

IA Intelligence Artificielle

- [Classification de bonbons avec TensorFlow](#)

Classification de bonbons avec TensorFlow

Essais de classification de bonbons par Intelligence Artificielle avec OpenCV et TensorFlow/Keras.

Note" open

Je ne suis pas un spécialiste de l'IA, seulement un utilisateur curieux. Cet article est un résumé de mes premiers essais avec Tensorflow. Les méthodes développées sont "largement" perfectibles mais peuvent donner des pistes à tout personne souhaitant se lancer avec Tensorflow.

Objectifs



Pour entraîner le modèle, il est nécessaire d'avoir suffisamment d'images de chaque confiserie à identifier. On effectuera de l'apprentissage supervisé, le label de chaque image utilisée pour l'entraînement sera connu.

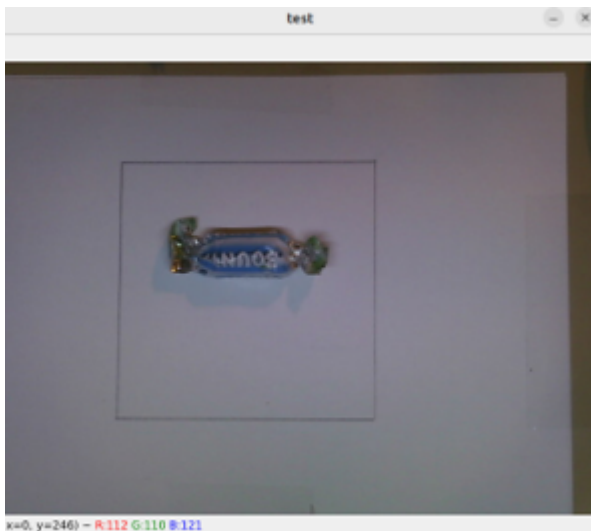
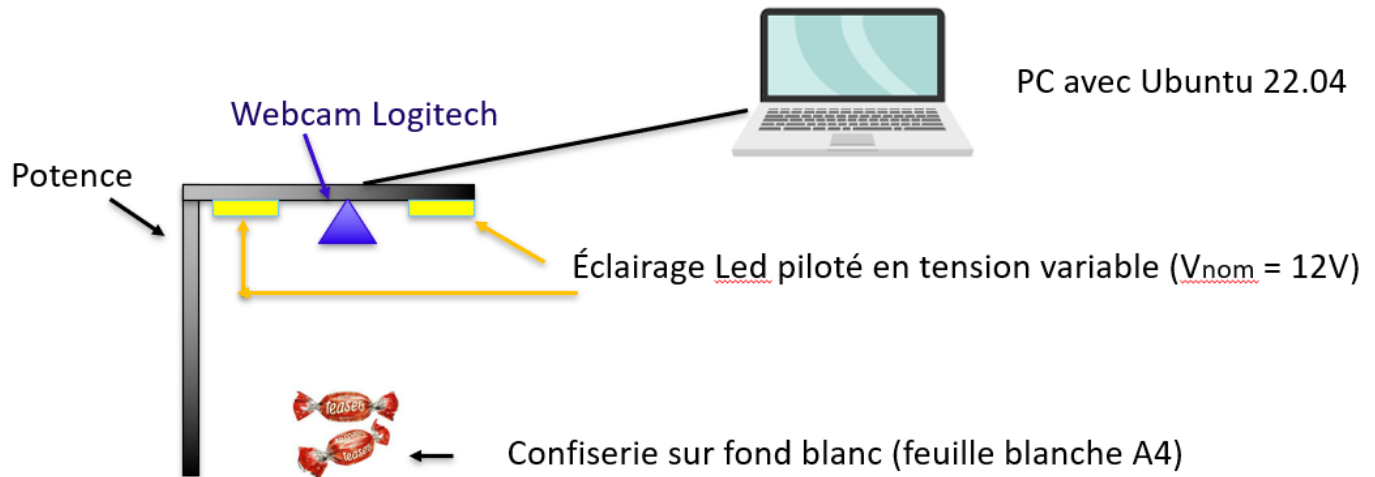
- Il y aura 4 Labels : **Bounty** , **Maltesers**, **Snickers**, **Twix**

Pour chaque confiserie:

- 660 images sont prises dont 250 en condition de lumière dégradée
- Ce qui fait $660 \times 4 = 2640$ images
- -> Il faut faire un programme permettant d'automatiser les prises de vue, sinon, on va y passer la journée.
- Pour rendre les images exploitables pour un réseau de neurone avec une puissance de PC classique, il faut limiter la résolution à 250 pixels x 250.

Le programme devra effectuer un « découpage » (Crop) pour extraire la zone utile d'image avec un format 250 x 250, car de base, la WebCam fait une capture en 640x480.

Installation expérimentale



- Une zone de travail de 12cm sur 12 cm est dessinée sur la feuille.
- Avec le crop à 250 x 250 pixels, on ne prendra comme image utile ce qui se trouve à l'intérieur du carré.

Logiciels

- Ubuntu 22.04 LTS installée sur une VirtualBox
- VisualStudioCode pour Coder en Python (ou autre éditeur au choix)
- Il faut installer pip3 pour installer facilement les libs python
- On installe tensorflow, numpy, opencv, matplotlib avec la commande `pip3 install`

Script pour l'acquisition des Datasets d'images

Le script python permettant la capture d'images pour constituer le DataSet d'images.

Quand on appuie sur espace :

- une image est prise
- Le crop en 250 x 250 est fait (position Y1:Y2 et X1:X2 des pixels pour la base du crop et le point d'arrivée)
- Cela enregistre l'image
- On incrémente le compteur d'image

Quand on appuie sur Echap :

- On arrête la webcam
- On détruit la fenêtre de capture

```
import cv2
cam = cv2.VideoCapture(0)

cv2.namedWindow("test")
img_counter = 0

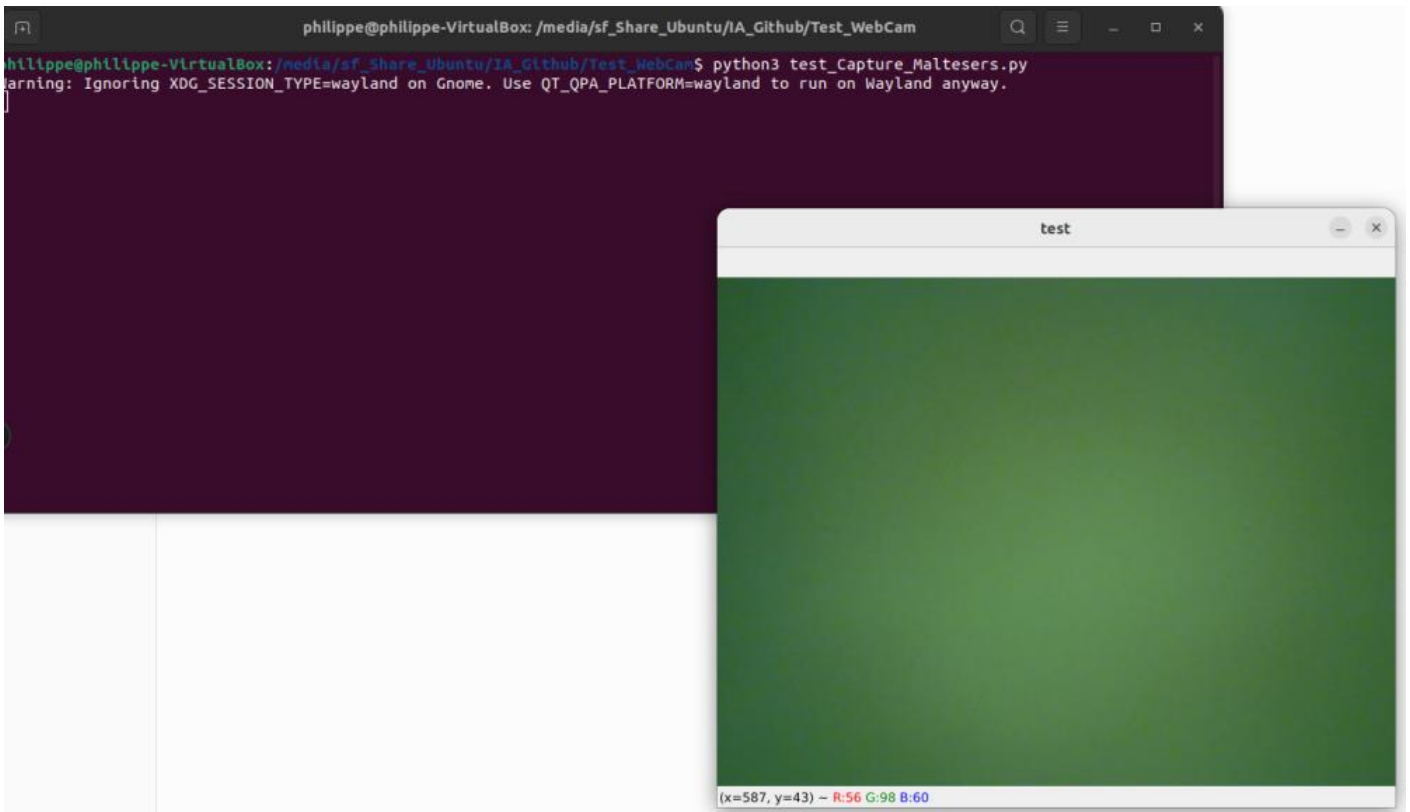
while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)

    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = "Maltesers_{}.png".format(img_counter)
        # Cropping an image Y1: Y2 , X1: X2
```

```
cropped_image = frame[115:365, 140:390]
cv2.imwrite(img_name, cropped_image)
print("{} written!".format(img_name))
img_counter += 1

cam.release()
cv2.destroyAllWindows()
```

- Pour exécuter le script de capture: utiliser la commande python3 test_Capture_Maltesers.py



Il faudra modifier le script python de capture pour avoir un nommage cohérent des images.

Grâce à cette méthode, 2 secondes suffisent pour capturer une image, soit tout de même une bonne 1h pour générer les 2640 images de DataSets.

Script Python pour générer le modèle avec TensorFlow+Keras

Le script utilisé pour générer le modèle est inspiré du tutoriel TensorFlow pour la classification d'images : [Tuto_classification_images](#)

L'adaptation de ce script à notre problématique de classification de bonbons est faite ici (cliquer pour ouvrir le code)

```
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
import matplotlib.pyplot as plt

data_dir = pathlib.Path('/home/philippe/Documents/DataSetsBonbons')
print(data_dir)
image_count = len(list(data_dir.glob('*/*.png')) )
print(image_count)
print(tf.__version__)

batch_size = 660
img_height = 250
img_width = 250

train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```

class_names = train_ds.class_names
print(class_names)

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
num_classes = len(class_names)

model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

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

model.summary()
epochs=30
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

```

```

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

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

model.save('saved_model/my_model.h5')

```

Ce script enregistre le modèle généré dans un répertoire "saved_model" sous le nom my_model.h5

Le nombre d'époch est **augmenté à 30** pour avoir une bonne convergence la précision d'entraînement et celle de validation.


```

Epoch 26/30
4/4 [=====] - 48s 11s/step - loss: 0.0717 - accuracy: 0.9777 - val_loss: 0.1002 - val_accuracy: 0.9735
Epoch 27/30
4/4 [=====] - 49s 15s/step - loss: 0.0559 - accuracy: 0.9811 - val_loss: 0.1116 - val_accuracy: 0.9678
Epoch 28/30
4/4 [=====] - 48s 11s/step - loss: 0.0426 - accuracy: 0.9943 - val_loss: 0.1010 - val_accuracy: 0.9697
Epoch 29/30
4/4 [=====] - 48s 11s/step - loss: 0.0402 - accuracy: 0.9896 - val_loss: 0.0946 - val_accuracy: 0.9621
Epoch 30/30
4/4 [=====] - 47s 11s/step - loss: 0.0286 - accuracy: 0.9986 - val_loss: 0.1117 - val_accuracy: 0.9621
Model: "sequential"

```

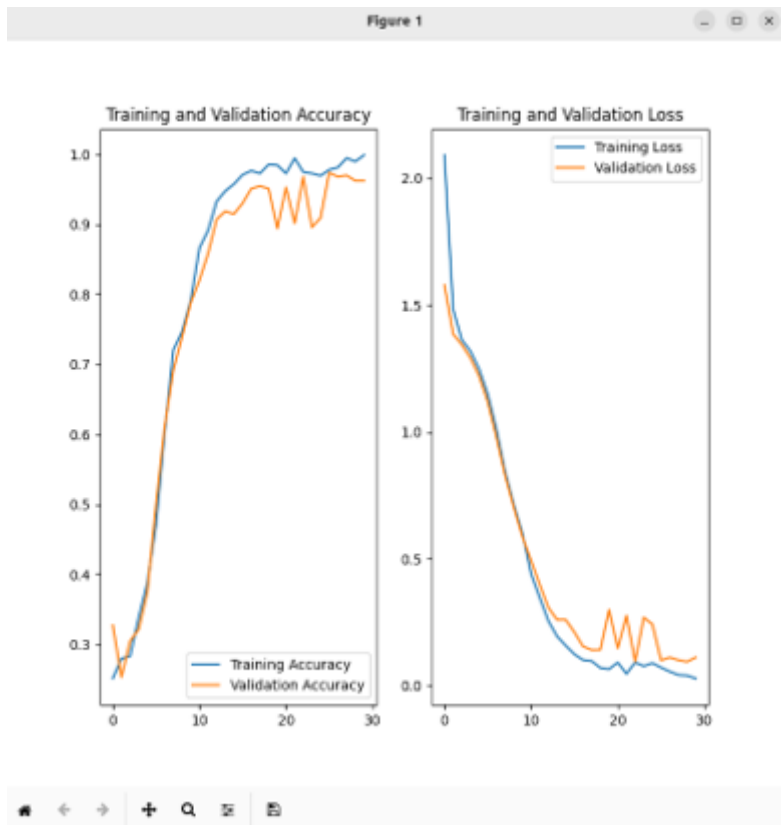
Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	7872640
dense_1 (Dense)	(None, 4)	516

```

Total params: 7,896,740
Trainable params: 7,896,740
Non-trainable params: 0

```

On remarque une bonne convergence entre l'entraînement et la validation



Script pour tester le modèle généré

Le script pour tester le modèle généré est disponible ici :

```
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf

import sys
import cv2

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

cam = cv2.VideoCapture(0)
cv2.namedWindow("test")

new_model =
tf.keras.models.load_model('/home/philippe/Documents/Test_TF_on_new_model/my_model.h5')
class_names=['Bounty','Maltesers','Snickers','Twix']
#new_model.summary()
img_height = 250
img_width = 250

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)
    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        # Cropping an image Y1: Y2 , X1: X2
        cropped_image = frame[115:365, 140:390]
        cv2.imwrite("test.png", cropped_image)
        print("written!")
```

```

img =
tf.keras.utils.load_img(' /home/philippe/Documents/Test_TF_on_new_model/test.png' , target_size=(
img_height, img_width))
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch
predictions = new_model.predict(img_array)
score = tf.nn.softmax(predictions[0])
print("This image most likely belongs to {} with a {:.2f} percent
confidence.".format(class_names[np.argmax(score)], 100 * np.max(score)))
cv2.waitKey(1)
cam.release()
cv2.destroyAllWindows()

```

(Il faudra adapter ce script à votre environnement)

Fonctionnement du script:

- On lance le script avec `python3 test_du_model.py`
- Une fenêtre WebCam s'ouvre
- Un appui sur Espace permet de capturer la confiserie
- L'analyse se fait
- Le résultat de détection est affiché avec un indice de confiance
- Le cycle se répète pour chaque appui sur la touche espace
- Un appui sur la touche Echap permet de quitter le script

Le modèle répond juste pour des conditions d'éclairage comprises entre normales et dégradées (réduction de l'alimentation des LEDs d'éclairage de 12V à 8V) :



```
Written!  
1/1 [=====] - 0s 27ms/step  
This image most likely belongs to Bounty with a 99.12 percent confidence.
```

```
Written!  
1/1 [=====] - 0s 30ms/step  
This image most likely belongs to Maltesers with a 99.94 percent confidence.
```



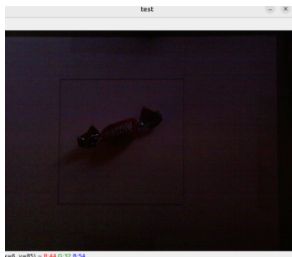
```
Written!  
1/1 [=====] - 0s 48ms/step  
This image most likely belongs to Twix with a 88.89 percent confidence.
```

```
Written!  
1/1 [=====] - 0s 28ms/step  
This image most likely belongs to Snickers with a 100.00 percent confidence.
```

Toutes les confiseries sont reconnues avec un indice de confiance compris entre 88.9% et 100%

Essais du modèle en mode TRÈS dégradé

On est dans un cas extrême, quand il n'y a plus d'éclairage, tous les bonbons sont des Snickers à 100% (ce qui est étrange, l'indice de confiance aurait du être plus faible...)



```
Written!  
1/1 [=====] - 0s 38ms/step  
This image most likely belongs to Snickers with a 100.00 percent confidence.
```

Bilan

- Les outils IA sont aujourd'hui suffisamment accessibles pour qu'un débutant puisse générer un modèle qui semble robuste.
- Les lignes de code Python sont très simples et se limitent à des appels de fonctions, aucune algorithmie complexe ne fut programmée.
- Il faut un PC avec 16 Go de RAM est confortable pour calculer le modèle, une fois le modèle calculé plus besoin d'autant de RAM.
- Le modèle semble étonnamment robuste, surtout dans des bonnes conditions d'éclairage.