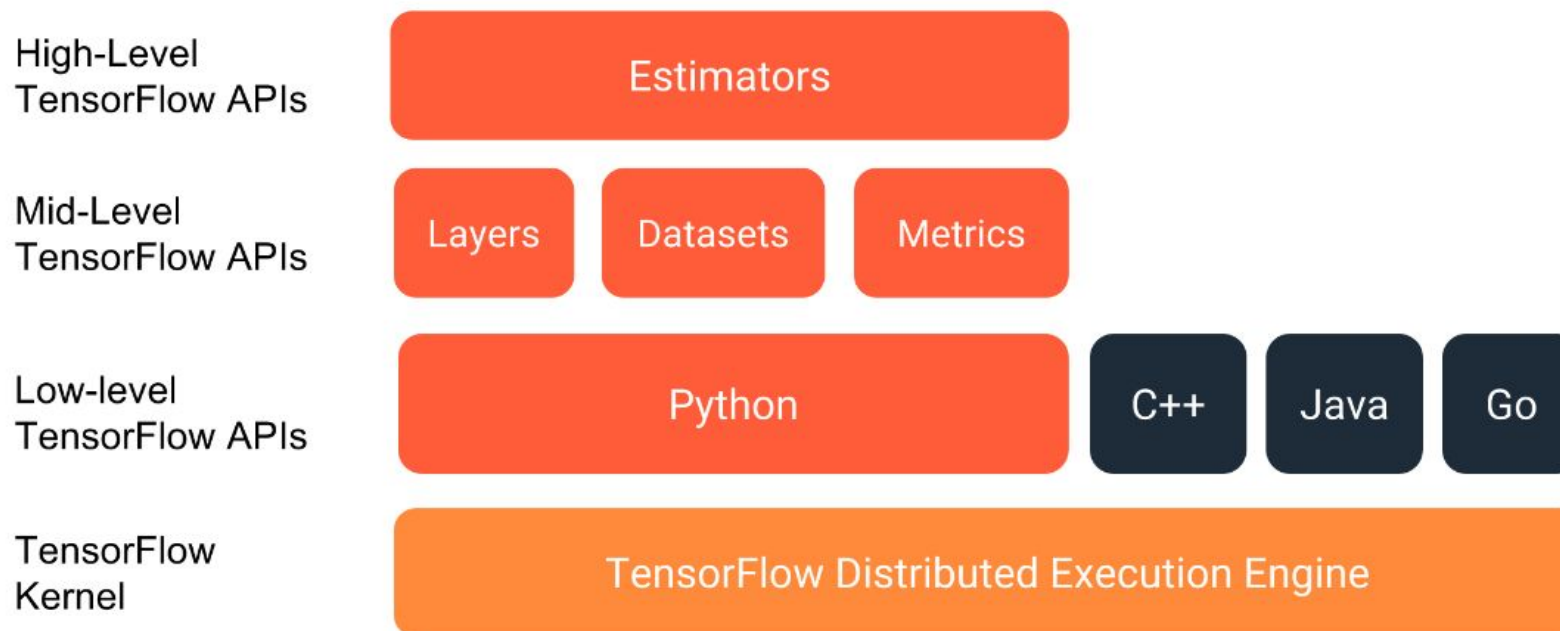
```
with open('dataset.pkl', 'wb') as f:
    #train set, valid set with images array and label array
    pickle.dump((X_train, Y_train, X_val_resized, Y_val_resized), f)
```

- 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

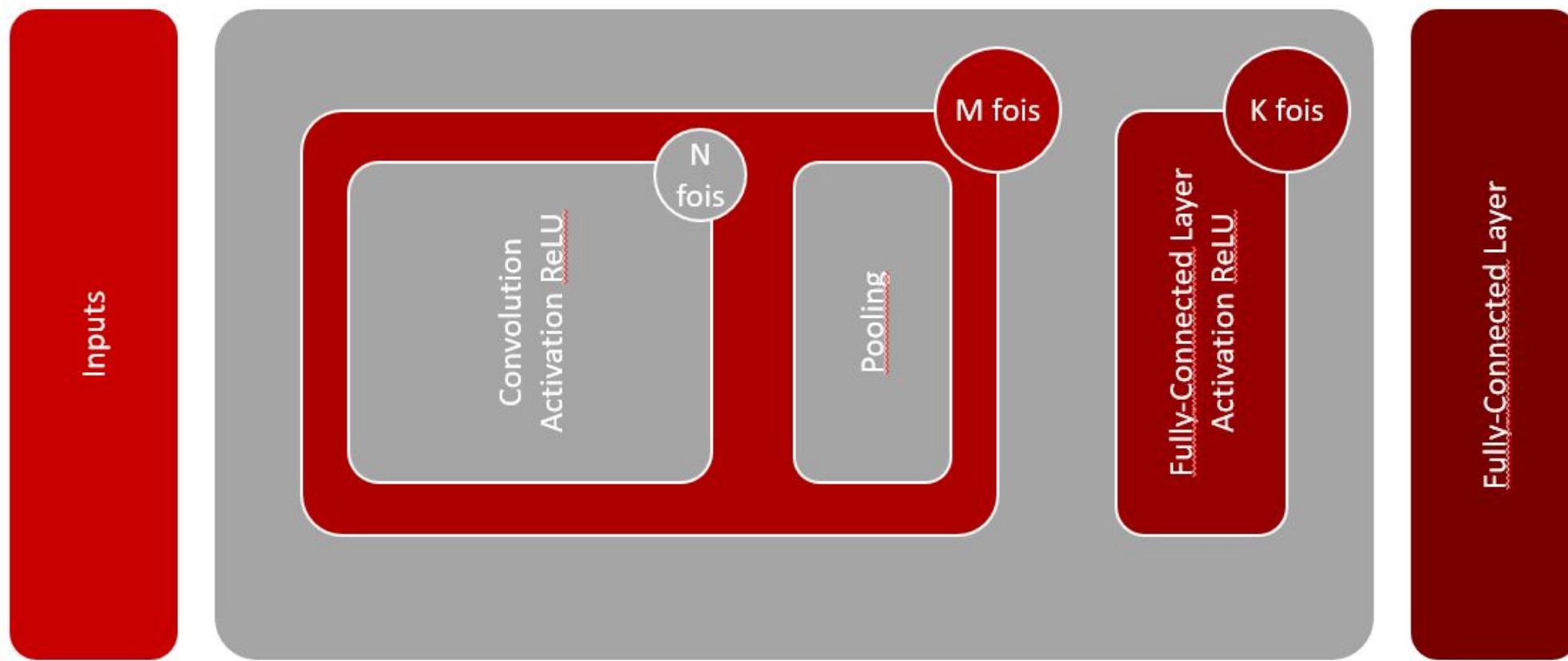
Framework open-source de Machine Learning

- 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

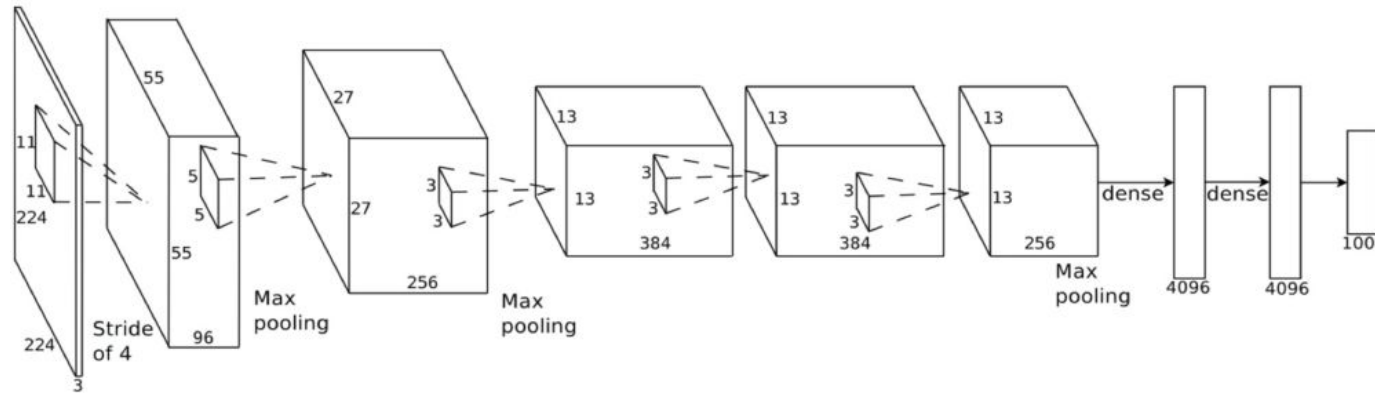


Architecture du réseau

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'époch : 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(addr):  
    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

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

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

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

➤ JSON créé, contient :

- 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.

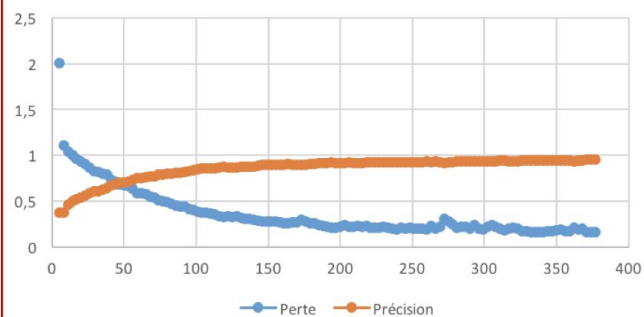
```
"pie_chart": [{  
  "precision": 0.9523809523809523,  
  "recall": 0.8888888888888888
```

Résultats - Analyse

Choix des paramètres

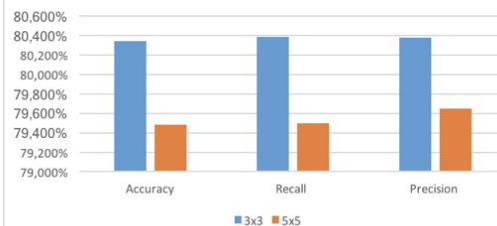
Epochs

Réseau 1, 3 classes

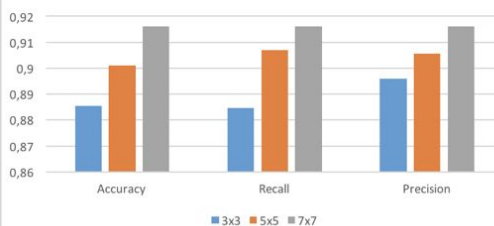


Taille du filtre

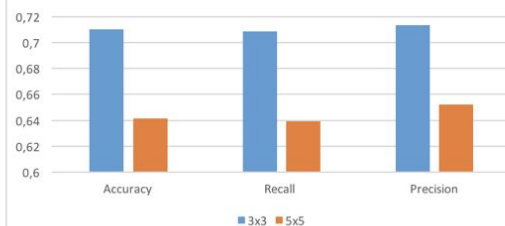
Résultats pour 6 classes en fonction de la taille du filtre



Résultats pour 3 classes en fonction de la taille du filtre

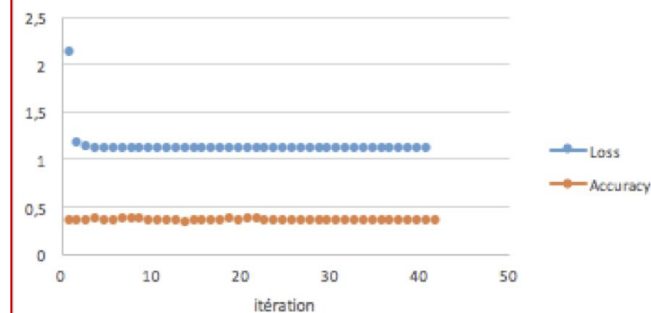


Résultats pour 10 classes en fonction de la taille du filtre



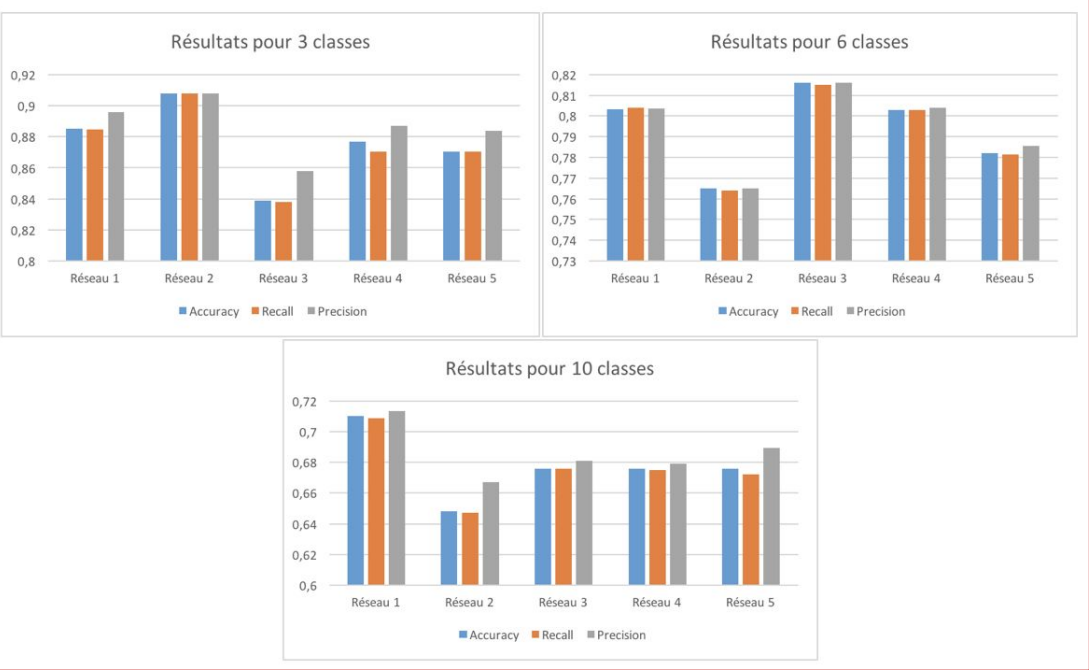
Learning Rate

Taux d'apprentissage = 0.01

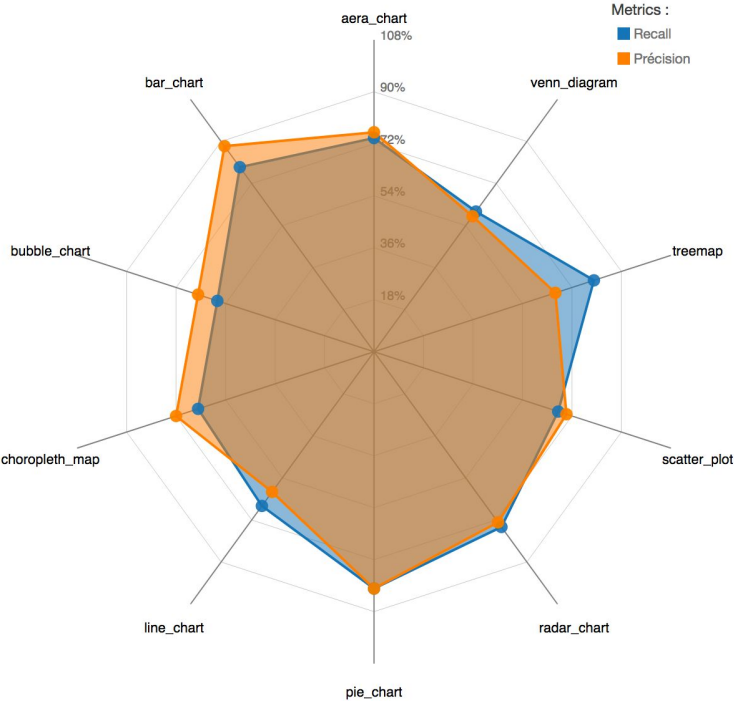


Résultats - Analyse

Choix du réseau

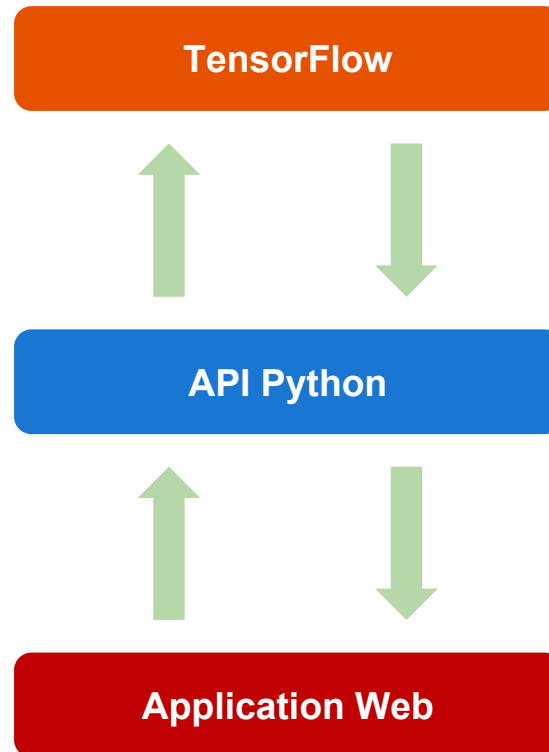


	3 classes	6 classes	10 classes
Réseau	1	3	1
Taille des filtres	7x7	3x3	3x3
Taille des images	64x64	64x64	64x64
Taux d'apprentissage	0.001	0.001	0.001
Nombre d'itérations	125	125	125
Accuracy	91,60%	81,60%	71,03%



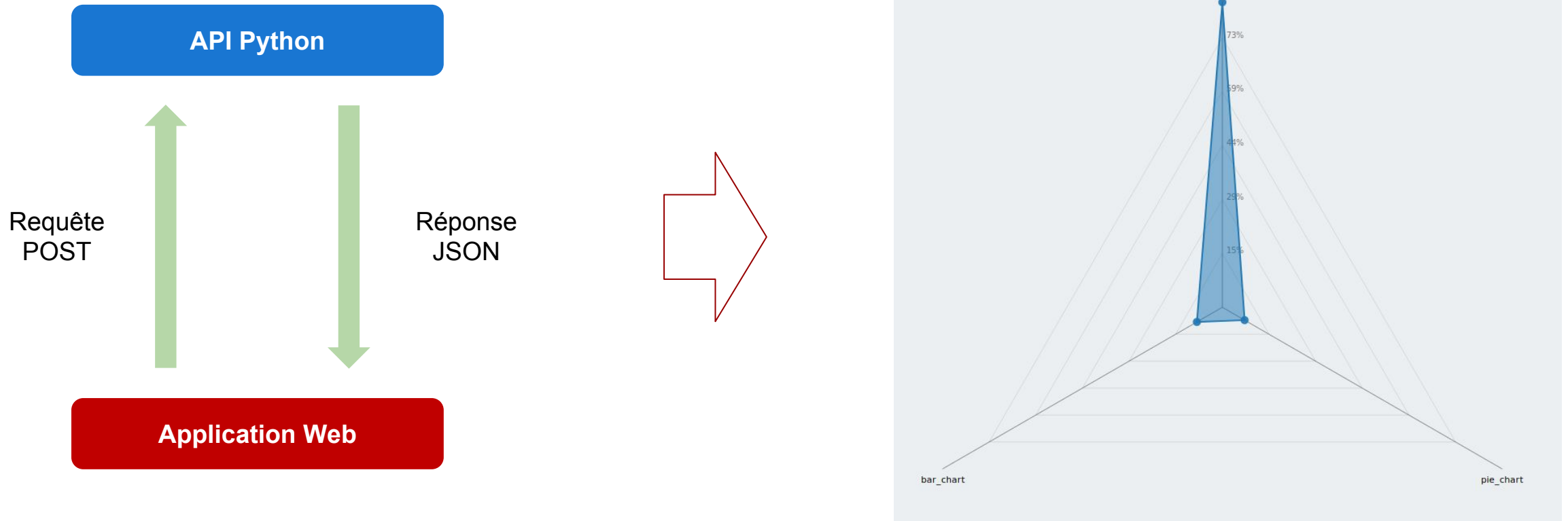
Visualisation des résultats

Architecture de l'application

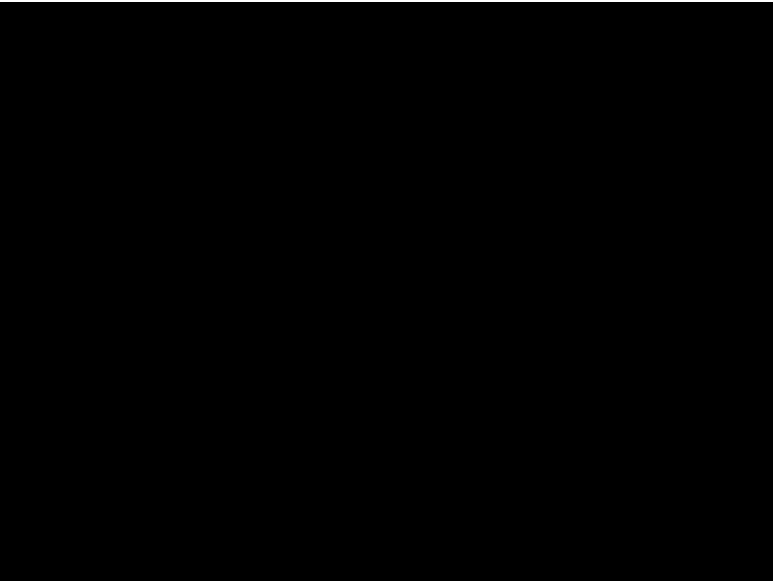
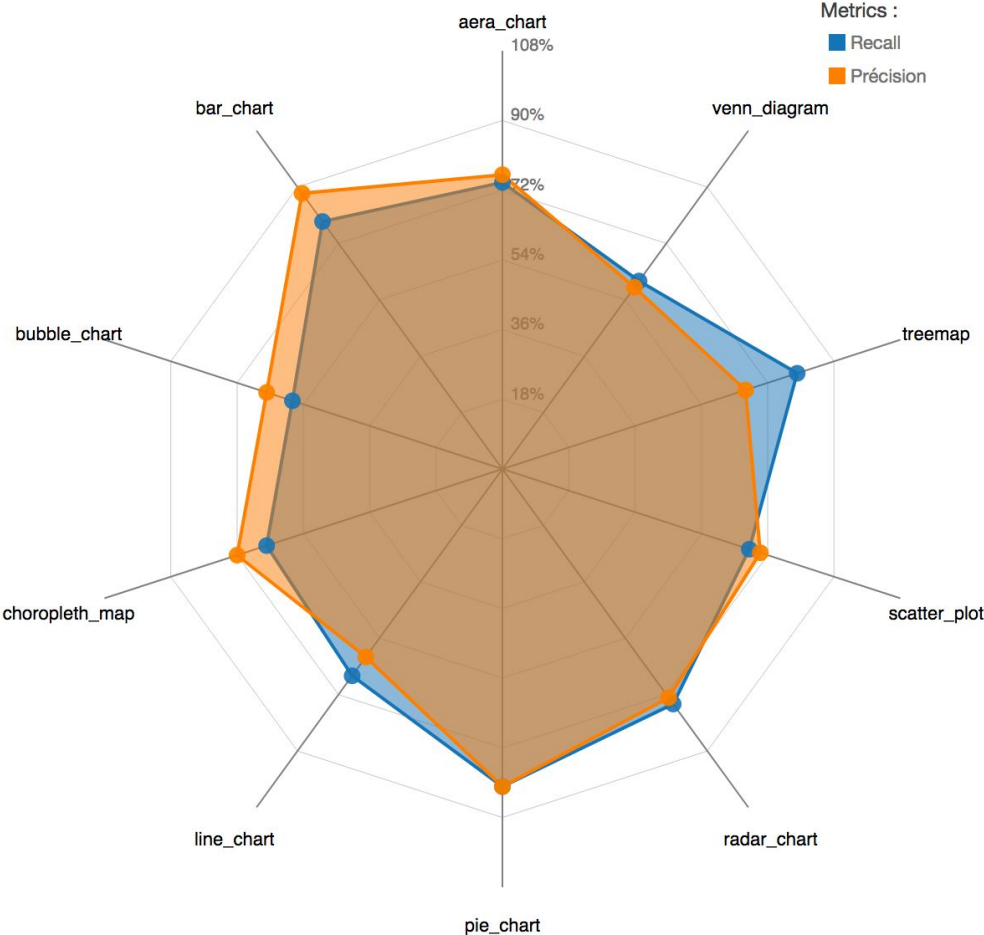


Visualisation des résultats

Fonctionnalité de l'application



Visualisation des résultats



Merci pour votre attention !
