

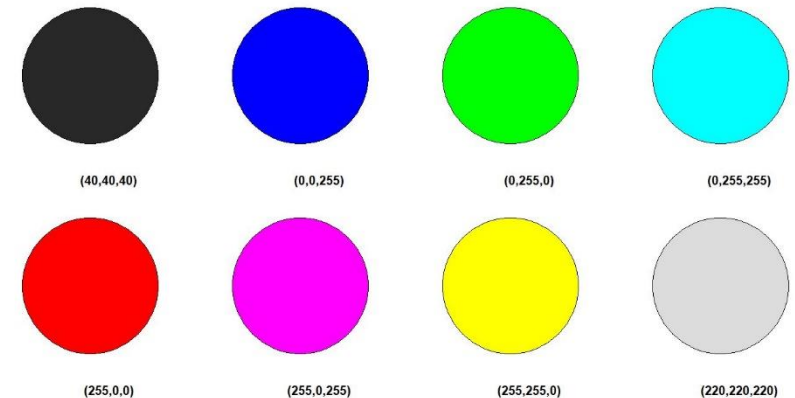
Utilisation d'un réseau de neurones entraîné  
pour différentes reconnaissances

Mécanismes d'apprentissage

Chris Arnault

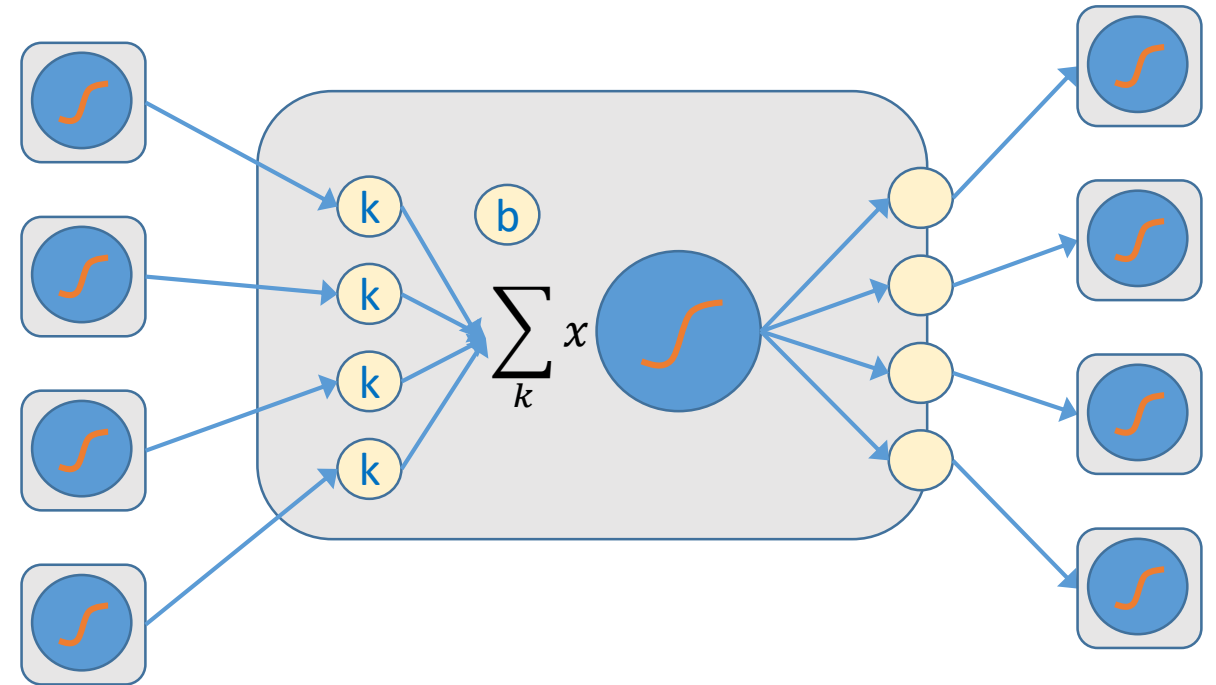
# Mécanisme de l'apprentissage

- Premier exemple: On veut entraîner un réseau de neurones (RdN) sur un ensemble de pastilles de couleurs
  - Chaque pastille est caractérisée par une valeur de RGB telle qu'un contraste existe entre toutes les pastilles, permettant de bien les discriminer
  - Un jeu, matérialisé par un véhicule-robot, est doté d'un capteur de couleurs RGB (cela peut être une caméra)
  - Une fois entraîné, le RdN sera interrogé pour demander à quelle pastille correspond une valeur RGB détectée



# Principe d'un réseau de neurones

- Un neurone contient:
  - **N entrées  $x_i$** 
    - Associées à N coefficients multiplicatifs
  - **Une fonction de transfert non linéaire**
    - Sortie:  $y_j = f(\sum_k^N k_i * x_i + b)$
  - **M sorties  $y_j$**
- Le réseau connecte **toutes** les sorties des neurones, aux entrées des **autres** neurones
- Un réseau est structuré en **couches** contenant plusieurs neurones
  - La première couche est la **couche d'entrée**
  - Plusieurs couches **profondes** (le choix dépend largement du problème considéré)
  - Une dernière couche est la **couche de sortie**



# Principe d'un réseau de neurones

- La couche d'entrée contient autant de neurones qu'il faut pour décrire les données d'entrée
  - Par exemple pour RGB, la couche d'entrée contient 3 neurones
- La couche de sortie contient autant de neurones qu'il faut pour produire la sélection
  - Par exemple, pour les pastilles, la sortie demande un nombre entier ([0 .. 8]) donc il suffit d'un seul neurone
- La dimension des couches profondes est uniquement déterminée par l'expérience (!!!) et la capacité en mémoire ou CPU
  - Malheureusement, il n'y a pas de règles (*algorithmiques*) qui nous permette de définir l'architecture des couches, on s'appuie sur des « *recettes de cuisines* » que d'autres experts ont expérimentées.
  - De nombreuses documentations montrent des exemples, que l'on peut faire évoluer selon notre intuition.

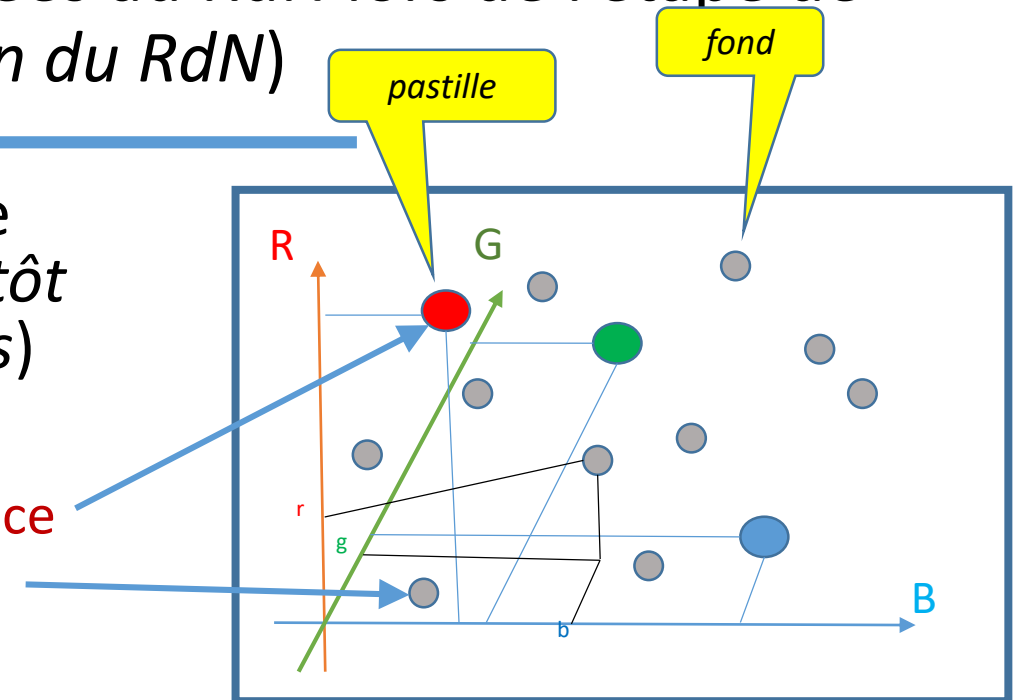
# Quelles sont les étapes de l'apprentissage?

1. On prépare les pastilles en définissant les  $N$  valeurs RGB (*on s'arrange que les  $N$  valeurs RGB soient choisies pour assurer un contraste entre les différentes pastilles mais aussi avec le fond*)
  - Le jeu de couleurs choisi, est caractéristique de l'apprentissage. Si on change ces valeurs, l'apprentissage sera à refaire !!!
2. On prépare des données d'apprentissage pour décrire
  - les valeurs RGB associées aux numéros des pastilles,
  - les autres valeurs RGB (*quelconques*) qui correspondent au fond (*quel qu'il soit*).
3. L'entraînement du RdN nécessite de produire deux **grands** jeux de données (*ayant exactement la même structure*) et reflétant la distribution statistique des données à reconnaître ensuite:
  - Un jeu pour les données d'apprentissage
  - Un jeu pour valider ou tester

# Préparation des données d'apprentissage

- **C'est une étape essentielle, et nécessaire à un apprentissage réussi**
- Le principe de cette préparation consiste à produire un très **grand** jeu de données, caractéristique de la *distribution statistique* effective des données qui seront réellement présentées au RdN lors de l'étape de reconnaissance (*c.à.d lors de l'utilisation du RdN*)

- 
- Pour notre problème, on va choisir comme données les valeurs **RGB** des couleurs (*plutôt que les données géométriques des pastilles*)
  - C'est un espace à 3 dimensions (R, G, B)
    - Chaque pastille occupe un point dans cet espace
    - Toutes les autres couleurs constituent le fond



# L'outillage informatique

- Ce sont des librairies **OpenSource**, d'accès libre, disponible sur toutes plateformes, utilisables dans une application **Python**
- **OpenCV**
  - Manipulation et traitement d'images
- **Numpy**
  - Gestion matricielle de base
- **Pandas**
  - Gestion de structures de données (au dessus de numpy)
- **Tensorflow**
  - Moteur du réseau de neurones
- **Keras**
  - Pilotage du réseau de neurones

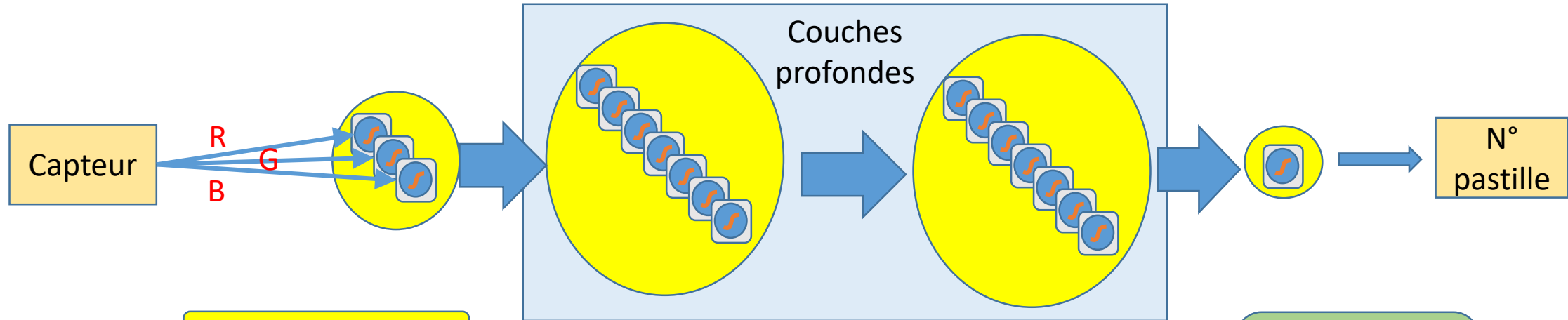
# Les pastilles

- On choisit N valeurs RGB telles qu'il y aura un contraste entre deux pastilles mais aussi avec un fond général (table par exemple)
- En pratique on considère 8 pastilles:
  - Chaque couleur fondamentale R, G, B, C, Y, M
  - Un gris foncé
  - Un gris clair

Rgb = 80 80 80	
Rgb = 0 0 255	(rouge)
Rgb = 0 255 0	(vert)
Rgb = 0 255 255	(cyan)
Rgb = 255 0 0	(jaune)
Rgb = 255 0 255	(magenta)
Rgb = 255 255 0	(cyan)
Rgb = 176 176 176	



# Définition d'un réseau de neurones



Architecture du réseau

```
model = keras.models.Sequential()
model.add(keras.layers.Input(3, name="InputLayer"))
model.add(keras.layers.Dense(64, activation="relu",
name="Dense_n1"))
model.add(keras.layers.Dense(64, activation="relu",
name="Dense_n2"))
model.add(keras.layers.Dense(1, name="Output"))

model.compile(optimizer = "rmsprop",
              loss      = "mse",
              metrics   = ["mae", "mse"])

return model
```

Couche d'entrée

2 Couches profondes  
Utilisant la fonction  
d'activation **relu**

Couche de sortie: il y a un  
seul neurone puisque le  
résultat est un **nombre**

Choix des algorithmes:

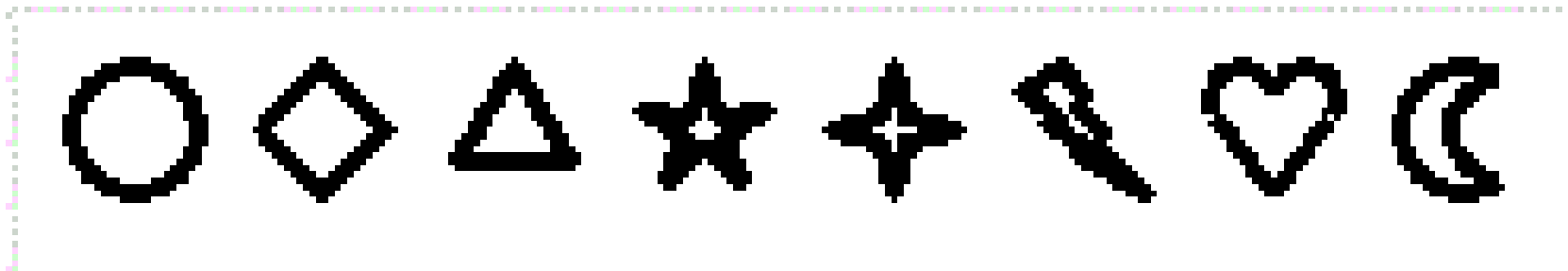
- descente de gradient
- métriques

La couche  
d'entrée est  
constituée de 3  
neurones (R, G, B)

Les 2 couches profondes  
sont constituées de 64  
neurones intégralement  
connectés entre eux

# Mécanisme de l'apprentissage

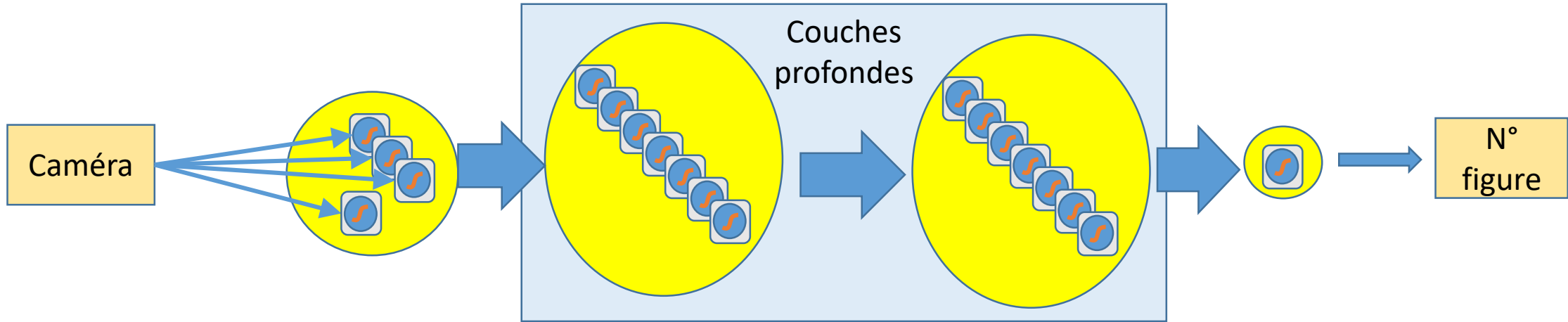
- Deuxième exemple: On veut entraîner un réseau de neurones (RdN) sur un ensemble de figures
  - Chaque figure sera dessinée (*imprimée*) sur un jeton (*papier*)
  - Un jeu, matérialisé par un véhicule-robot, est doté d'une caméra
  - Une fois entraîné, le RdN sera interrogé lorsque la caméra détecte un des jetons, pour obtenir le Numéro de la figure qui est imprimée.
  - Si la détection et la reconnaissance se produit à une fréquence adéquate, on peut espérer une reconnaissance en temps réel (*25 image par seconde*)



# Préparation des données

- Ici on considère que les jetons seront déposés aléatoirement sur la table où le véhicule-robot circuler. **En particulier les jetons ne seront pas orientés.**
- Ce qui implique que les données d'apprentissage doivent refléter des **rotations aléatoires**
- Puisque les données correspondent aux pixels des images fournies par une caméra, on va aussi produire des images un peu abîmées pour refléter les conditions de lumière, des reflets, des ombres, etc...

# Construction de l'architecture



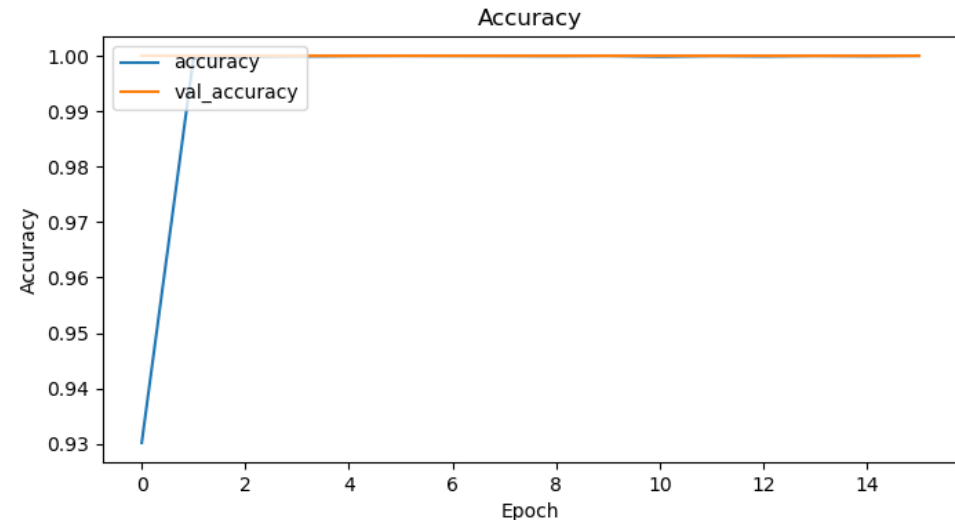
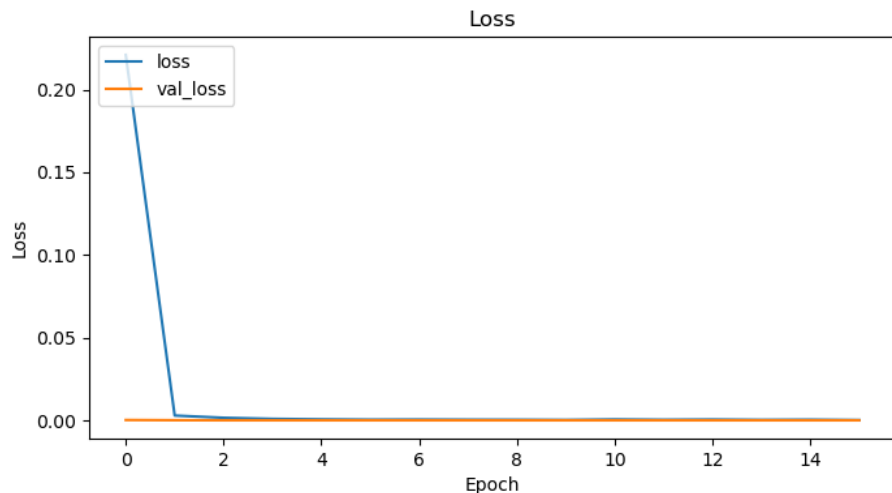
- La structure générale est identique à notre premier exemple: 1 couche d'entrée, 1 couche de sortie, et plusieurs couches profondes
  - Pour la couche d'entrée, il nous faudra donc un nombre de neurones correspondant au nombre de pixels des images =  $40 \times 40 \times 3$  (RGB)
  - Pour la couche de sortie, Ici, la sortie correspond au numéro de la figure à reconnaître c'est-à-dire un nombre entier ( $[0 .. 7]$ ) donc il suffit d'un seul neurone!
  - Pour les couches profondes, on insère plusieurs des couches utiles pour des opérations de traitement des images (*convolutions, optimisations, mise à plat, ...*)
- Ici, les nombre de neurones peut devenir très grand et donc le temps d'apprentissage peut devenir très long.

# Processus d'apprentissage

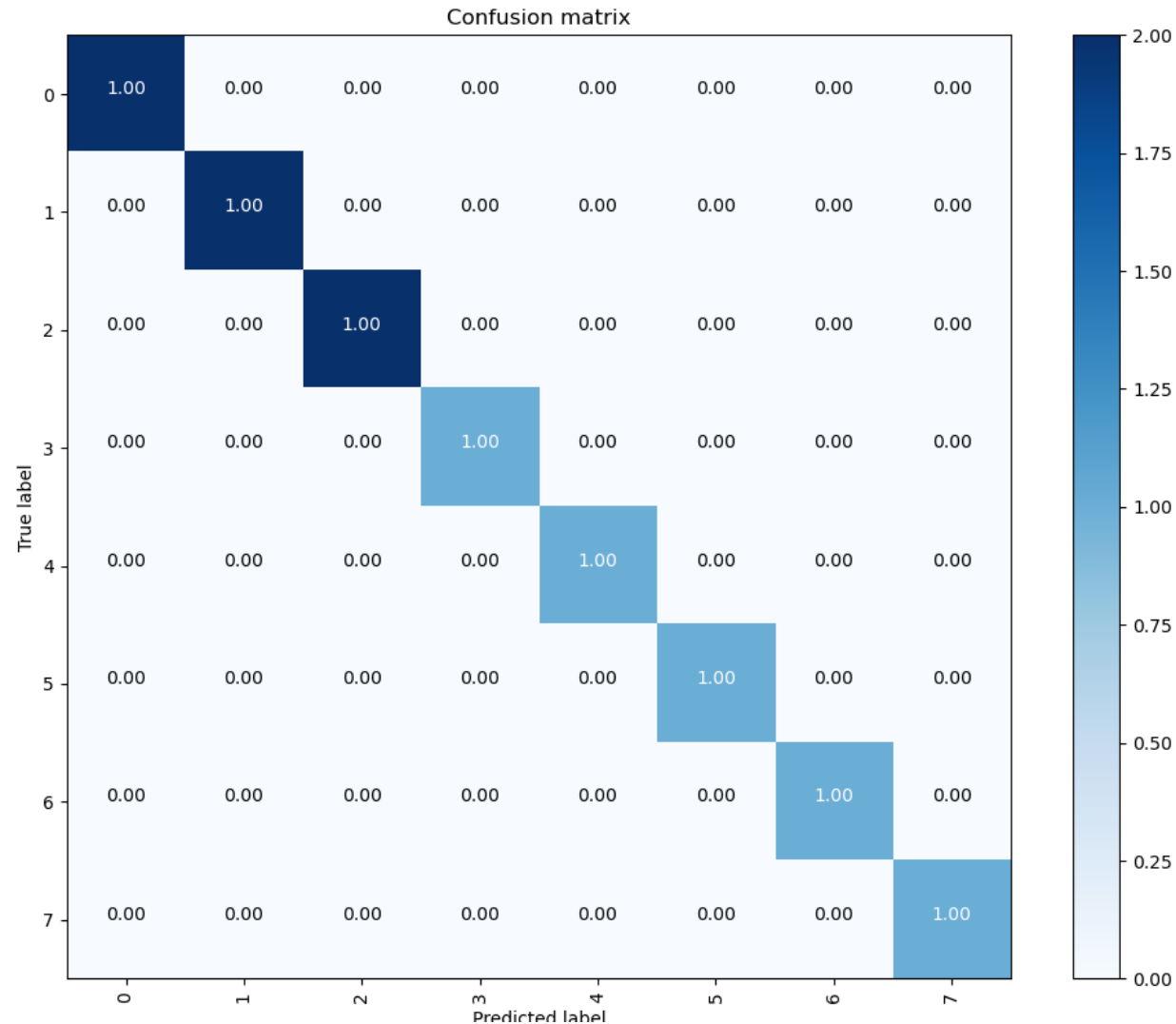
- On a préparé des données:
  - Les 8 figures
  - Puis on produit  $N \times 8$  versions des figures avec une rotation aléatoire, plus d'autres modifications des images (perspective, flou, etc...)
  - Pour cet exemple, j'ai produit environ 100 000 images modifiées avec la librairie OpenCV.
  - L'apprentissage a demandé plusieurs dizaines de minutes !!

# Évolution de l'apprentissage

- Plusieurs outils statistiques permettent de suivre l'évolution de l'apprentissage
  - Convergence du calcul des paramètres
  - Mesure de précision de la détermination de la reconnaissance
  - Estimation de la probabilité de confusion entre deux figures

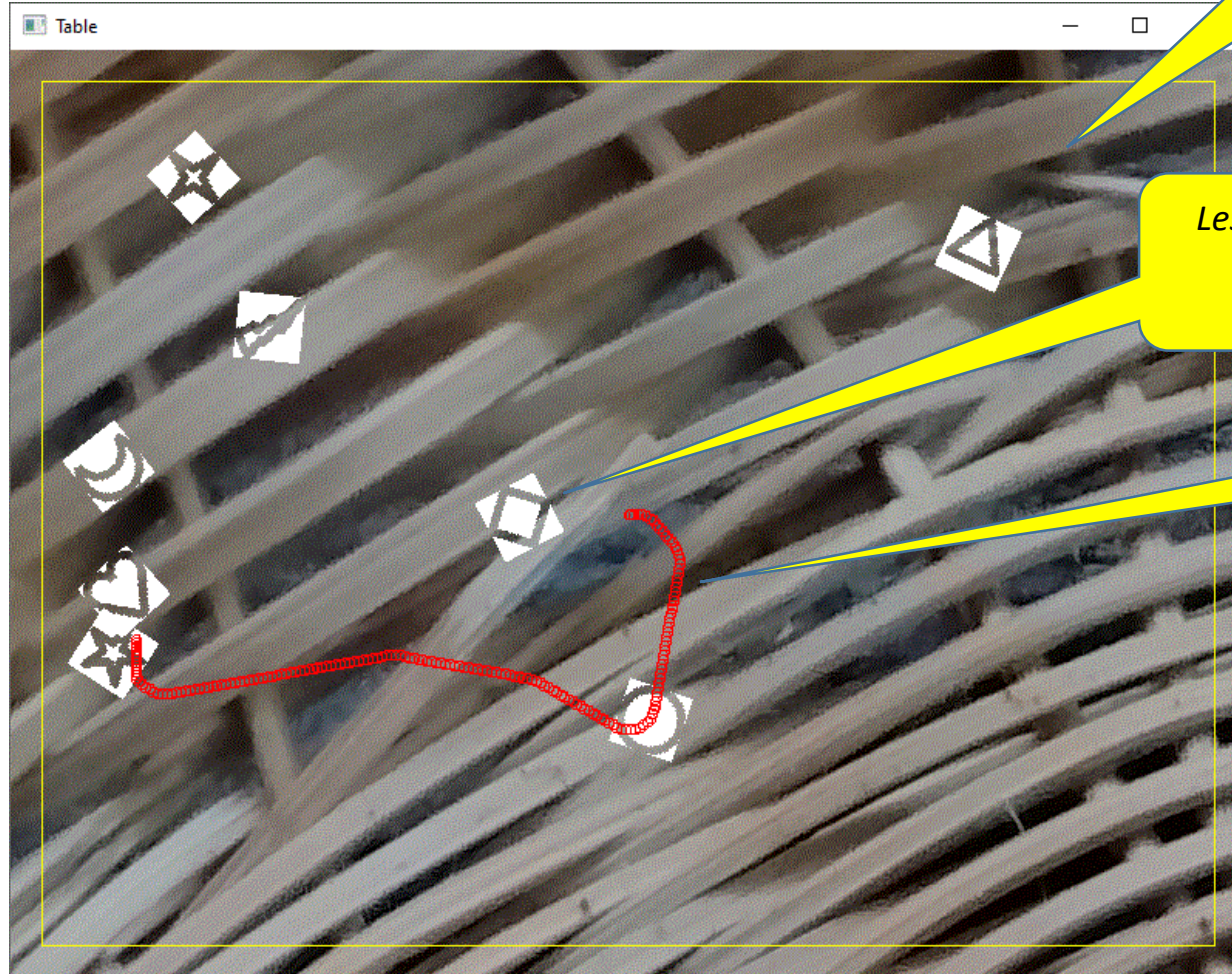


# Évaluation de l'apprentissage, la matrice de confusion





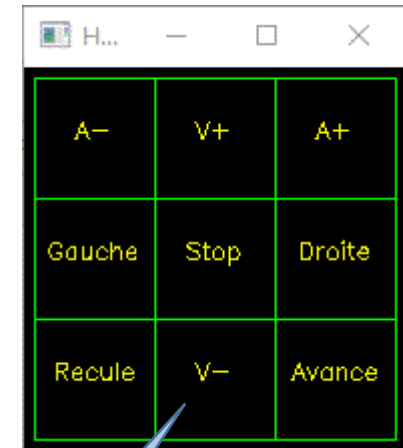
# Outil de simulation



Un fond quelconque

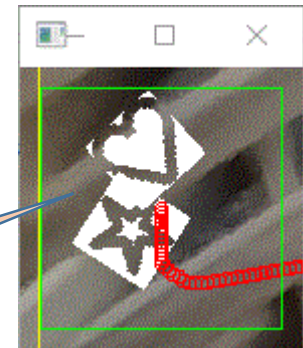
Les figures positionnées et orientées aléatoirement

La trace du robot



Panneau de contrôle (clavier numérique)

Simulation de la caméra





Merci pour votre attention !!

On peut voir ensuite quelques  
exemples de codes en option...

# Construction des données

```
data = pd.DataFrame(columns=['r', 'g', 'b', 'pastille'])
```

```
sigma = 1.5  
w = 6
```

```
data_size *= len(pastilles.keys())
```

*Il y aura une ligne de donnée par pastille*

```
for i in range(data_size):  
    for p in pastilles.keys():  
        rr, gg, bb = pastilles[p]  
        r = limits(int(gauss(mu=float(rr), sigma=sigma)))  
        g = limits(int(gauss(mu=float(gg), sigma=sigma)))  
        b = limits(int(gauss(mu=float(bb), sigma=sigma)))  
        insert(data, [float(r), float(g), float(b), float(p)])
```

*Simulation quand on est  
environ sur une pastille*

```
for k in pastilles.keys():  
    r = int(randrange(0, 256))  
    g = int(randrange(0, 256))  
    b = int(randrange(0, 256))  
    p = None  
    for pastille in pastilles.keys():  
        rr, gg, bb = pastilles[pastille]  
        if r > (rr - w) and r < (rr + w) and g > (gg - w) and g < (gg + w) and b > (bb - w) and b < (bb + w):  
            p = pastille  
            break  
    if p is None: p = 0  
    insert(data, [float(r), float(g), float(b), float(p)])
```

*Simulation générale*

*Mais on peut tomber sur  
une pastille par hasard*

```
return data
```

Les données: 1 matrice:

4 colonnes x N lignes

- 3 colonnes pour RGB
- 1 colonne pour le résultat

Deux séries de données vont être produites:

- Les RGB *proches* des pastilles
- Les RGB pour le fond

# Préparation des données

## Randomization des données

```
data = data.sample(frac=1., axis = 0)

data_train = data.sample(frac=0.8, axis=0)
data_test = data.drop(data_train.index)

x_train = data_train.drop(columns=['pastille'])
y_train = data_train['pastille']

x_test = data_test.drop(columns=['pastille'])
y_test = data_test['pastille']
```

## Séparation des données:

- 80% pour les données d'apprentissage
- 20% pour les données de test

## Séparation des données:

- Les colonnes RGB pour les données input
- La colonne **pastille** pour la sortie

```
mean = x_train.mean()
std = x_train.std()
```

## Normalisation des données

```
x_train = (x_train - mean) / std
x_test = (x_test - mean) / std
```

## Sauvegarde de **mean** / **std**

```
with open("./run/models/mean_std.txt", "w+") as f:
    f.write("{}\n".format(mean))
    f.write("{}\n".format(std))
```

La série X contient les données d'entrée  
La série Y contient les données de sortie

L'apprentissage consiste à

- appliquer une valeur RGB ( $X_i$ ) à l'entrée puis affirmer que la sortie vaut  $Y_i$
- Puis adapter les coefficients  $k_i$  pour que ce soit vrai (**descente de gradient**)

## x\_train

r	g	b	
0	-0.489185	-0.497316	-0.503679
1	-1.340131	-1.326919	1.332992
2	-1.340131	1.340413	-1.332804
3	-1.340131	1.340413	1.343487
4	1.328267	-1.337421	-1.322309
...	...	...	...
127995	-0.615251	-1.232407	0.724267
127996	-1.045977	-1.032883	-0.083869
127997	-0.163515	-0.360799	0.997143
127998	-0.447163	0.531813	-0.335755
127999	-0.184526	0.836351	0.965657

## y\_train

0	1.0
1	2.0
2	3.0
3	4.0
4	5.0
...	...
127995	0.0
127996	0.0
127997	0.0
127998	0.0
127999	0.0

# Entraînement du réseau

Création des données  
(voir plus haut)

Préparation des données  
(voir plus haut)

```
data = build_data(pastilles, data_number)
```

```
x_train, y_train, x_test, y_test, mean, std = prepare_data_for_training(data)
```

```
model = get_model_v1(3)
```

Construction du modèle

```
savemodel_callback = keras.callbacks.ModelCheckpoint(filepath=save_dir, verbose=0, save_best_only=True)
```

Déclaration d'actions à effectuer  
durant l'apprentissage  
Activation de la métrologie)

```
x_train, y_train = np.array(x_train), np.array(y_train)
```

```
x_test, y_test = np.array(x_test), np.array(y_test)
```

Conversion en array *numpy*

```
history = model.fit(x_train,  
                    y_train,
```

Etapes de l'apprentissage

Lance  
l'apprentissage

```
epochs = 100,  
batch_size = 10,
```

Données par paquets

```
verbose = 1,  
validation_data = (x_test, y_test),  
callbacks = [savemodel_callback])
```

```
score = model.evaluate(x_test, y_test, verbose=0)
```

Évaluation du processus  
d'apprentissage,  
impression des statistiques

L'apprentissage peut demander un temps assez long (en fonction de la taille et la complexité des données d'apprentissage, mais aussi du choix des couches profondes).

Donc, on effectue l'apprentissage par *paquets* de données, afin de suivre en temps réel la progression.

La métrologie permet de vérifier que les résultats s'améliorent au fur et à mesure.

# Utilisation du réseau entraîné

```
simulation_data = build_data(pastilles, N)

x_simulation = simulation_data.drop(columns=['pastille'])
y_simulation = simulation_data['pastille']

x_simulation = (x_simulation - mean) / std

predictions = loaded_model.predict(x_simulation, verbose=2)

real_data = x_simulation * std + mean

for n, i in enumerate(x_simulation.index):
    prediction = predictions[n][0]

    real = y_simulation.loc[i]
    delta = real - prediction
    error = prediction % 1.0

    pred = int(prediction)
    if pred in pastilles and abs(error) < 0.03:
        r, g, b = real_data.loc[i]
        rr, gg, bb = pastilles[pred]
        print(f"{i:03d} prediction={pred} error={error} color=[r{int(r)}, g{int(g)}, b{int(b)}] pastille=[r{int(rr)}, g{int(gg)}, b{int(bb)}]")

    elif pred == 0 and abs(error) < 0.03:
        r, g, b = real_data.loc[i]
        print(f"{i:03d} prediction={pred} error={error} color=[r{int(r)}, g{int(g)}, b{int(b)}]")
```

Préparation de N données simulées

Utilise le réseau entraîné pour faire des prédictions

Dénormalisation pour récupérer les vraies valeurs

Analyse des prédictions

On prédit que l'on est sur une pastille

On prédit que l'on est sur le fond

# Résultats

Prédiction: numéro de la pastille ou zéro pour le fond

Color effectivement lue

Color de la pastille

On a détecté une pastille

On est sur le fond

Estimation de l'erreur lors de reconnaissance

```
002 prediction=3 error=0.0031223297119140625 color=[r0, g255, b0] pastille=[r0, g255, b0]
004 prediction=5 error=0.020924091339111328 color=[r254, g0, b1] pastille=[r255, g0, b0]
008 prediction=0 error=0.003482341766357422 color=[r201, g33, b22]
009 prediction=0 error=0.001229703426361084 color=[r132, g17, b114]
010 prediction=0 error=0.0013559162616729736 color=[r127, g71, b57]
046 prediction=0 error=0.008603900671005249 color=[r59, g0, b70]
047 prediction=0 error=0.025277554988861084 color=[r202, g217, b26]
050 prediction=3 error=0.007489681243896484 color=[r0, g255, b1] pastille=[r0, g255, b0]
056 prediction=0 error=0.014719843864440918 color=[r238, g175, b30]
058 prediction=0 error=0.0102158784866333 color=[r104, g2, b177]
061 prediction=0 error=0.01276630163192749 color=[r228, g86, b5]
062 prediction=0 error=0.018812984228134155 color=[r7, g15, b84]
063 prediction=0 error=0.0006569921970367432 color=[r112, g202, b36]
072 prediction=0 error=0.001902170101274 color=[r141, g151, b63]
074 prediction=0 error=0.0013886183500289917 color=[r206, g133, b35]
076 prediction=0 error=0.001902170101274 color=[r73, g245, b77]
077 prediction=0 error=0.0013886183500289917 color=[r30, g166, b70]
078 prediction=0 error=0.008311629295349121 color=[r37, g210, b59]
093 prediction=0 error=0.001646280288696289 color=[r127, g71, b89]
095 prediction=0 error=0.014390379190444946 color=[r224, g26, b250]
098 prediction=3 error=0.0031223297119140625 color=[r0, g255, b0] pastille=[r0, g255, b0]
100 prediction=5 error=0.020924091339111328 color=[r254, g0, b1] pastille=[r255, g0, b0]
105 prediction=0 error=0.004931032657623291 color=[r216, g10, b124]
107 prediction=0 error=0.00182381272315979 color=[r212, g24, b42]
111 prediction=0 error=0.014600306749343872 color=[r145, g214, b105]
120 prediction=0 error=0.0019124150276184082 color=[r242, g34, b181]
```