

# Utilisation d'un réseau de neurones pour détecter les pastilles de couleurs

Chris Arnault

# Mécanisme de l'apprentissage

- On veut entraîner un réseau de neurones (RdN) sur un ensemble de pastille de couleurs
  - Chaque pastille est caractérisée par une valeur de RGB telle qu'un contraste existe entre toutes les pastilles
  - Une jeu est doté d'un capteur de couleur RGB
  - Le RdN sera interrogé pour demander à quelle pastille correspond une valeur RGB détectée

# Les étapes de l'apprentissage

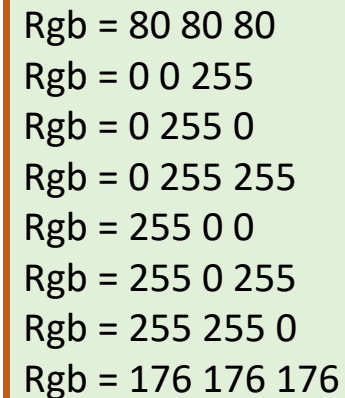
- On prépare les pastilles en définissant les N valeurs RGB (on s'arrange que les N valeurs RGB forment un contraste entre les différentes pastilles et aussi avec le fond)
- On prépare des données d'apprentissage consistant à la valeur RGB associée à la réponse attendue c.à.d soit le numéro de la pastille lorsqu'elle correspond, soit la valeur zéro lorsque la valeur RGB correspond à aucune pastille
- L'entraînement du RdN nécessite de diviser les données produites en deux sous-ensembles:
  - Les données d'apprentissage
  - Les données de validation

# L'outillage informatique

- 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 clair
  - Un gris foncé



Rgb = 80 80 80  
Rgb = 0 0 255  
Rgb = 0 255 0  
Rgb = 0 255 255  
Rgb = 255 0 0  
Rgb = 255 0 255  
Rgb = 255 255 0  
Rgb = 176 176 176

# Définition d'un réseau de neurones

Architecture du réseau

```
model = keras.models.Sequential()
model.add(keras.layers.Input(shape, 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

# Construction des données

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

Structure des données (3 colonnes)

```
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(0)])
```

Simulation générale

On prédit que l'on est sur une pastille

Mais on peut tomber sur une pastille par hasard

```
return data
```

# 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']

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

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

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

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

Normalisation des données

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

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

```
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)

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

x_train, y_train = np.array(x_train), np.array(y_train)
x_test, y_test = np.array(x_test), np.array(y_test)

history = model.fit(x_train,
                    y_train,
                    epochs = 100,
                    batch_size = 10,
                    verbose = 1,
                    validation_data = (x_test, y_test),
                    callbacks = [savemodel_callback])

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

Création de données

Préparation des données

Construction du modèle

Déclaration d'actions à effectuer  
durant l'apprentissage

Conversion en array **numpy**

Etapes de l'apprentissage

Données par étape

Lance  
l'apprentissage

Évaluation du processus  
d'apprentissage,  
impression des statistiques

# 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=0.001302170161214 color=[r141, g151, b63]
074 prediction=0 error=0.001302170161214 color=[r206, g133, b35]
076 prediction=0 error=0.001302170161214 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]
```