PROJET MACHINE LEARNING

LACHAUSSEE QUENTIN

14-01-2022

mesea
MAINTENANCE SEA TOURS · BORDEAUX

SOMMAIRE

I - ALGORITHME PERCEPTRON SIMPLE

Pour développer le perceptron simple, j'ai utilisé cette formule :

$$\sum_{j=0}^{d} w_j x_j \text{ avec } w_0 = \theta \text{ et } x_0 = 1$$

Threshold

```
y = w[1]*x[0] + w[2]*x[1] + w[0]*1

if active == 0:
    y = np.sign(y)
else:
    y = np.tanh(y)
```

Dans le cas du « OU », on utilise le « sign »

```
x = [0,0]
w = [-0.5,0,1]
y = perceptron_simple(x.copy(), w, active=0)
print(y)
```

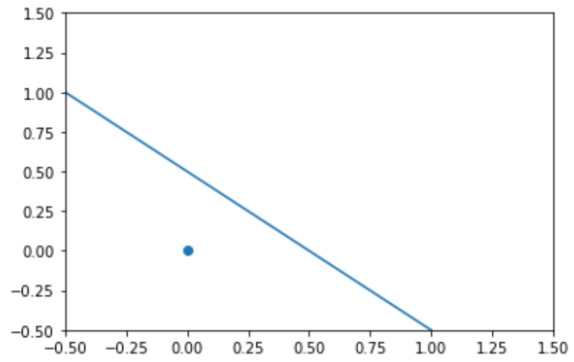
```
-1.0
```

On obtient bien « -1 »

La figure individu / frontière suivante montre que dans le cas du « OU »,

```
a = range(-10,10)
b = a*y - w[0]
plt.plot(a, b)
plt.scatter(x[0], x[1])
plt.axis([-0.5, 1.5, -0.5, 1.5])
```

```
(-0.5, 1.5, -0.5, 1.5)
```



les points en dessous de la ligne auront un résultat égal à « -1 », et « 1 » pour les points au dessus

II - APPRENTISSAGE WIDROW

Pour développer l'apprentissage Widrow, j'ai utilisé cette formule :

$$w_j^{n+1} = w_j^n - \alpha\left[-(d - \varphi(w^t x)).\varphi'(w^t x).x(j)\right]$$

```
loss = list()
w = [random.random(), random.random(), random.random()]
alpha = 0.9
for e in tqdm(range(epoch)):
    for i in range(len(x[0])):
        y = perceptron_simple(x[:,i], w, prime=0)

        erreur = y - yd[i]
        loss.append(erreur**2)

        for j in range(len(w)):
            S_erreur = erreur * perceptron_simple(x[:,i], w, prime=1)
            w[j] = w[j] - alpha * S_erreur * x[j][i]

    if erreur==0:
        break

return w, loss
```

Et de celle-ci pour l'erreur :

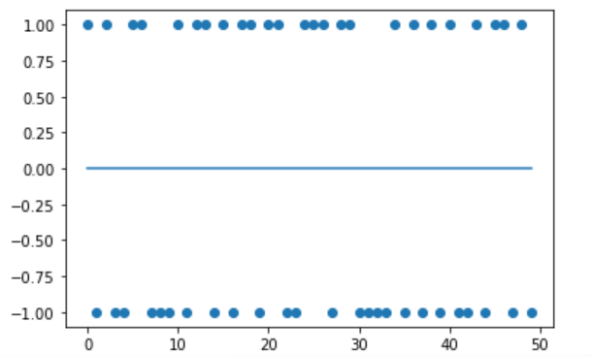
$$E = \frac{1}{2}\sum_{t=1}^{n}(d^{(t)} - \varphi(w^T s^{(t)})^2$$

Et pour tester l'apprentissage, le processus est le suivant :

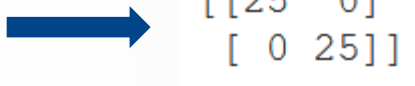
```
# on charge les données
Data = np.loadtxt("Data/p2_d1.txt")
# on concatène les données avec un vecteur de 50 1 et avec un vecteur de 25 -1 et 25 1 pour les classes
Data_full = np.concatenate((np.array([[1]*50]),Data,np.array([[-1]*25+[1]*25])))
# on mélange le dataset
Data_mix = Data_full[:, np.random.permutation(Data_full.shape[1])]
# on extrait l'entré
x = Data_mix[:-1]
# on extrait les classes à prédire
yd = Data_mix[-1]
```

II - APPRENTISSAGE WIDROW – TEST 1

Le graphique des prédictions

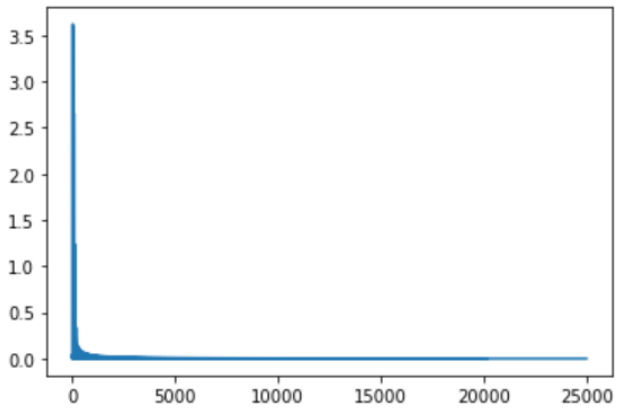


La matrice de confusion



```
[[25  0]
 [ 0 25]]
```

La courbe d'erreur



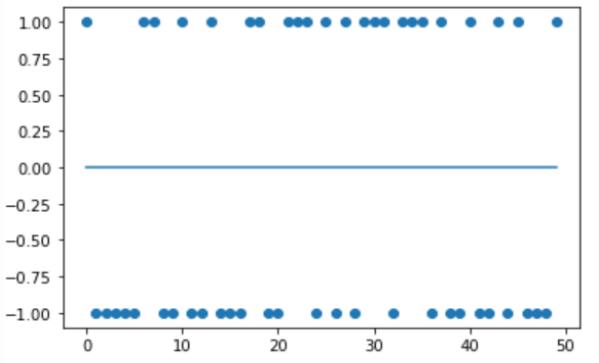
- Les prédictions sont excellentes
- La matrice de confusion affiche un résultat parfait
- On voit bien que la courbe d'erreur est de plus en plus faible et converge vers 0

Après quelques tests avec des initialisations différentes :

- Le nombre d'epoch devient de moins en moins significatif à partir d'environ 2500
- La valeur du batch_size influe sur la longueur de la courbe, mais diminue l'erreur plus rapidement

II - APPRENTISSAGE WIDROW – TEST 2

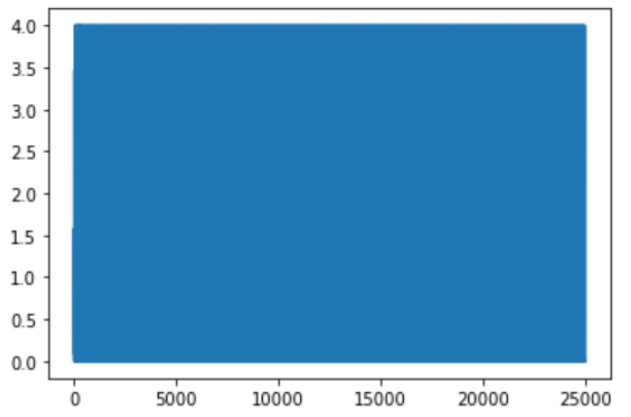
Le graphique des prédictions



La matrice de confusion

```
[[25  0]
 [ 2 23]]
```

La courbe d’erreur



- Les prédictions sont très bonnes
- La matrice de confusion affiche un résultat quasiment parfait
- En revanche la courbe d’erreur n’affiche pas de convergence vers 0

Après quelques tests avec des initialisations différentes, les résultats ne sont pas meilleurs

-> Les données n'ont sûrement pas assez de liens avec leur classe

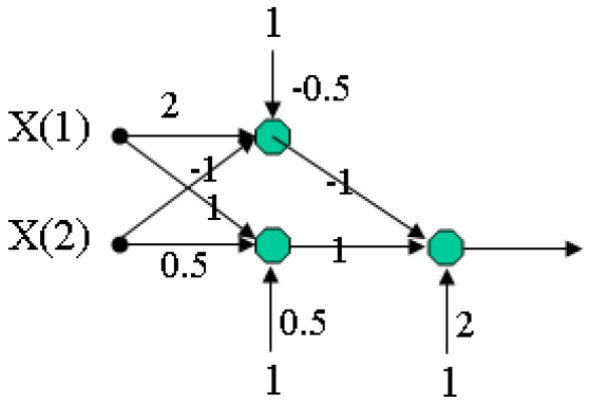
III - MISE EN PLACE D'UN PERCEPTRON MULTICOUCHE

Pour développer le perceptron multicouche, j'ai utilisé cette formule :

$$\varphi(x) = \frac{1}{1+e^{-x}}$$

```python
def fonction_activate(u: int):
    y = 1.0/(1 + np.exp(-u))

    return y
```

```python
x = np.append([1], x).reshape(3,1)

u1 = np.dot(w1[:,0].reshape(1,3), x)
y1 = fonction_activate(u1[0,0])

u2 = np.dot(w1[:,1].reshape(1,3), x)
y2 = fonction_activate(u2[0,0])

xf = np.append([1], [y1, y2]).reshape(3,1)

uf = np.dot(w2.reshape(1,3), xf)
yf = fonction_activate(uf[0,0])
```




```python
w1 = np.array([[-0.5,0.5],
               [2,1],
               [-1,0.5]])

w2 = np.array([[2],
               [-1],
               [1]])

x = [1,1]

y = multiperceptron(x, w1, w2)
print(y)
```

```
0.9053673095402572
```

Couche cachée :

- 1er neurone : -0.5 x 1 + 2 x 1 + -1 x 1 = 0.5
  -> donc 1/ (1 + e^(-0.5)) = 0.62
- 2e neurone : 0.5 x 1 + 1 x 1 + 0.5 x 1 = 2
  -> donc 1/ (1 + e^(-2)) = 0.88

Couche de sortie :

- 1er neurone : 2 x 1 + -1 x 0.62 + 1 x 0.88 = 2.26
  -> donc 1/ (1 + e^(-2.26)) = 0.91

mesea
MAINTENANCE SEA TOURS · BORDEAUX

IV - APPRENTISSAGE MULTICOUCHES

Pour développer l'apprentissage multicouche, j'ai utilisé cette formule :

$$\Delta w_k^{(\mu)}(j) = -\alpha . Err^{(\mu)}(k) . y^{(\mu-1)}(k) = -\alpha . \left[\varphi'(v^{(\mu)}(k)) \sum_{j \in \text{ couche } \mu+1} w_j^{(\mu+1)}(k) Err^{(\mu+1)}(j) \right] . y^{(\mu-1)}(j)$$

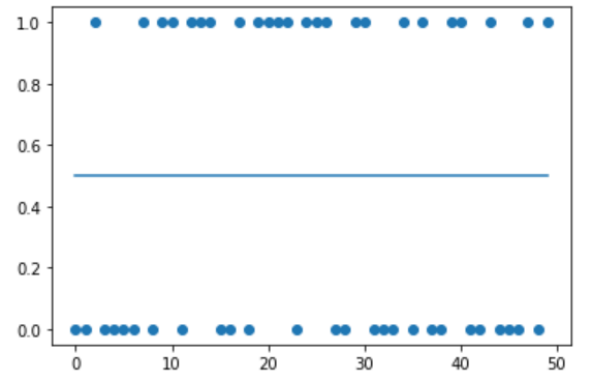
```
alpha = 0.5

# nombre de poids
nb_ligne_w = len(x[0])
# nombre de neuronne
nb_colonne_w = nb_neurone

w1 = np.random.uniform(-1, 1, (nb_ligne_w, nb_colonne_w))
w2 = np.random.uniform(-1, 1, nb_colonne_w + 1)

nb_ligne_x = len(x[:,0])
nb_colonne_x = len(x[0])

loss = list()
```

```
for e in tqdm(range(epoch)):
    for i in range(nb_ligne_x):
        u_w1 = np.zeros(nb_colonne_w)
        y_w1 = np.zeros(nb_colonne_w)
        y_prime_w1 = np.zeros(nb_colonne_w)
        for j in range(nb_colonne_w):
            u_w1[j] = np.dot(x[i,:], w1[:, j])
            y_w1[j] = fonction_activate(u_w1[j])
            y_prime_w1[j] = fonction_activate_deriv(u_w1[j])

        y_w1 = np.append([1], y_w1)
        y_prime_w1 = np.append([1], y_prime_w1)

        u_w2 = np.dot(w2, y_w1.reshape(nb_colonne_w + 1,1))
        y_w2 = fonction_activate(u_w2)

        error = y_w2 - yd[i]
        loss.append(error**2)

        for j in range(nb_colonne_w):
            S_error = error * fonction_activate_deriv(u_w2)

            for k in range(nb_colonne_x):
                gradient_w1 = S_error * w2[j] * y_prime_w1[j] * x[i, k]

                w1[k, j] -= alpha * gradient_w1

        for j in range(nb_colonne_w+1):

            gradient_w2 = S_error * y_w1[j]

            w2[j] -= alpha * gradient_w2

    if error==0:
        break

return w1, w2, loss
```

```
x = x.reshape(len(x),1)

y = list()
for i in range(len(w1[0])):
    u = np.dot(w1[:,i].reshape(1,len(w1)), x)[0,0]
    y.append(fonction_activate(u))


xf = np.append([1], y).reshape(len(y)+1,1)

uf = np.dot(w2.reshape(1,len(w2)), xf)[0,0]
yf = fonction_activate(uf)

return yf
```

IV - APPRENTISSAGE MULTICOUCHES

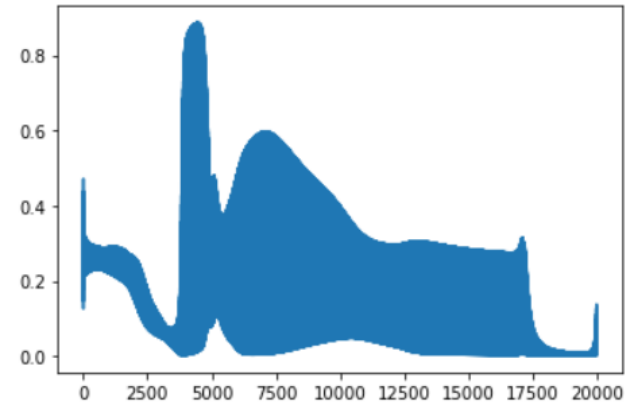
Le graphique des prédictions



La matrice de confusion

```
[[24  1]
 [ 2 23]]
```

La courbe d'erreur



- Les prédictions sont très bonnes
- La matrice de confusion affiche un résultat quasiment parfait
- En revanche la courbe d'erreur n'affiche pas une convergence nette vers 0

Après quelques tests avec des initialisations différentes, les résultats peuvent s'améliorer et présenter une belle convergence
et une fois relancé, peut grandement se détériorer

J'ai peut-être fait une erreur dans la fonction d'apprentissage mais je n'ai pas réussi à trouver où (malgré plusieurs heures de recherche)
Je n'ai donc pas fais la suite pour cette partie IV

V - VOLET CHARGEMENT DES DESCRIPTEURS

Pour charger les données, j'avais premièrement utilisé un dictionnaire, mais j'ai finalement opté pour une boucle qui fera directement :

- Le chargement des données
- Le split des données
- La création du modèle
- La prédiction
- L'affichage des résultats

```
# pour chaque feuille du fichier Excel (chaque features)
for name in pd.ExcelFile("Data/WangSignatures.xlsx").sheet_names:

    # on affiche la feature
    print(name)

    # on charge les données
    Wang = load_data(name)

    # on charge les différents dataset et listes de classe
    X_train, X_test, y_train, y_test, y_train2, y_test2 = separate_data(Wang)

    # on crée et entraine le model
    model = create_model(X_train, y_train2)

    # on montre les resultats des prédictions du model sur le dataset de test
    prediction = show_resultats(model, X_test, y_test, y_test2)
```

V - VOLET CHARGEMENT DES DESCRIPTEURS - CHARGEMENT

Il fallait faire attention à l'ordre des données qui était faussé par les mention « .jpg » dans la 1ère colonne.
Ensuite on attribue les bonnes classes aux lignes de données

	A	B	C	D	
1	0.jpg	1.0	4.0	3.0	1.0
2	1.jpg	1.0	5.0	5.0	2.0
3	10.jpg	5.0	5.0	2.0	1.0
4	100.jpg	1.0	4.0	2.0	2.0
5	101.jpg	6.0	6.0	1.0	0.0
6	102.jpg	2.0	6.0	1.0	0.0
7	103.jpg	2.0	6.0	1.0	0.0
8	104.jpg	3.0	7.0	0.0	0.0
9	105.jpg	0.0	1.0	5.0	1.0
10	106.jpg	5.0	5.0	1.0	0.0
11	107.jpg	1.0	4.0	2.0	0.0
12	108.jpg	1.0	5.0	2.0	1.0
13	109.jpg	1.0	5.0	5.0	0.0
14	11.jpg	3.0	3.0	3.0	1.0
15	110.jpg	0.0	3.0	3.0	0.0

```python
def load_data(type: str):
    """
    Load Data
    qui à partir du fichiers Excel de données
    va créer un Dataframe avec les données du type spécifié et leur classe

    :param type: le nom de la feuille du fichier Excel WangSignatures
    :type type: str
    :return Wang: le DataFrame avec les données du type spécifié et leur classe
    :rtype Wang: pd.DataFrame
    """

    # on lit la feuille du fichier Excel des données WangSignatures
    Wang = pd.read_excel("Data/WangSignatures.xlsx", sheet_name=type, header=None)

    # on retire la mention jpg qui apparait à chaque ligne de la colonne des nom d'image pour pouvoir trier les données par ordre numéric
    Wang[0]=Wang[0].apply(lambda x: float(x.replace(".jpg","")))
    Wang = Wang.sort_values(by=0).reset_index(drop=True)

    # on leur affecte une classe selon leur ordre numérique
    Wang["y"] = [0]*100+[1]*100+[2]*100+[3]*100+[4]*100+[5]*100+[6]*100+[7]*100+[8]*100+[9]*100
    # on supprime la colonne avec le nom de l'image
    Wang = Wang.drop(0,axis=1)

    # on retourne le DataFrame
    return Wang
```

V - VOLET CHARGEMENT DES DESCRIPTEURS - SPLIT

Il fallait faire attention à la « shape » des données de classe (y)
Par exemple on modifie le y_train qui est égal à « 3 » par une liste de 9 « 0 » et 1 « 1 » en position 3.
Pareil pour y_test.

```python
def separate_data(df: pd.DataFrame):
    """
    Separate Data
    qui à partir du DataFrame de données
    va spliter les données en dataset d'entrainement et de test

    :param df: le DataFrame de données
    :type df: pd.DataFrame
    :return X_train: le dataset d'entrainement
    :rtype X_train: np.array
    :return X_test: le dataset de test
    :rtype X_test: np.array
    :return y_train: les classes du dataset d'entrainement au format (n,1)
    :rtype y_train: np.array
    :return y_test: les classes du dataset de test au format (n,1)
    :rtype y_test: np.array
    :return y_train2: les classes du dataset d'entrainement au format (n,10) où la colonne correspond à la classe
    :rtype y_train2: np.array
    :return y_test2: les classes du dataset d'entrainement au format (n,10) où la colonne correspond à la classe
    :rtype y_test2: np.array
    """

    # on conserve tous sauf la dernière colonne pour l'ensemble des données
    X = np.array(df.astype('float32').iloc[:,:-1])
    # on ne conserve que la derniere colonne pour l'ensemble des classe
    y = np.array(df.astype('float32').iloc[:,-1])

    # on utilise sklearn pour spliter les données en dataset d'entrainement et de test selon une proportion de 0.8 pour 0.2
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)

    # on modifie la structure des classes pour une matrice (n,10) où 10 est le nombre de classe pour que chaque colonne corre
    y_train2 = one_hot(y_train, depth=10)
    y_test2 = one_hot(y_test, depth=10)

    # on retourne les datasets et les classes selon leur 2 formats
    return X_train, X_test, y_train, y_test, y_train2, y_test2
```

3

[0,0,0,1,0,0,0,0,0,0]

VI - CLASSIFICATION PAR FULL CONNECTED – 1ERE CRÉATION

Pour une première approche autour de la création du modèle, j'ai fais le choix de 2 couches Dense :

- 1 de type « relu » avec 30 neurones
- 1 de type « softmax » avec 10 neurones (pour les 10 classes à prédire)

Pour la compilation, j'ai utilisé :

- La fonction de perte « categorical_crossentropy » qui est souvent considérée comme la meilleure pour les classifications à plusieurs classes
- L'optimiseur « Adam » pour s'appuyer sur le gradient stochastique

```python
def create_model(X_train: np.ndarray, y_train2: np.ndarray):
    """
    Create model
    qui à partir du dataset d'entrainement et des classes correspondantes
    va créer un model et l'entrainer selon plusieurs couches de convolution

    :param X_train: le dataset d'entrainement
    :type X_train: np.array
    :param y_train2: les classes du dataset d'entrainement au format (n,10) où la colonne correspond à la classe
    :type y_train2: np.array
    :return model: le model de prédiction
    :rtype model: keras.engine.sequential.Sequential
    """

    # on crée un model avec plusieurs couches
    model = Sequential()

    model.add(Dense(30, activation="relu"))
    model.add(Dense(10, activation="softmax"))

    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

    model.fit(X_train, y_train2, epochs=50, batch_size=10, validation_split=0.1, verbose=0)

    # on retourne le modèle
    return model
```

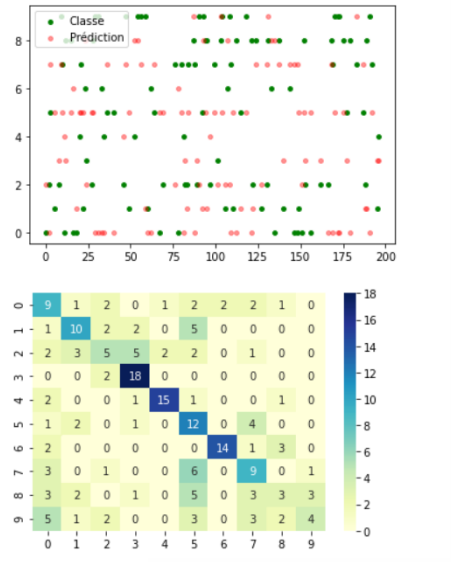
VI - CLASSIFICATION PAR FULL CONNECTED – 1ERS RÉSULTATS

Pour les résultats, j'ai opté pour affiché sous forme de Score : [*l'erreur*, *l'accuracy*]
Puis un nuage de point qui représente :

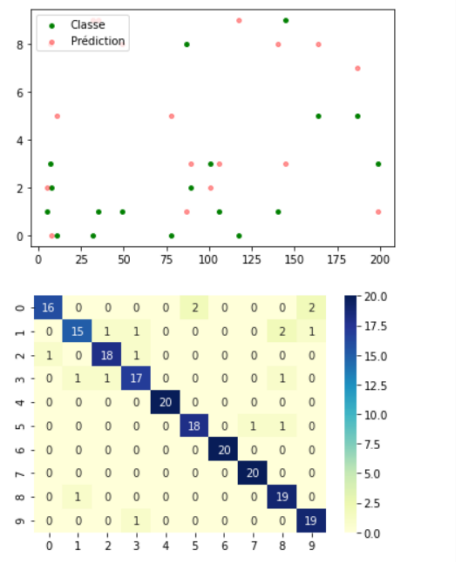
- En rouge, les classes mal prédites
- En vert, leur vrai classe

Et enfin une matrice de confusion en utilisant la palette de couleur « YlGnBu »

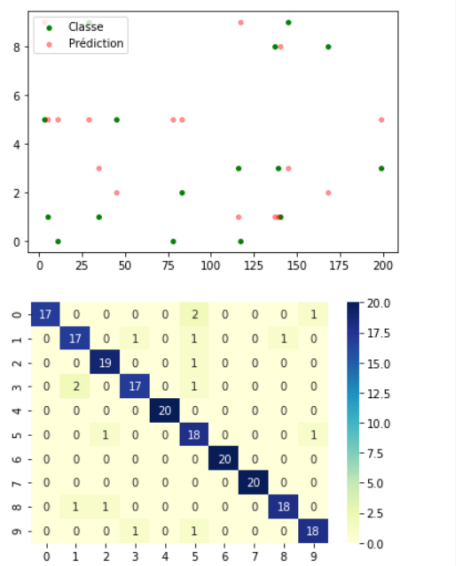
WangSignaturesPHOG
Score : [1.7613438367843628, 0.4950000047683716]



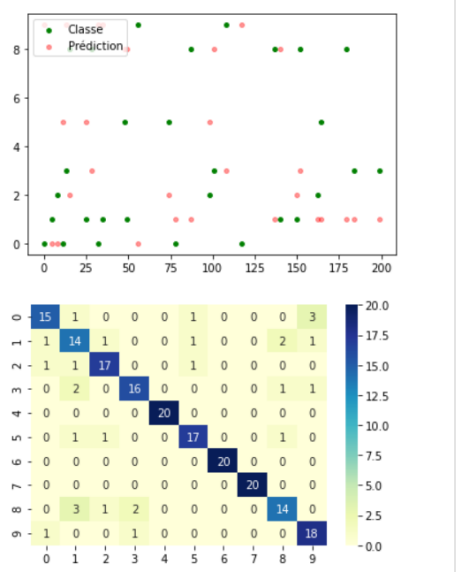
WangSignaturesJCD
Score : [0.2863759696483612, 0.9100000262260437]



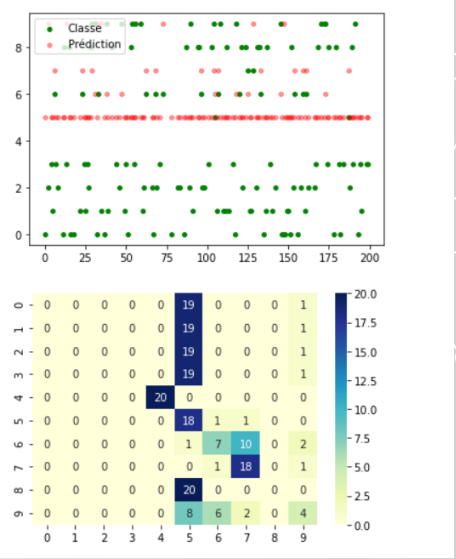
WangSignaturesCEDD
Score : [0.3429938554763794, 0.9200000166893005]



WangSignaturesFCTH
Score : [0.3990945816040039, 0.8550000190734863]



WangSignaturesFuzzyColorHistogr
Score : [1.6111088991165161, 0.33500000834465027]



Les résultats sont plutôt bons sur 3 ensembles de données sur 5 -> **peut mieux faire**

mesea
MAINTENANCE SEA TOURS · BORDEAUX

VI - CLASSIFICATION PAR FULL CONNECTED – CRÉATION

Après une dizaine de test en modifiant :

- Le nombre de couche
- Le nombre de neurone
- Les fonctions d'activation

J'en ai ressorti comme conclusion que :

- Plus il y avait de couches, mieux c'était
- Le nombre de neurone pouvait faire passer les résultats de très bon à très mauvais
- La fonction d'activation « relu » était très puissante
- La dernière fonction d'activation devait être « softmax » pour obtenir les meilleurs résultats

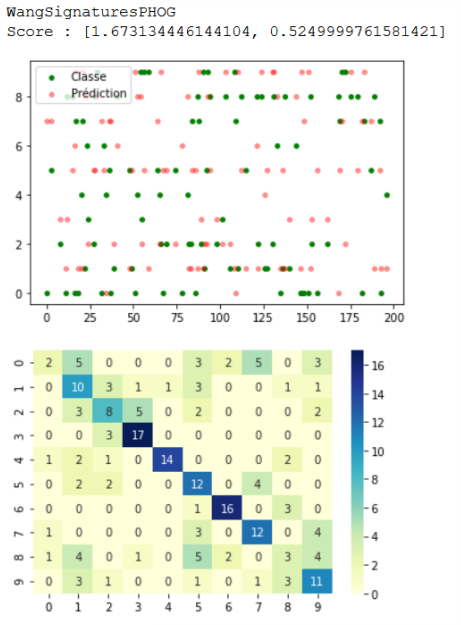
J'ai donc fais le choix de 5 couches :

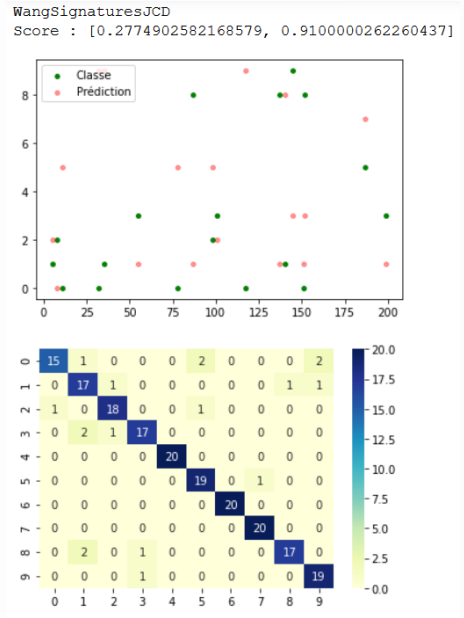
- 4 de type « relu » :
  - 250 / 120 / 60 / 30 / 10 neurones
- 1 de type « softmax » :
  - 10 neurones

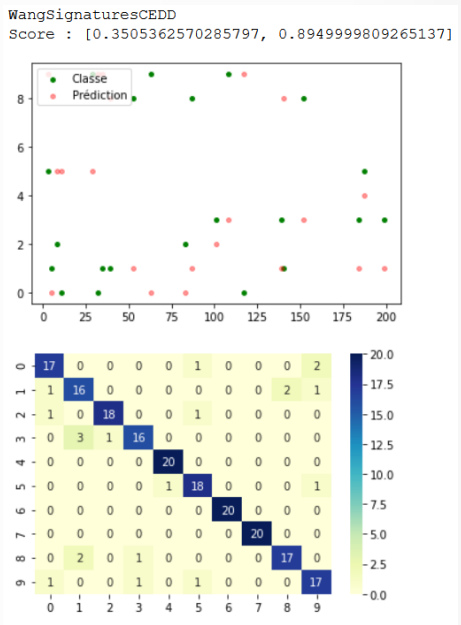
```
# on crée un model avec plusieurs couches
model = Sequential()

model.add(Dense(250, activation="relu"))
model.add(Dense(120, activation="relu"))
model.add(Dense(60, activation="relu"))
model.add(Dense(30, activation="relu"))
model.add(Dense(10, activation="softmax"))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(X_train, y_train2, epochs=50, batch_size=10, validation_split=0.1, verbose=0)

# on retourne le modèle
return model
```

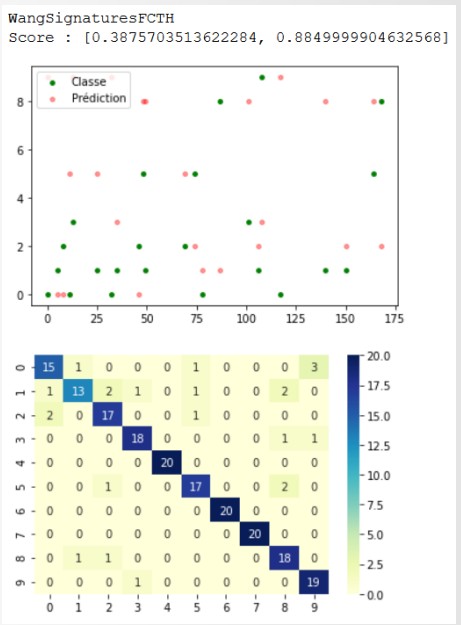
VI - CLASSIFICATION PAR FULL CONNECTED – RÉSULTATS

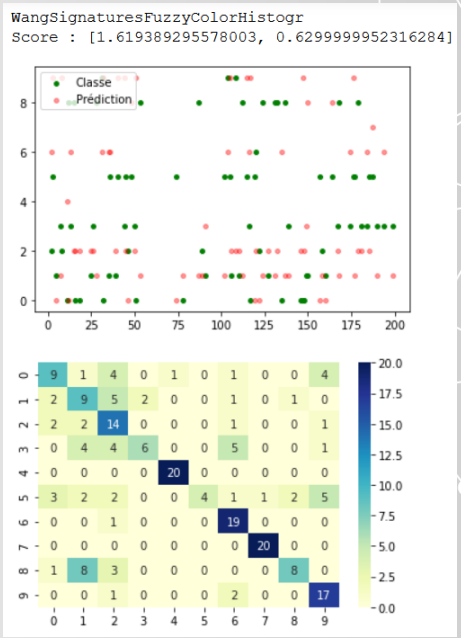
Les résultats sont très bons pour les ensembles de données JCD/ CEDD/ FCTH
Les résultats des ensembles de données PHOG/ Fuzzy se sont améliorés -> **peut encore mieux faire**











Mais sur l'ensemble des données c'est le meilleur résultat que j'ai pu obtenir sans détériorer les uns des autres

VI - CLASSIFICATION PAR FULL CONNECTED – REGROUPEMENT

Donc je me suis dit que si je n'arrivais pas à améliorer chacun des résultats sans détériorer les autres,
Il fallait que je regroupe toutes les données en 1 seul et unique ensemble de données :

```
Wang_final = pd.DataFrame()
# pour chaque feuille du fichier Excel (chaque features)
for name in tqdm(pd.ExcelFile("Data/WangSignatures.xlsx").sheet_names):

    # on charge les données
    Wang = load_data(name).drop(["y"], axis=1)

    Wang_final = pd.concat([Wang_final, Wang], axis=1)


# on leur affecte une classe selon leur ordre numérique
Wang_final["y"] = [0]*100+[1]*100+[2]*100+[3]*100+[4]*100+[5]*100+[6]*100+[7]*100+[8]*100+[9]*100
```

On a bien 1000 lignes
et le 885 colonnes pour :

- 255 pour PHOG
- 168 pour JCD
- 144 pour CEDD
- 192 pour FCTH
- 125 pour Fuzzy
- 1 pour la classe

```
Wang_final
```

	1	2	3	4	5	6	7	8	9	10	...	117	118	119	120	121	122	123	124	125	y
0	10.0	8.0	7.0	7.0	7.0	7.0	7.0	7.0	6.0	7.0	...	72.0	78.0	79.0	71.0	57.0	62.0	67.0	69.0	69.0	0
1	15.0	13.0	12.0	11.0	10.0	10.0	10.0	8.0	9.0	10.0	...	76.0	84.0	89.0	81.0	60.0	66.0	72.0	76.0	75.0	0
2	8.0	7.0	7.0	5.0	4.0	4.0	4.0	4.0	4.0	4.0	...	66.0	74.0	81.0	82.0	53.0	59.0	66.0	76.0	132.0	0
3	2.0	2.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	83.0	97.0	113.0	111.0	64.0	74.0	86.0	106.0	206.0	0
4	15.0	12.0	10.0	7.0	6.0	5.0	5.0	5.0	4.0	5.0	...	48.0	54.0	64.0	66.0	39.0	44.0	51.0	70.0	139.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	15.0	12.0	11.0	10.0	10.0	10.0	9.0	8.0	8.0	8.0	...	18.0	25.0	37.0	42.0	14.0	18.0	24.0	42.0	255.0	9
996	15.0	13.0	14.0	13.0	13.0	13.0	14.0	13.0	13.0	14.0	...	84.0	105.0	136.0	140.0	65.0	78.0	96.0	124.0	153.0	9
997	15.0	14.0	14.0	10.0	7.0	7.0	6.0	5.0	5.0	4.0	...	56.0	74.0	103.0	114.0	43.0	55.0	75.0	133.0	255.0	9
998	15.0	12.0	11.0	9.0	8.0	7.0	7.0	7.0	6.0	6.0	...	44.0	48.0	55.0	52.0	36.0	39.0	43.0	48.0	48.0	9
999	15.0	13.0	13.0	11.0	11.0	10.0	10.0	9.0	8.0	8.0	...	90.0	109.0	130.0	118.0	68.0	80.0	95.0	113.0	109.0	9

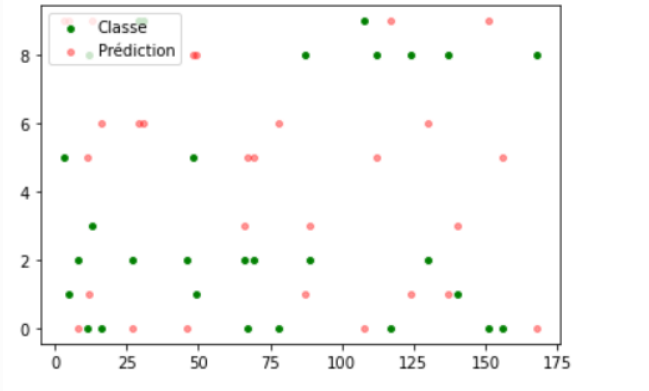
1000 rows × 885 columns

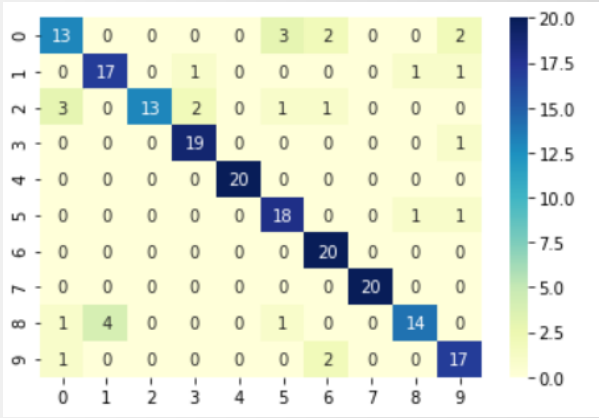
VI - CLASSIFICATION PAR FULL CONNECTED – RÉSULTATS

J'ai donc utilisé mes conclusions précédentes pour créer mon super model de cette manière :

```
# on crée un model avec plusieurs couches
model = Sequential()

model.add(Dense(800, activation="relu"))
model.add(Dense(750, activation="relu"))
model.add(Dense(700, activation="relu"))
model.add(Dense(600, activation="relu"))
model.add(Dense(300, activation="relu"))
model.add(Dense(200, activation="relu"))
model.add(Dense(80, activation="relu"))
model.add(Dense(30, activation="relu"))
model.add(Dense(10, activation="softmax"))
```

```
Score : [0.5389025211334229, 0.8550000190734863]
```





Et on arrive à un très bon résultats : 86 % en accuracy
Ce qui me satisfait parfaitement

VII - CLASSIFICATION PAR DEEP – CHARGEMENT

Pour charger les images en 2 ensembles de données Train et Validation, j'ai utilisé la fonction :

tf.keras.utils.image_dataset_from_directory

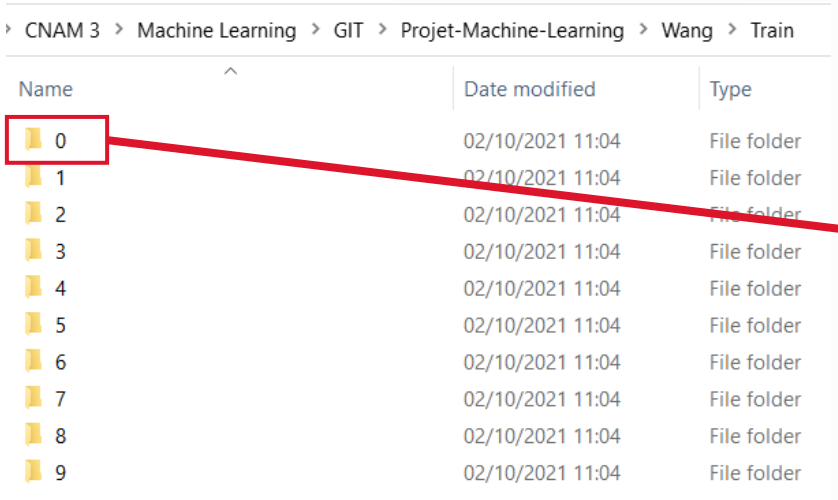
```
# on va créer un dataset d'entrainement depuis le dossier Train
train_data = image_dataset_from_directory(
    directory="Wang/Train",
    validation_split=0.2,
    subset="training",
    seed=42,
    batch_size=3)

# on va créer un dataset de validation depuis le dossier Test
val_data = image_dataset_from_directory(
    directory="Wang/Train",
    validation_split=0.2,
    subset="validation",
    seed=42,
    batch_size=3)
```

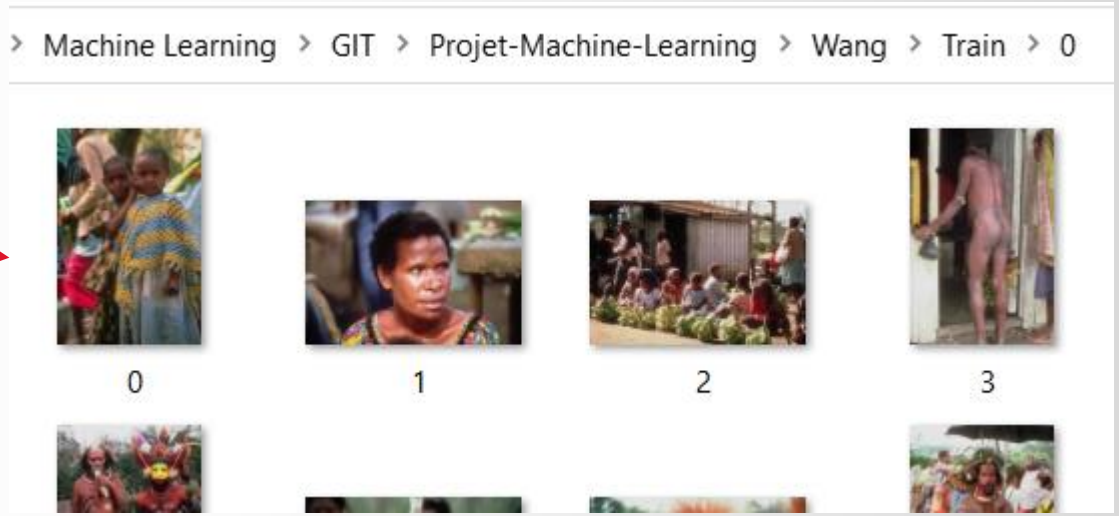
Qui va aller :

- Cibler un dossier
- Lire les images pixel par pixel
- Les transformer en liste ou chaque pixel est une liste de 3 valeurs (RGB)
- Leur attribuer la classe qu'il convient
- Splitter les données en ensemble Train et Validation

VII - CLASSIFICATION PAR DEEP – CHARGEMENT

Mais pour utiliser la fonction « **image_dataset_from_directory** », il fallait que le dossier soit dans un format très précis, chaque sous-dossier doit correspondre à la classe de l'image.





Donc j'ai créé la fonction « **creat_directory_and_class** » pour m'automatiser tout ça :

```
def create_directory_and_class():
    """
    Create Directory And Classlass
    qui à partir du dossier Wang
    va créer 2 sous dossier de Train et Test composé de 9 sous sous dossiers pour les 9 classes d'image

    """
```

VII - CLASSIFICATION PAR DEEP – SPLIT

On arrive donc à mettre en place nos jeux de donnée Train et Validation avec les 10 classes :

```
# on prépare les données
train_data, val_data, y = load_data()
```

```
Found 1000 files belonging to 10 classes.
Using 800 files for training.
Found 1000 files belonging to 10 classes.
Using 200 files for validation.
```

Où chaque image est
une liste de 256 listes de 3 valeurs
(256 pixels divisés en 3 valeurs RGB) :

```
[[ 15  16  44]
 [ 27  31  58]
 [ 34  40  64]
 [ 56  67  89]
 [ 56  68  90]
 [ 58  74  94]
 [ 65  83 103]
```

...

mesea
MAINTENANCE SEA TOURS · BORDEAUX

VII - CLASSIFICATION PAR DEEP – CRÉATION

Pour cette création du modèle je mes suis aidé d'une chaine YouTube « Defend Intelligence » qui m'a proposé différents modèles, et c'est le suivant qui m'a apporté les meilleurs résultats :

```
# on crée un model avec plusieurs couches de convolution d'image 2D
model = Sequential([
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(128,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16,4, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss=losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'],)

tensorboard_callback = keras.callbacks.TensorBoard(log_dir="logs",
                                                   histogram_freq=1,
                                                   write_images="logs",
                                                   embeddings_data=train_data)

model.fit(train_data,
        validation_data=val_data,
        epochs=10,
        callbacks=[tensorboard_callback]
)
```

En oubliant pas de mettre 10 neurones à la dernière couche (pour les 10 classes à prédire)

Et en rajoutant le dataset de Validation lors du « **fit** »

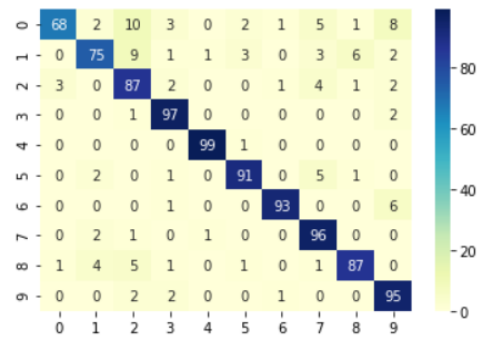
VII - CLASSIFICATION PAR DEEP – RÉSULTATS

Pour l'affichage, j'ai préféré laisser le « verbose » par défaut pour voir l'évolution de « l'accuracy ».
Et toujours ma matrice de confusion en utilisant la palette de couleur « YlGnBu »

```
267/267 [==============================] - 113s 417ms/step - loss: 1.9563 - accuracy: 0.2850 - val_loss: 1.4739 - val
_accuracy: 0.5300
Epoch 2/10
267/267 [==============================] - 117s 436ms/step - loss: 1.3114 - accuracy: 0.5450 - val_loss: 1.4063 - val
_accuracy: 0.5550
Epoch 3/10
267/267 [==============================] - 107s 399ms/step - loss: 0.9557 - accuracy: 0.6612 - val_loss: 1.1611 - val
_accuracy: 0.6200
Epoch 4/10
267/267 [==============================] - 123s 460ms/step - loss: 0.6661 - accuracy: 0.7688 - val_loss: 1.3611 - val
_accuracy: 0.6050
Epoch 5/10
267/267 [==============================] - 109s 406ms/step - loss: 0.4090 - accuracy: 0.8587 - val_loss: 1.6791 - val
_accuracy: 0.5900
Epoch 6/10
267/267 [==============================] - 122s 456ms/step - loss: 0.3155 - accuracy: 0.9000 - val_loss: 2.2486 - val
_accuracy: 0.5500
Epoch 7/10
267/267 [==============================] - 119s 446ms/step - loss: 0.2012 - accuracy: 0.9275 - val_loss: 2.1893 - val
_accuracy: 0.5750
Epoch 8/10
267/267 [==============================] - 123s 460ms/step - loss: 0.1116 - accuracy: 0.9575 - val_loss: 2.6942 - val
_accuracy: 0.6450
Epoch 9/10
267/267 [==============================] - 129s 482ms/step - loss: 0.1096 - accuracy: 0.9625 - val_loss: 3.7420 - val
_accuracy: 0.5500
Epoch 10/10
267/267 [==============================] - 130s 487ms/step - loss: 0.1654 - accuracy: 0.9538 - val_loss: 2.8544 - val
_accuracy: 0.6300
```

```
# on prédit le Train
y, class_predict = model_prediction_folder(model, "Train")
```

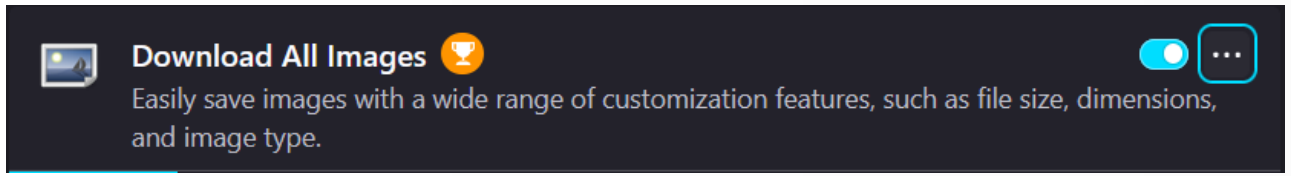
```
100%|
s]
Accuracy : 0.888
AxesSubplot(0.125,0.125;0.62x0.755)
```

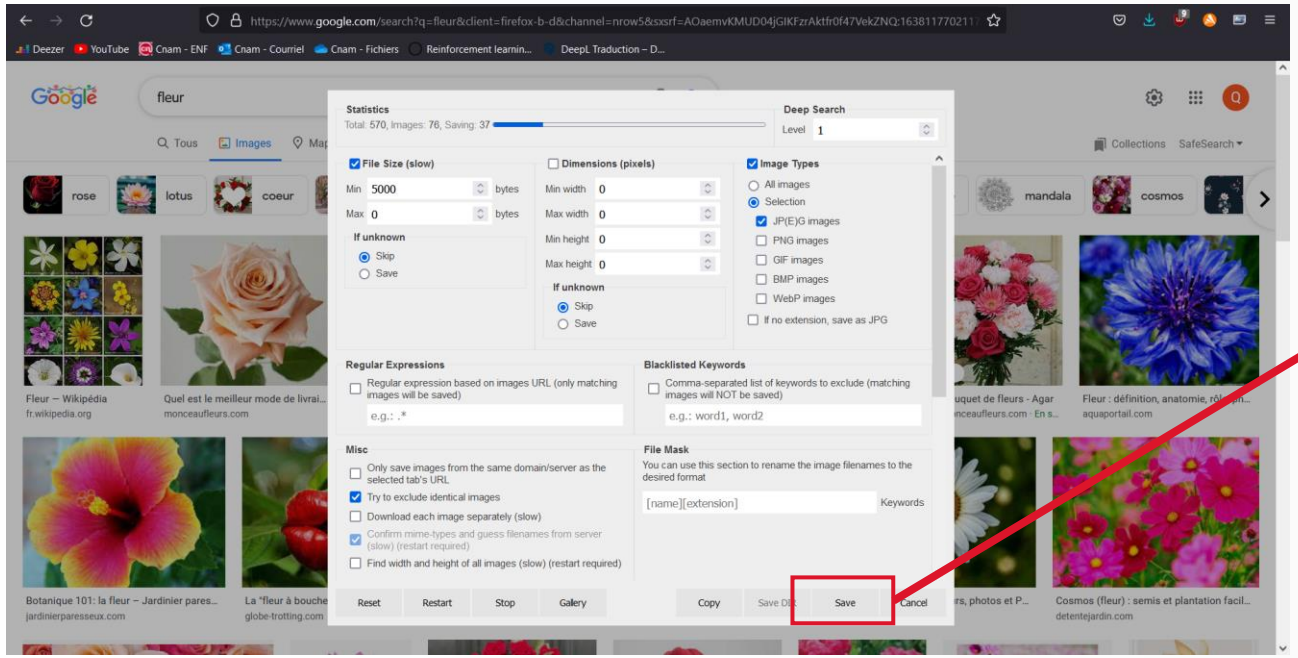


On remarque que « l'accuracy » est déjà très bonne sur le dataset de Validation (**95%**), mais également sur le dataset de Test (**89%**).
La matrice de confusion est vraiment très bonne, on constate que la 5ème classe (n°4, les éléphants) se prédit quasiment parfaitement, et pour les pires prédiction, le modèle se trompe moins d'1 fois sur 3.
Les résultats sont donc très satisfaisants.

VII - CLASSIFICATION PAR DEEP – DATA AUGMENTATION

Pour cette partie j'ai utilisé une extension de Mozilla Firefox « **Download All Image** »
Cette extension permet de sauvegarder, via Google Images, environ 50 images de chaque classe,
Pour les mettre dans le sous-dossier **Test** du dossier Wang



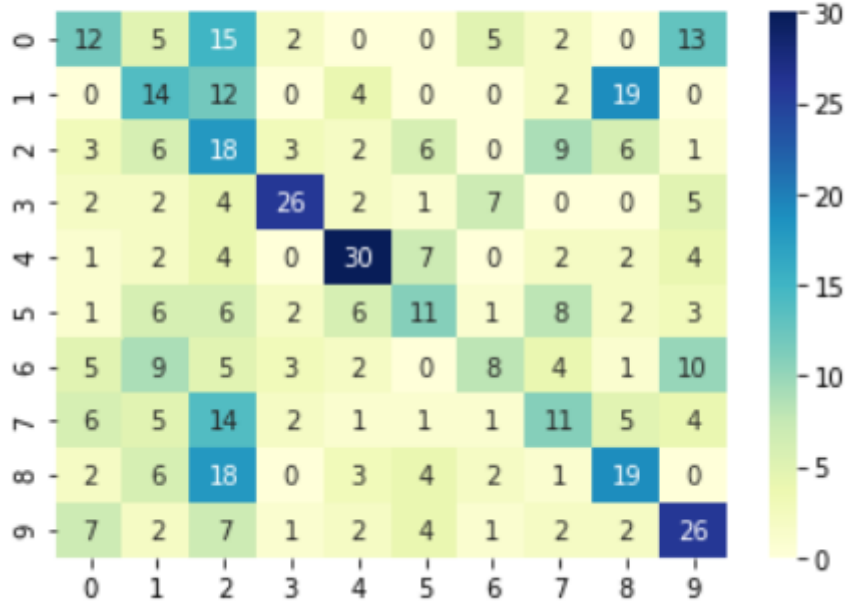


mesea
MAINTENANCE SEA TOURS · BORDEAUX

VII - CLASSIFICATION PAR DEEP – DATA AUGMENTATION

En utilisant donc les images de Test téléchargées en masse d'internet, on obtient ces résultats :

```
Accuracy : 0.341796875
AxesSubplot(0.125,0.125;0.62x0.755)
```



Malheureusement les prédictions ne sont pas bonnes car les images d'internet ne sont pas toutes ressemblantes à nos images d'entrainement.

Il faudrait prendre plus de temps pour bien sélectionner uniquement les images qui pourraient être reconnues.

Ou alors augmenter nos échantillons d'apprentissage…

MERCI DE VOTRE ATTENTION

mesea
MAINTENANCE SEA TOURS · BORDEAUX