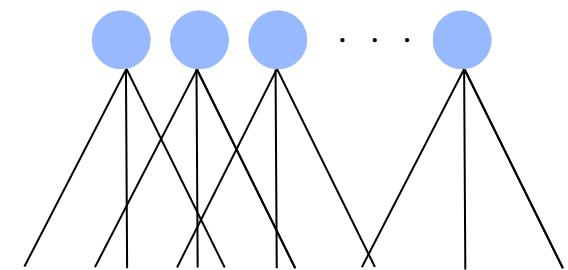
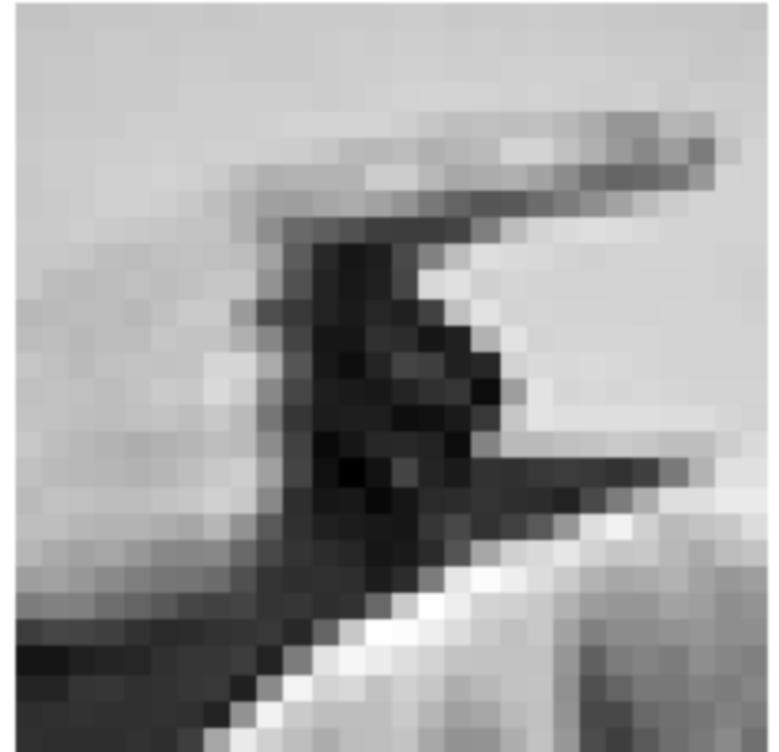


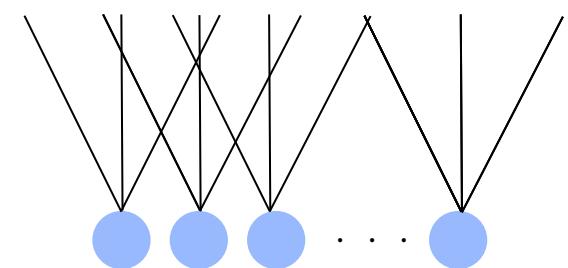
MODÈLE DE RECONNAISSANCE GESTUELLE

A partir d'un jeu de donnée de langage des signes, nous avons modéliser un **CNN (Convolutional Neural Network)** pour détecter les gestes de la main.

Les principales difficultés furent d'implémenter un tel réseau sur un ESP32 et d'optimiser ses résultats.

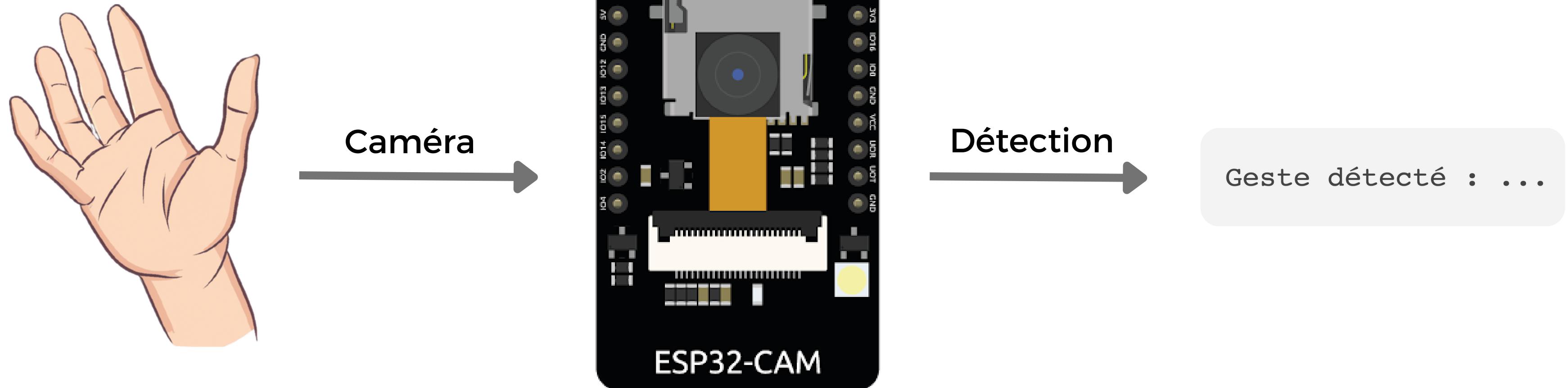


CNN



detected label : ?

OBJECTIF : RECONNAISSANCE GESTUELLE



SOLUTION : RÉSEAU DE NEURONNE

Data Set Open Source

Donnée PNG, CSV, ...

Label

- > 10k données d'entraînement
- > 1k données de validation
- > 1k données de test

Créer son propre Data Set

Donnée PNG, CSV, ...

Label

~200 données labélisables / heure

1. Prise de la photo
2. Adaptation de la donnée
3. Labélisation de la donnée

Téléchargeable en 1 clic

Grand nombre de données

Données labélisées imposent le modèle à choisir

Biais de qualité des données

Labélisation des données en fonction du modèle choisi

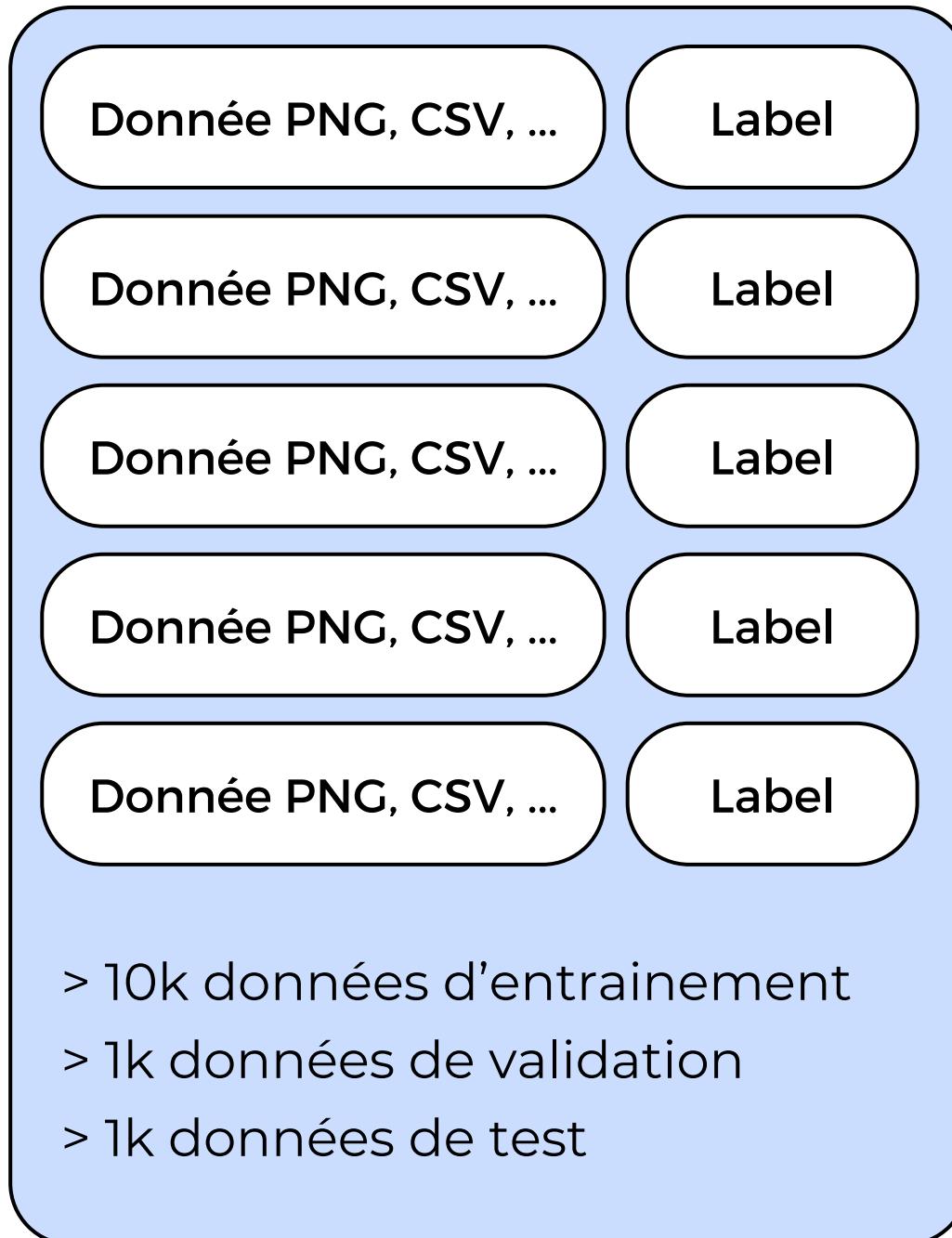
Pas de biais de qualité des données

Manque d'exhaustivité des données

Temps conséquent de récolte et de labélisation des données

DATA SET CHOISI

Data Set Open Source



✓ Téléchargeable
en 1 clic

✓ Grand nombre
de données

✗ Données labélisées
imposent le modèle
à choisir

✗ Biais de qualité
des données

ASL Data Set

ASL = American Sign Language



ASL DATA SET

Différents labels ASL détectables

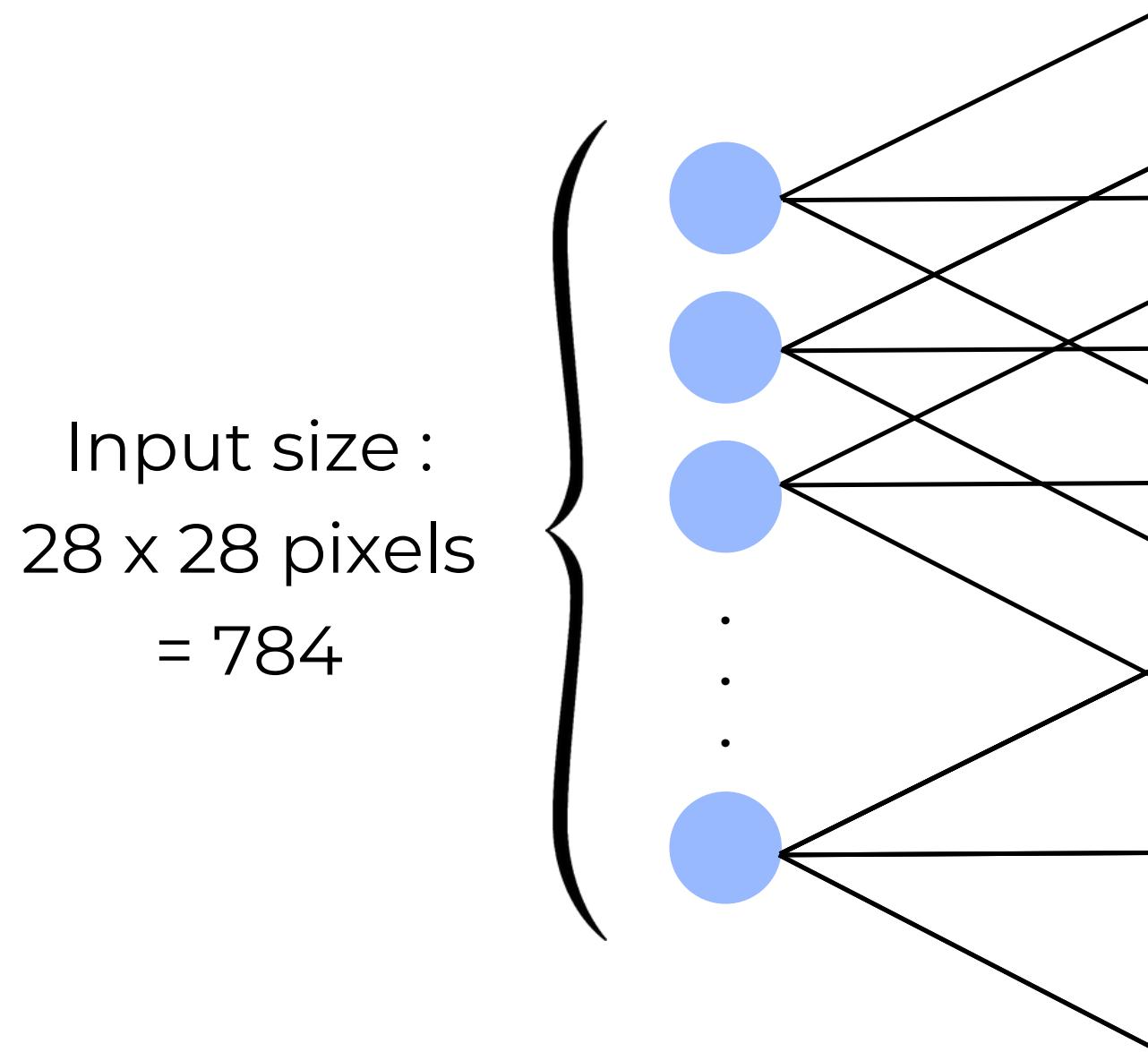


Données dans le Data Set



- 27 455 données d'entraînement
- 7 172 données de test et de validation
- 28 x 28 pixels
- Grayscale 0 - 255
- 24 labels (J et Z exclus)

DONNÉES D'ENTRÉE ET DE SORTIE DU MODÈLE



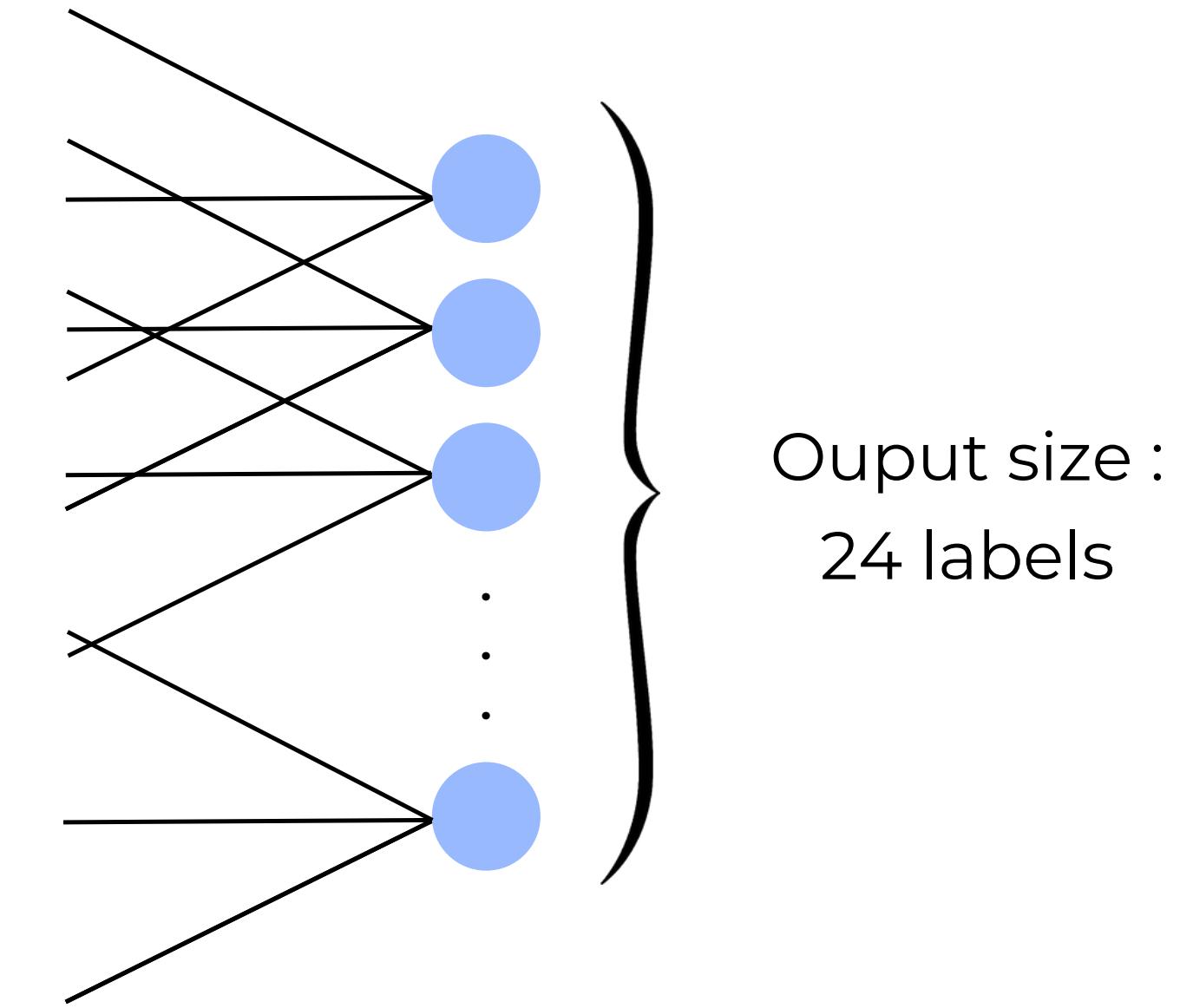
Input size :
28 x 28 pixels
 $= 784$

Input type : int

Input plage : 0 - 255

Valeur de gris

Hidden layers : 8
Parameters : $> 100k$



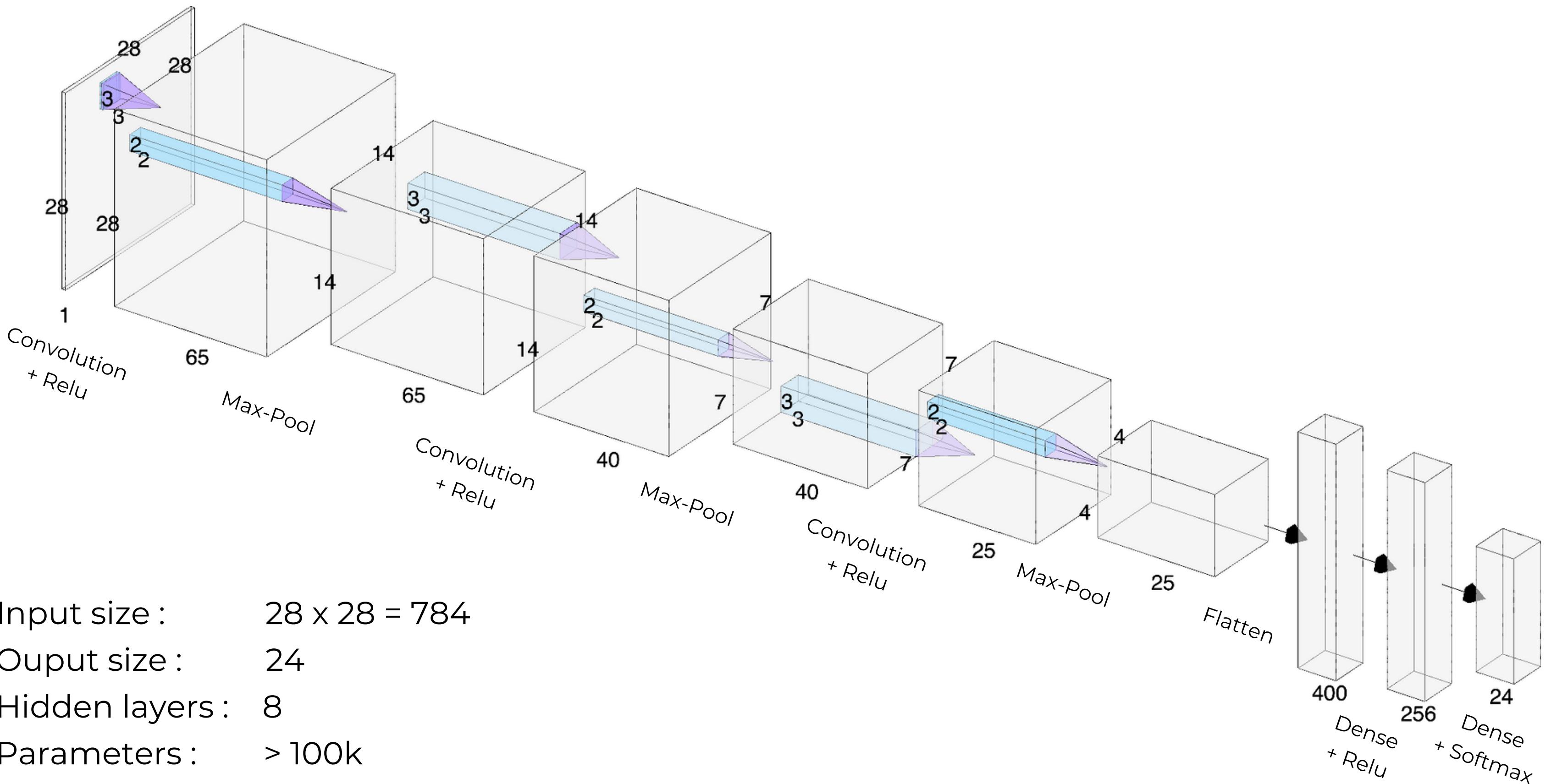
Output size :
24 labels

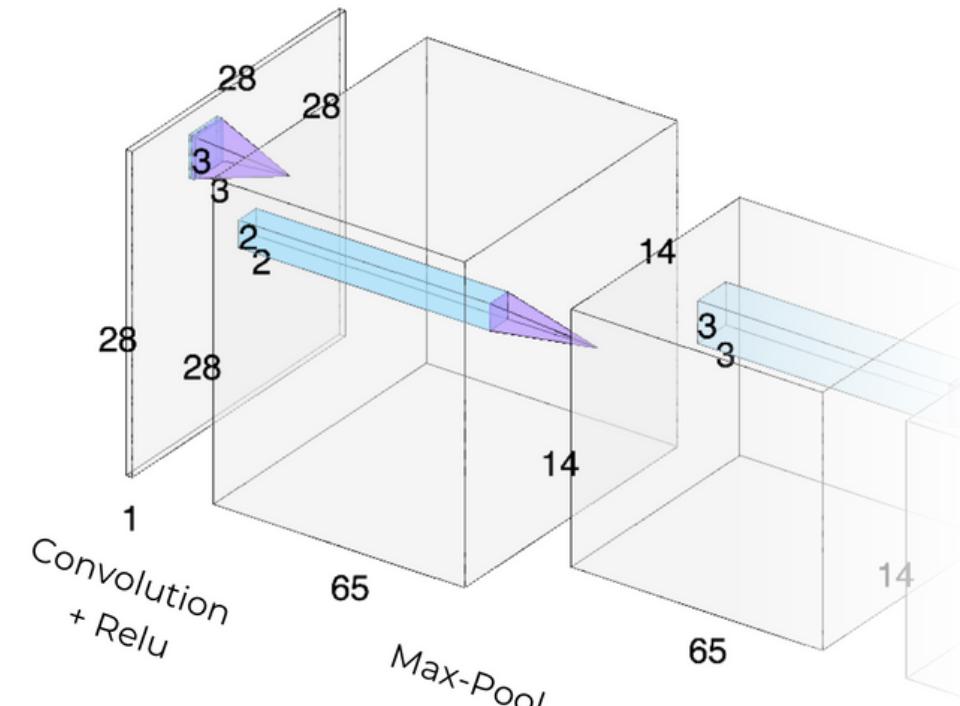
Output type : float

Output plage : 0 - 1

Probabilité de détection

MODÈLE DU RÉSEAU DE NEURONNE : CNN

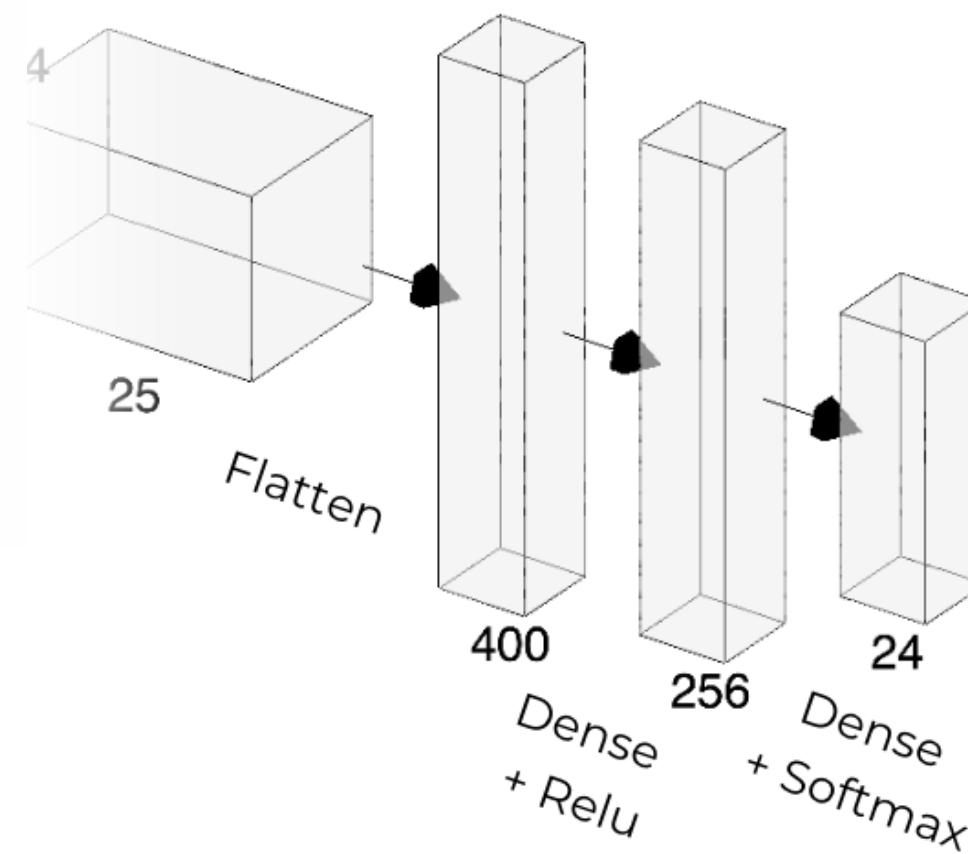




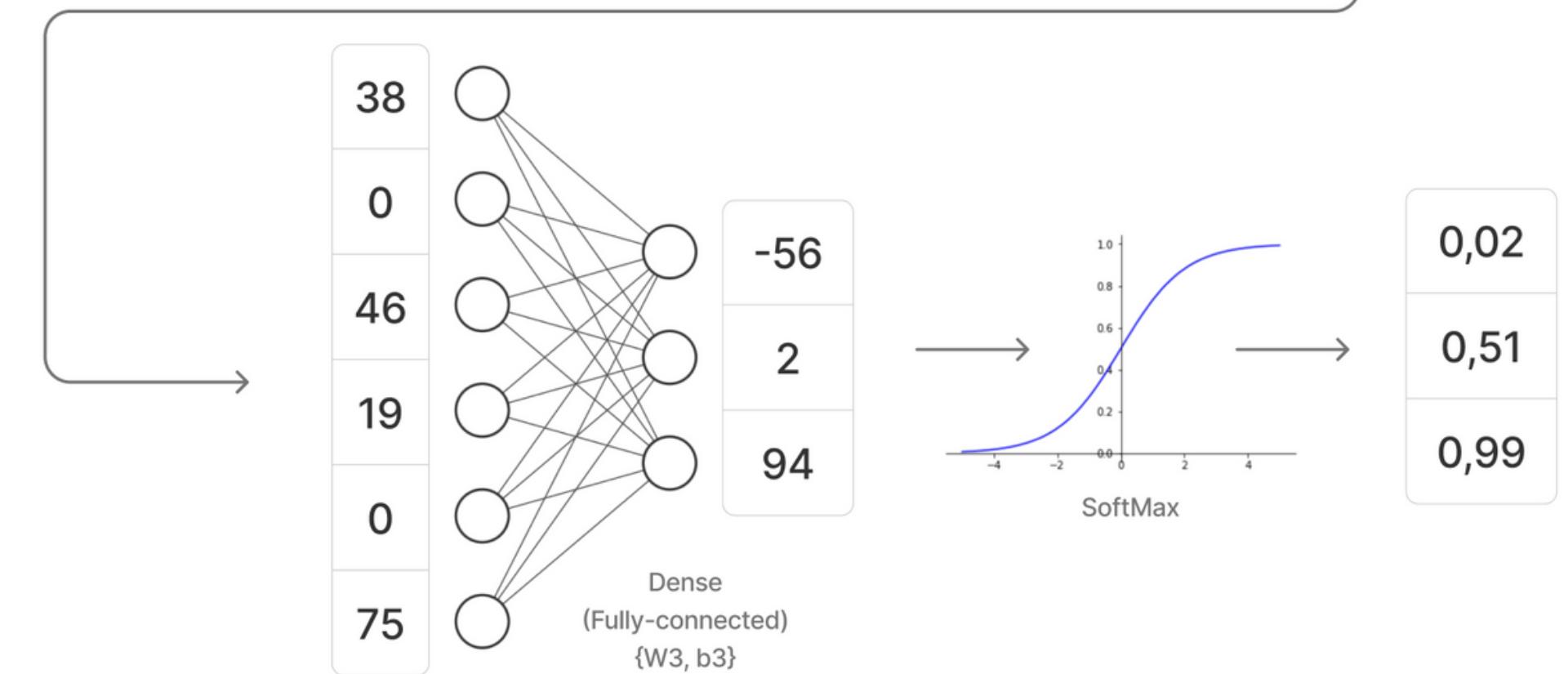
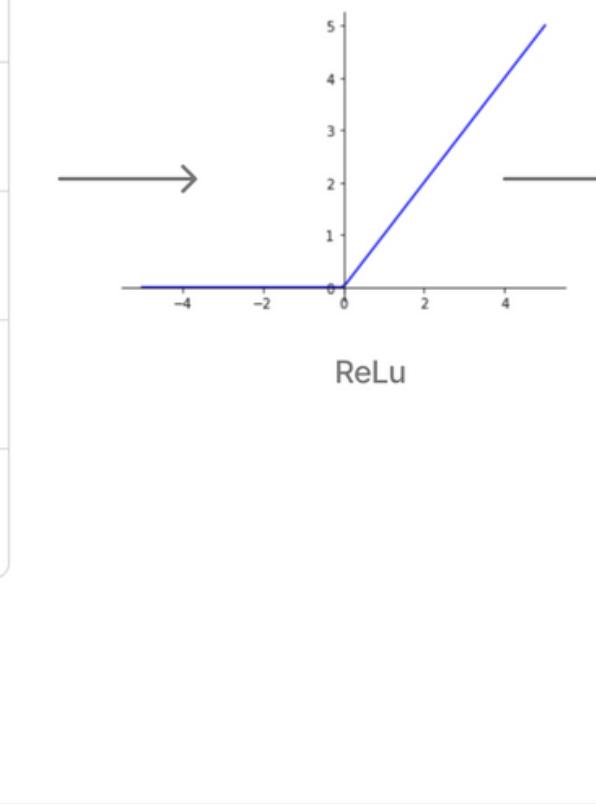
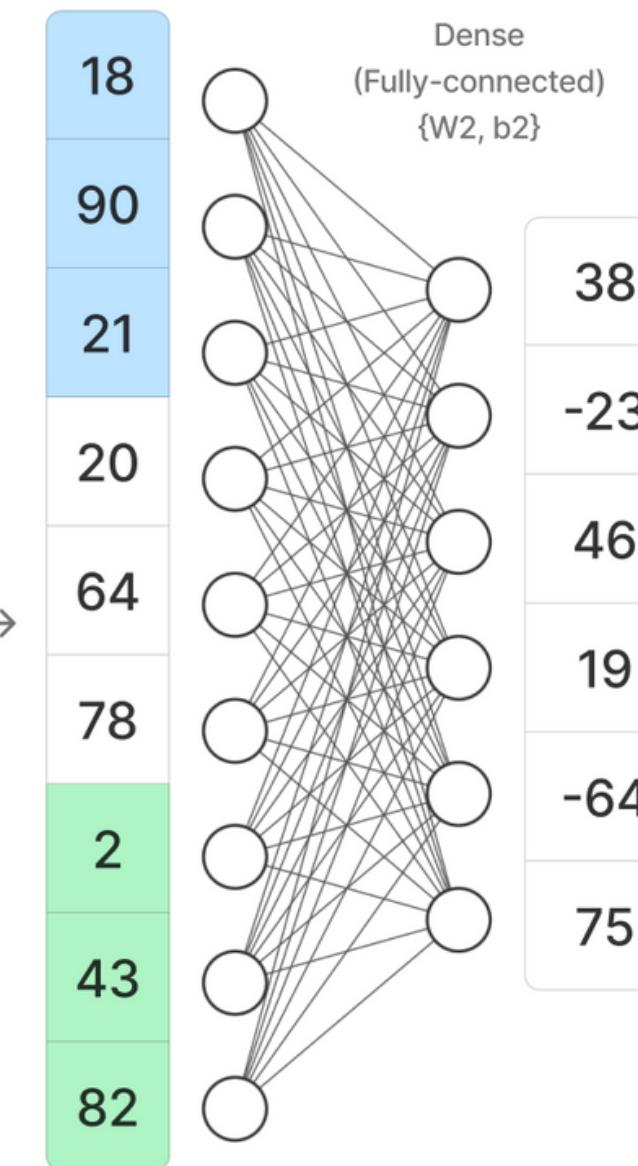
Convolution
+ Relu

Max-Pool





Flatten



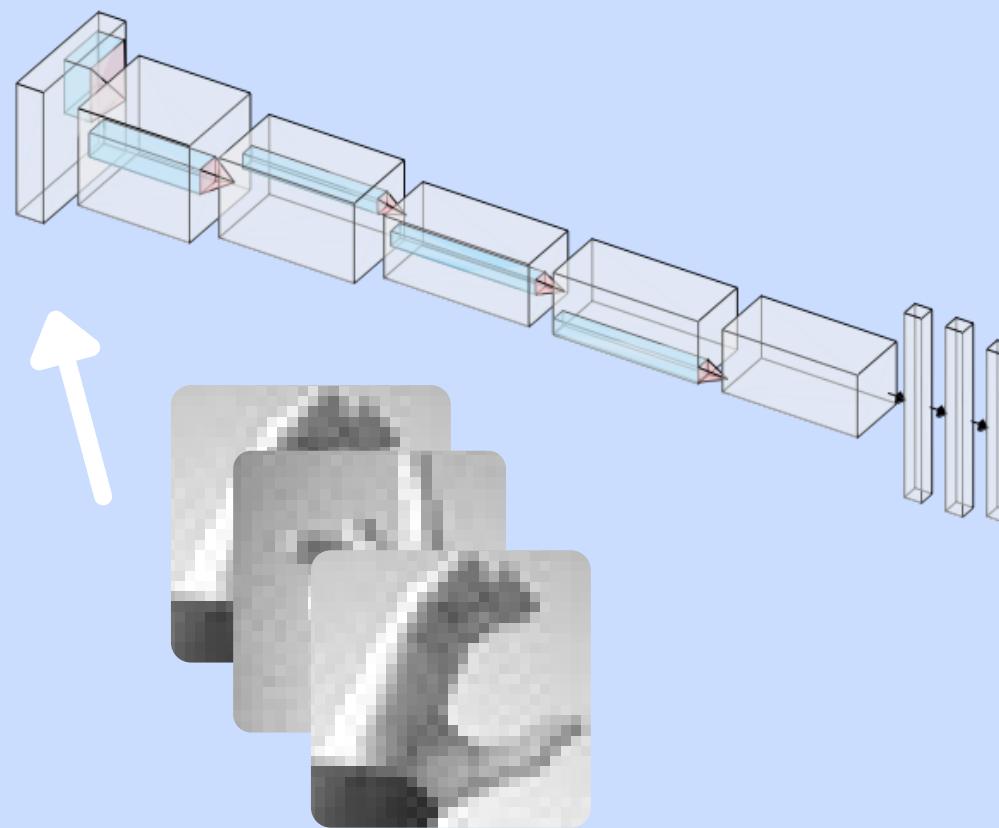
PHASE D'ENTRAINEMENT SUR PC AVEC JUPYTER

notebook.ipynb



Keras
(Python API)

Création et entraînement du modèle à partir des données.



TensorFlowLite
(Python API)

Converti le modèle à l'aide du TensorFlowLite Converter en un tableau d'octets interprétable par l'API C++.

model.cpp

```
alignas(8) const unsigned char model[] = {
    0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c,
    0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10,
    0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00,
    0x88, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00,
    0xe4, 0x31, 0x02, 0x00, 0x74, 0x4e, 0x02,
    0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00,
    0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72,
    0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74,
    0x04, 0x00, 0x00, 0x00, 0x94, 0xff, 0xff,
    0x04, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00,
```

PHASE D'INFÉRENCE SUR ESP32

main.cpp



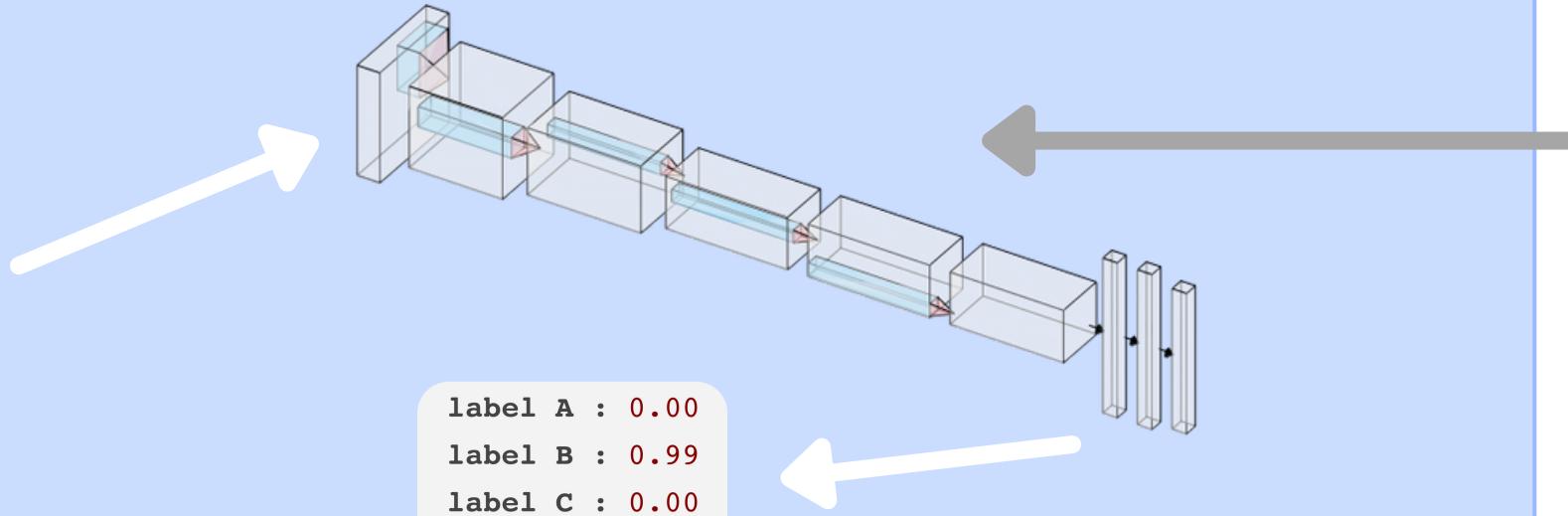
ESP32 SDK

Capture d'une image depuis le périphérique caméra de l'ESP32-CAM. Adaptation de la donnée : réduction de la taille, format grayscale.



TensorFlowLite
(C++ API)

Interprétation du modèle (model.cpp) en définissant les différentes couches. Inférence après ajout de nouvelles données d'entrées et récupération des données de sorties.

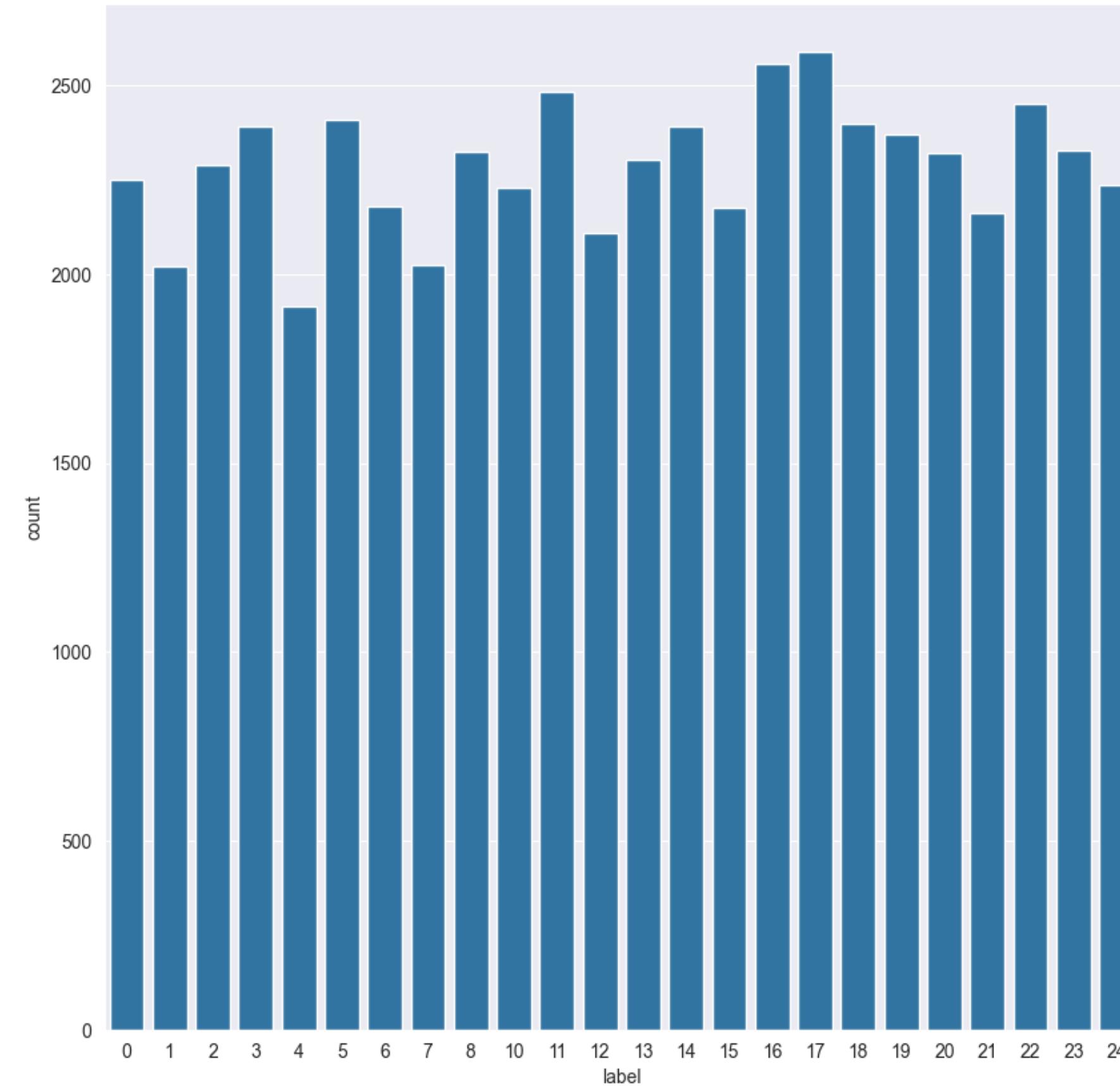


model.cpp

```


```

RÉPARTITION DES DONNÉES D'ENTRAÎNEMENT

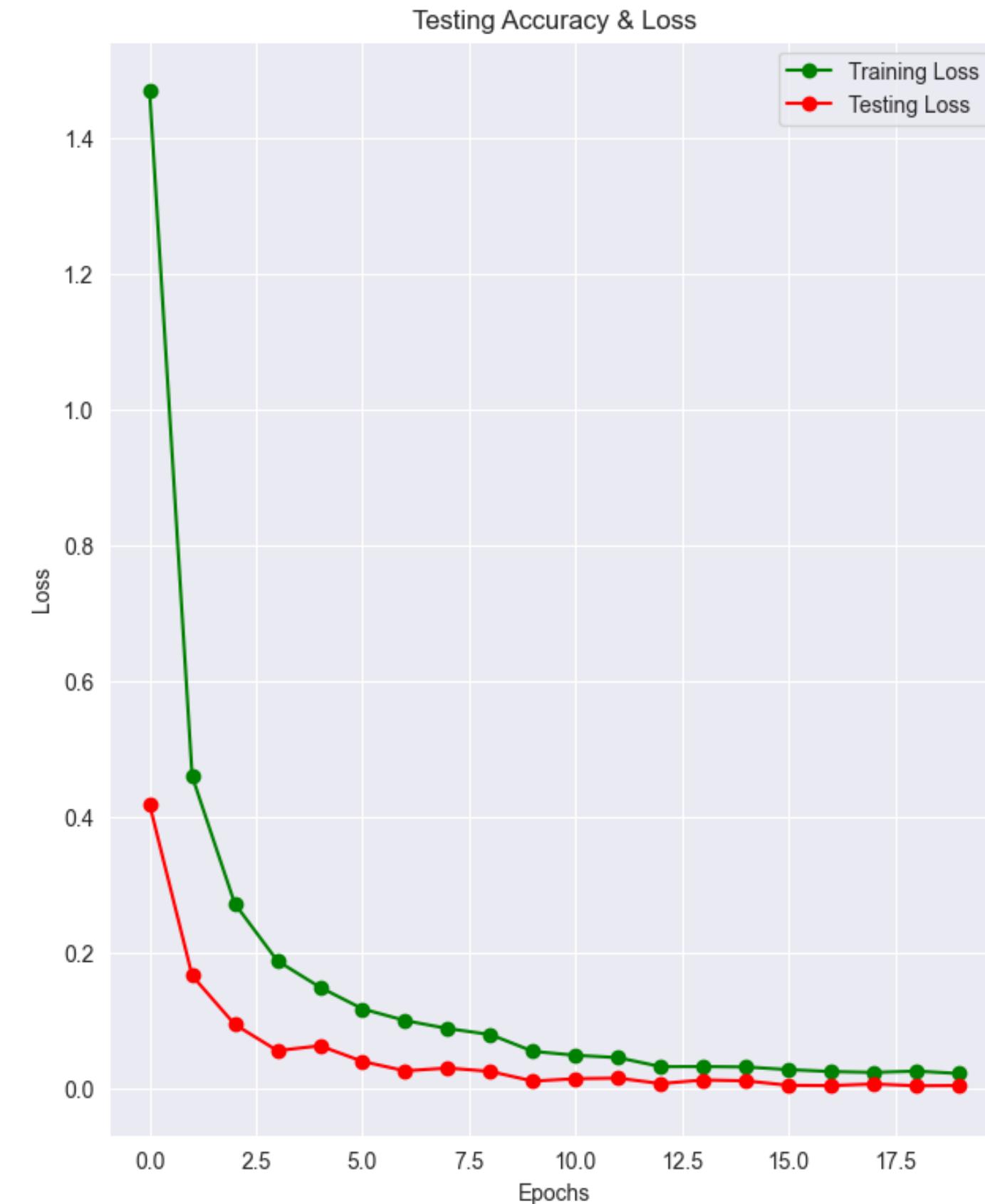
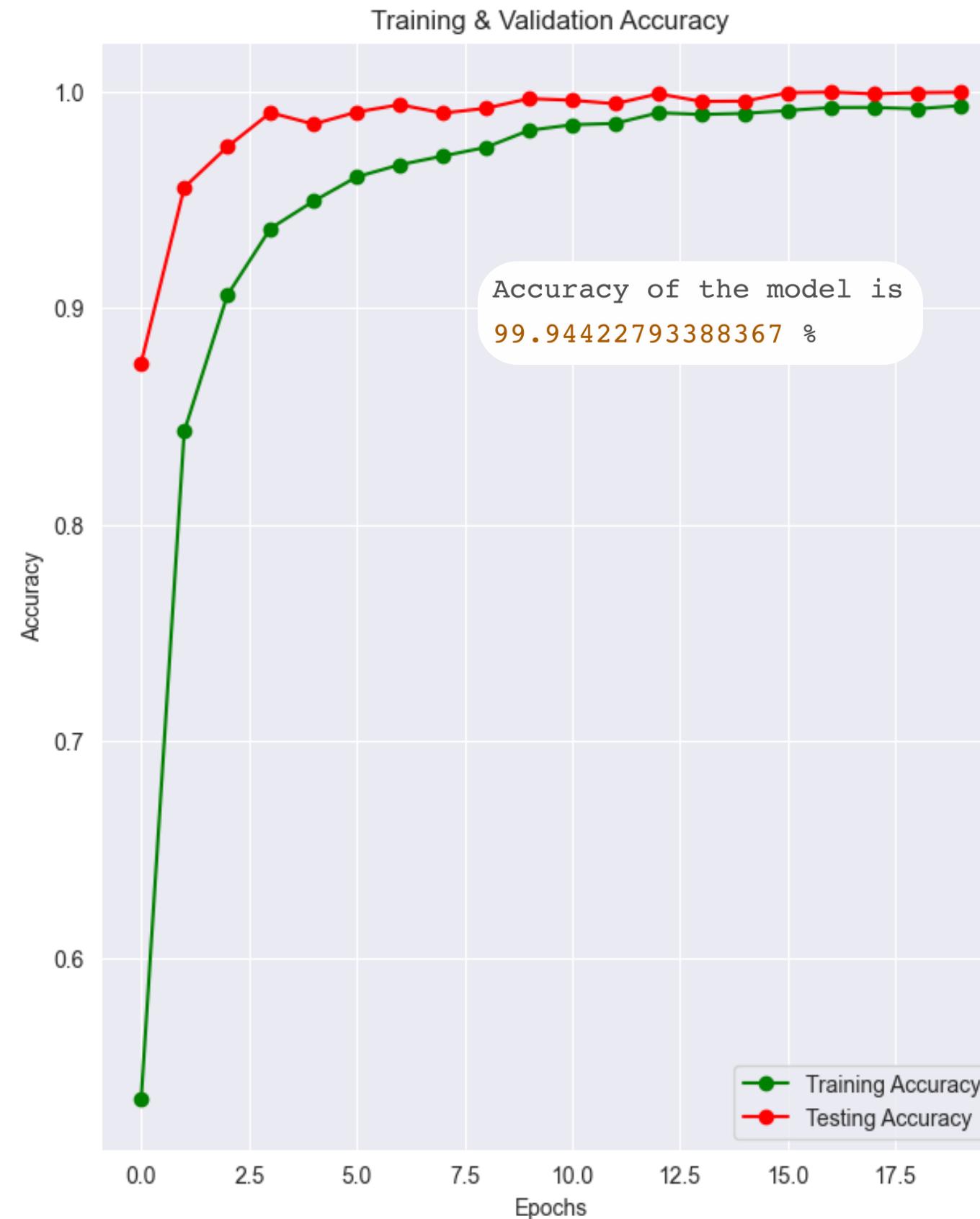


AUGMENTATION DES DONNÉES D'ENTRAÎNEMENT

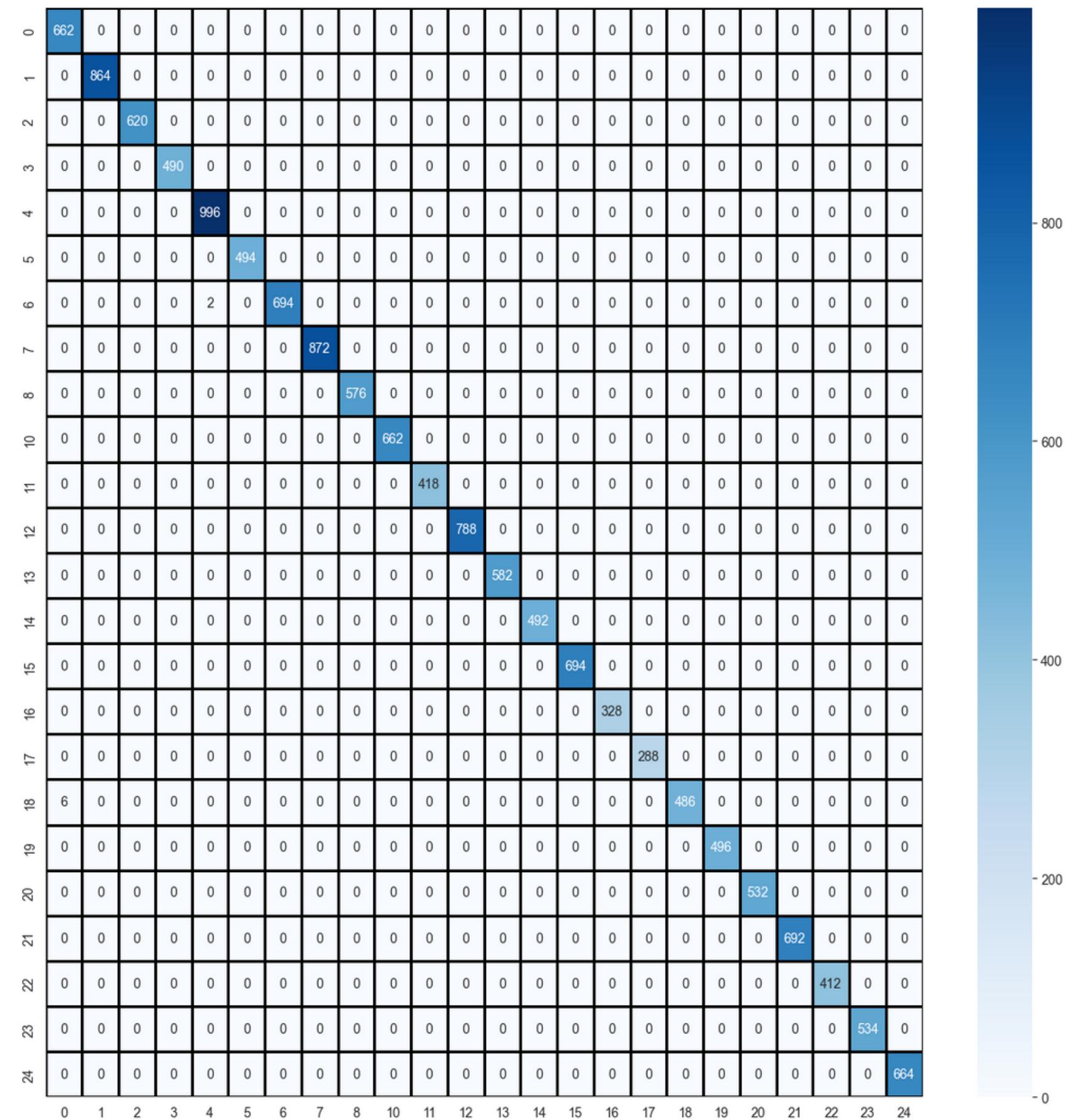
```
1 # With data augmentation to prevent overfitting
2
3 datagen = ImageDataGenerator(
4     featurewise_center=False, # set input mean to 0 over the dataset
5     samplewise_center=False, # set each sample mean to 0
6     featurewise_std_normalization=False, # divide inputs by std of the dataset
7     samplewise_std_normalization=False, # divide each input by its std
8     zca_whitening=False, # apply ZCA whitening
9     rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
10    zoom_range = 0.1, # Randomly zoom image
11    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
12    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
13    horizontal_flip=True, # randomly flip images
14    vertical_flip=False) # randomly flip images
15
16
17 datagen.fit(x_train)
```

Executed at 2023.12.28 17:19:41 in 58ms

RÉSULTATS D'ENTRAÎNEMENT (~4 MIN)



RÉSULTATS DE TEST



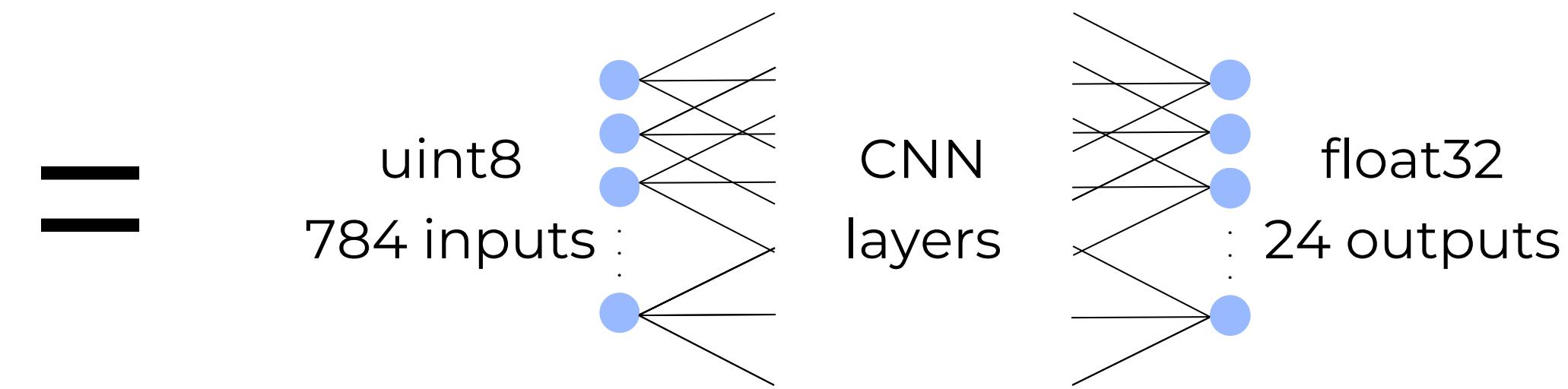
CONVERSION KERAS - TENSORFLOW LITE

```
1 import tensorflow as tf
2
3 converter = tf.lite.TFLiteConverter.from_keras_model(model) # trained keras model
4 converter.representative_dataset = representative_dataset
5 converter.optimizations = [tf.lite.Optimize.DEFAULT]
6 converter.inference_input_type = tf.uint8
7 converter.inference_output_type = tf.float32
8
9 tflite_model = converter.convert()
```

Test accuracy TFLITE model : 100.0 %

model.cpp

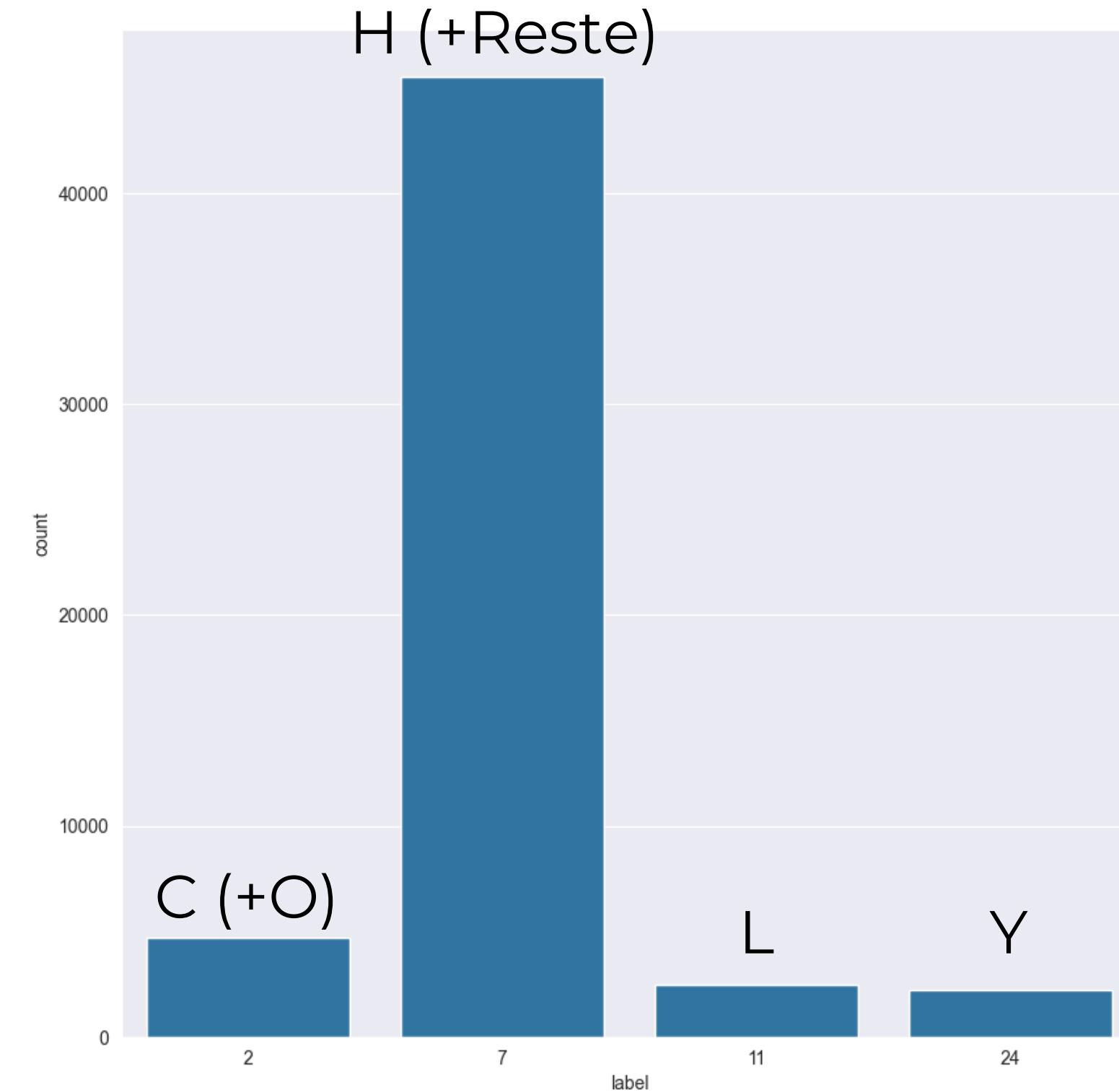
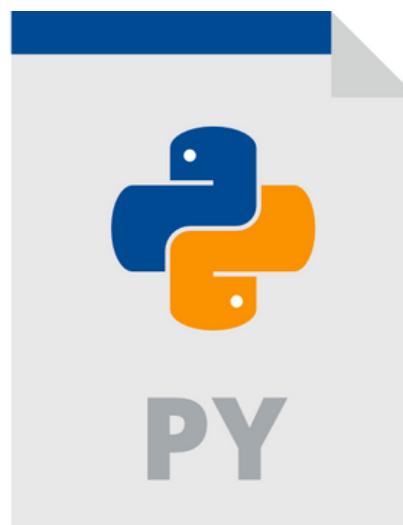
```
    0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c,  
    0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10,  
    0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00,  
    0x88, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00,  
    0xe4, 0x31, 0x02, 0x00, 0x74, 0x4e, 0x02,  
    0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00,  
    0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00,  
    0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72,  
    0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74,  
    0x04, 0x00, 0x00, 0x00, 0x94, 0xff, 0xff,  
    0x04, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00}
```



RÉSULTATS SUR ESP32

Lettre	Essai1	Essai2	Essai3	Essai4	Essai5	Essai6	Essai7	Essai8
Rien	H	G	G	G	G	H	G	H
A	G	X	P	E	G	G	X	G
B	X	X	X	X	X	X	B	X
C	C	C	C	C	H	C	C	
D	T	T	T	T	T	D	T	D
E	H	X	X	X	X	H	H	X
F	F	F	F	F	G	F	F	G
G	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H
I	D	T	D	T	T	D	T	T
K	T	T	T	T	T	T	K	T
L	L	L	L	L	H	L	L	L
M	H	X	T	Q	O	X	T	Q
N	Q	X	T	H	H	H	T	H
O	O	O	O	O	C	O	C	C
P	H	H	H	H	H	H	H	H
Q	P	P	P	P	P	Q	P	P
R	T	D	K	K	K	D	K	K
S	O	O	X	X	X	O	O	X
T	P	T	T	T	T	P	H	T
U	R	R	R	R	R	G	R	R
V	K	K	K	K	T	T	K	K
W	Y	Y	Y	K	W	W	Y	K
X	O	H	X	X	X	H	X	O
Y	T	Y	Y	Y	Y	Y	Y	Y

OPTIMISATION 1: RÉAJUSTEMENT DES LABELS



OPTIMISATION 2: DUPLICATION NEGATIVE DES DONNÉES

Différents labels ASL détectables



Exemple aléatoire de donnée
d'entraînement pour chaque label



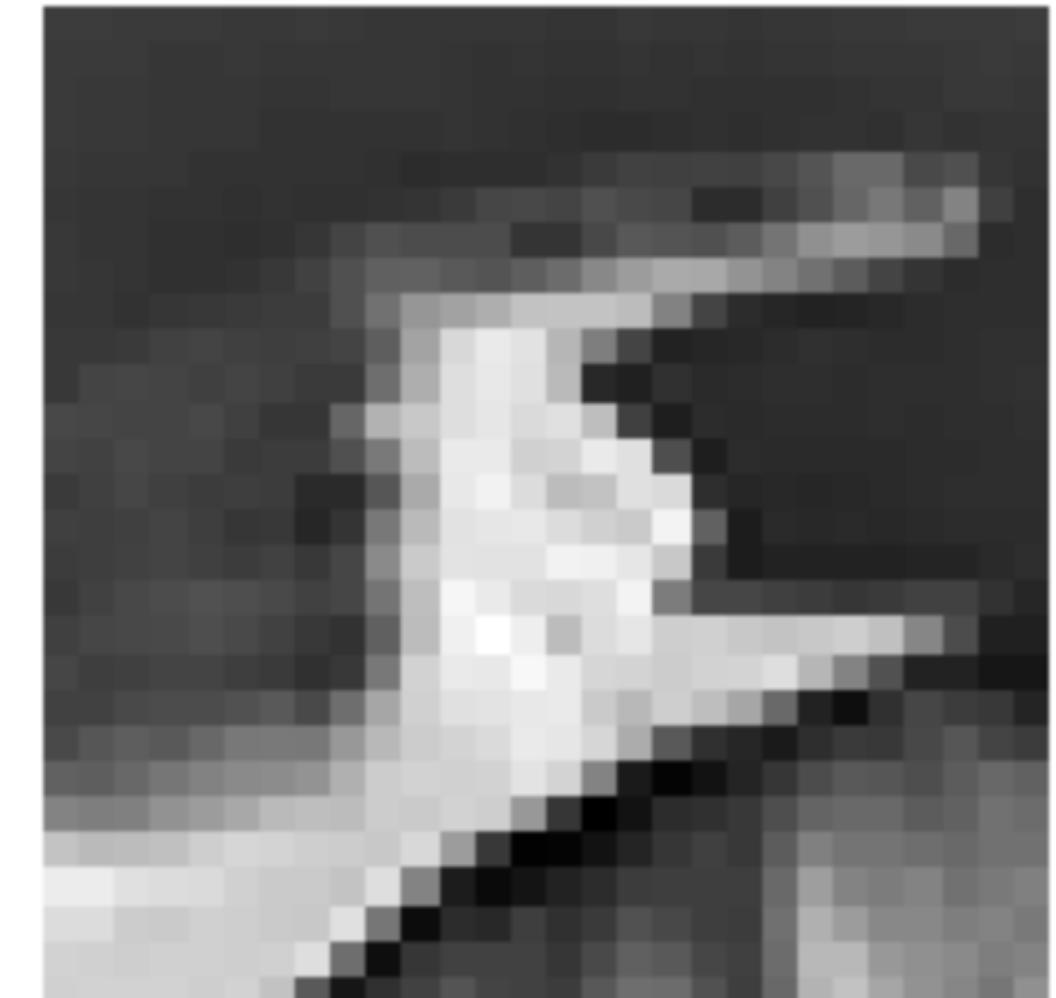
OPTIMISATION 2: DUPLICATION NEGATIVE DES DONNÉES



label : G

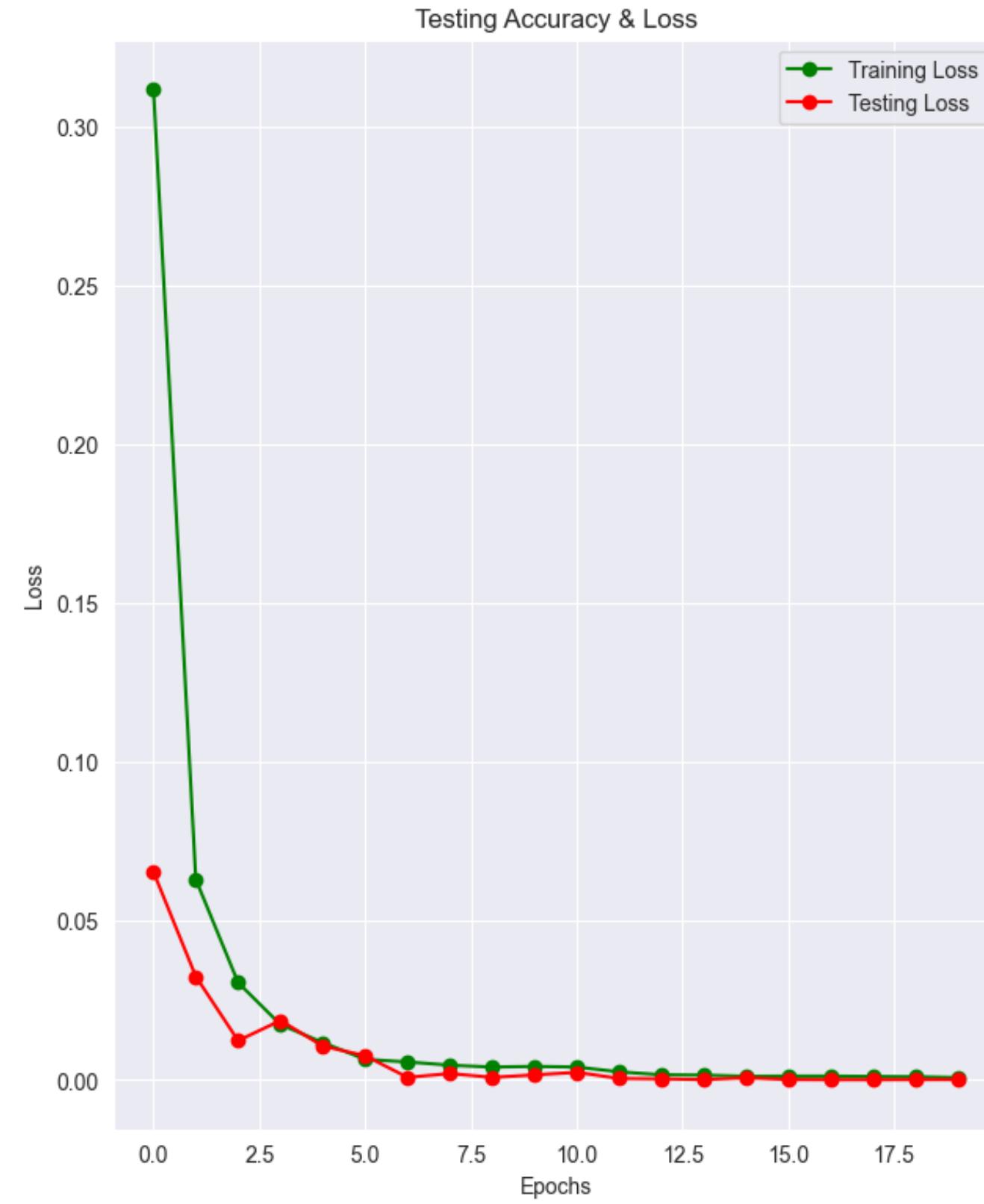
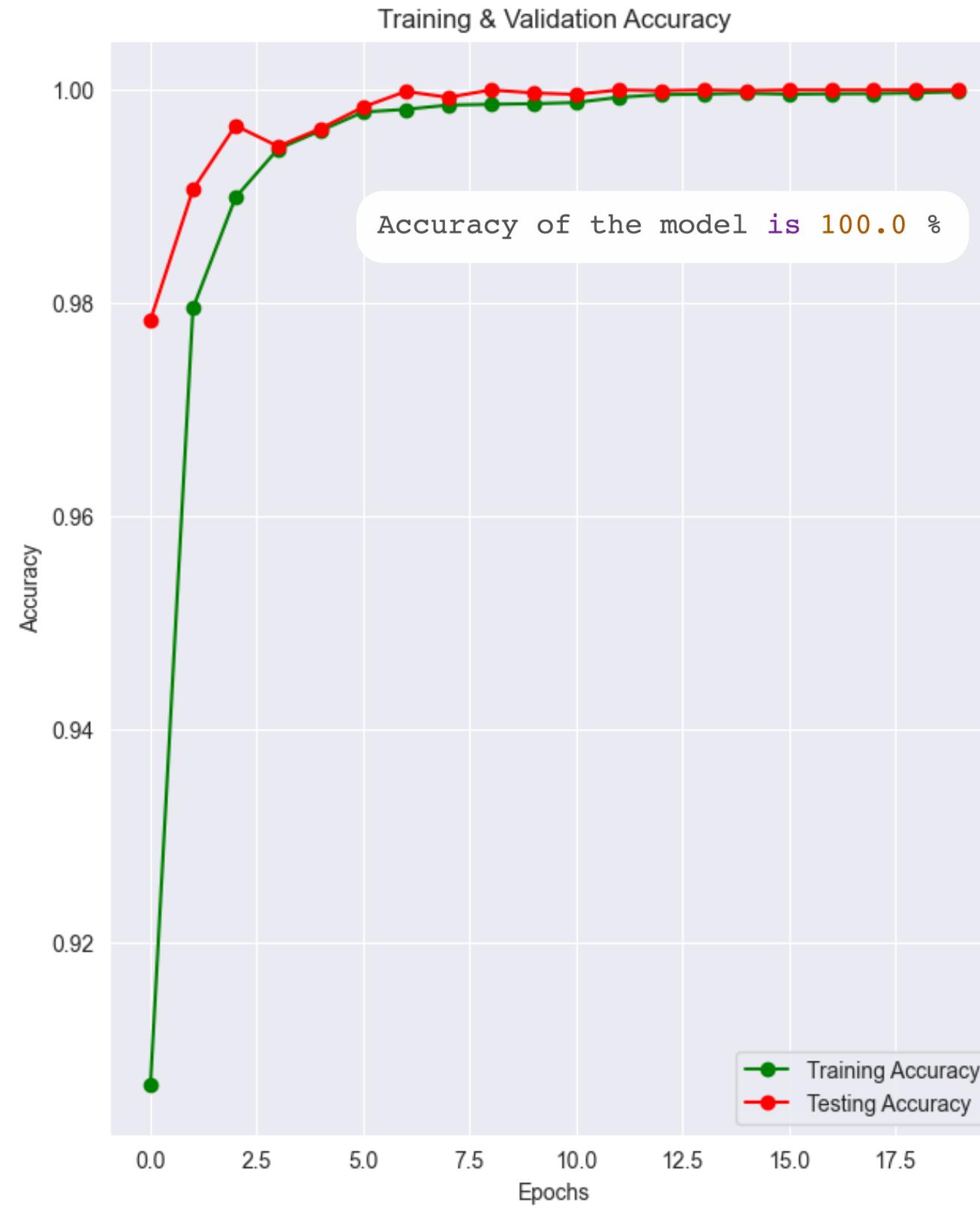


$\text{pixel}^* = 255 - \text{pixel}$

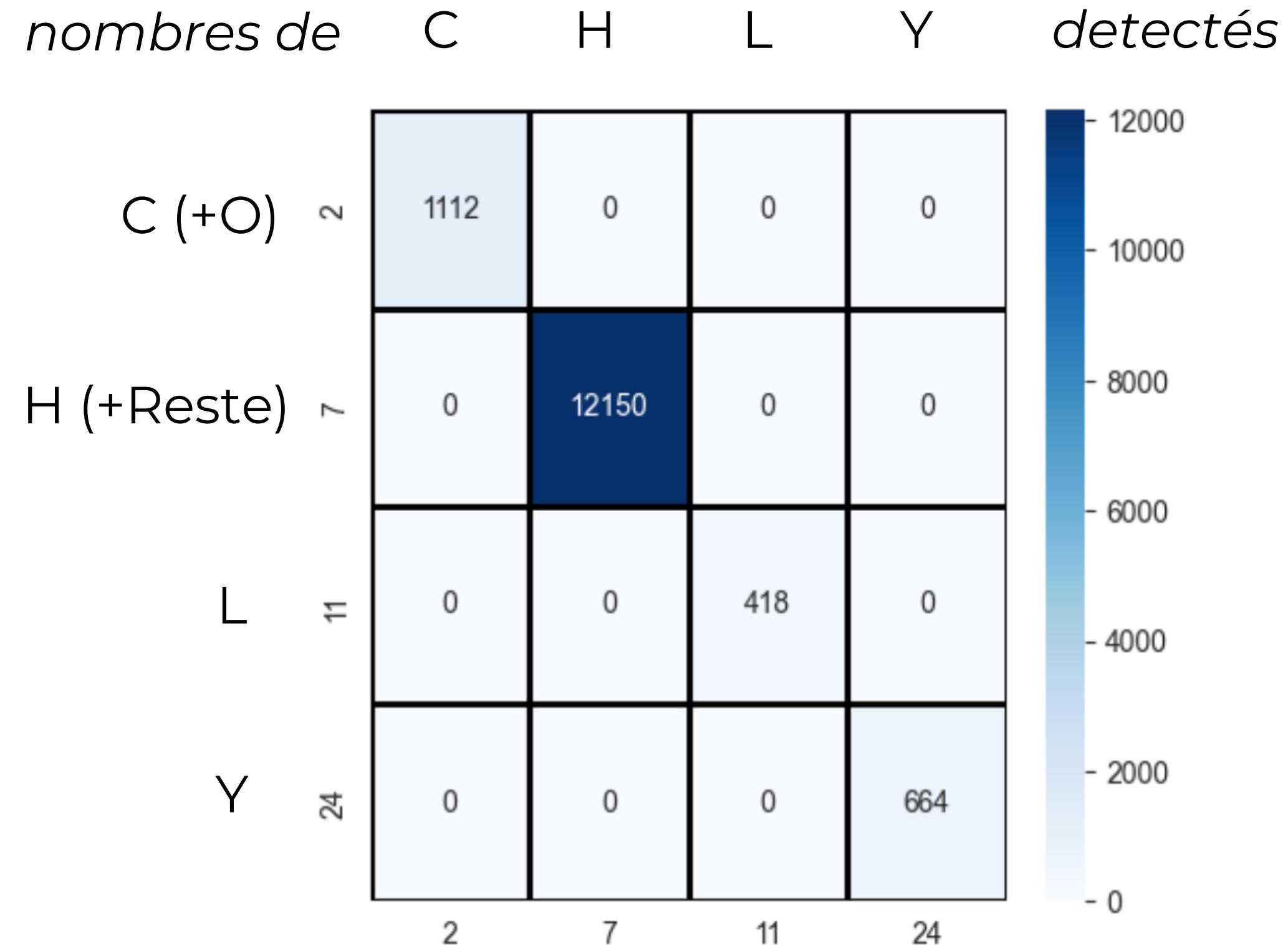


label : G

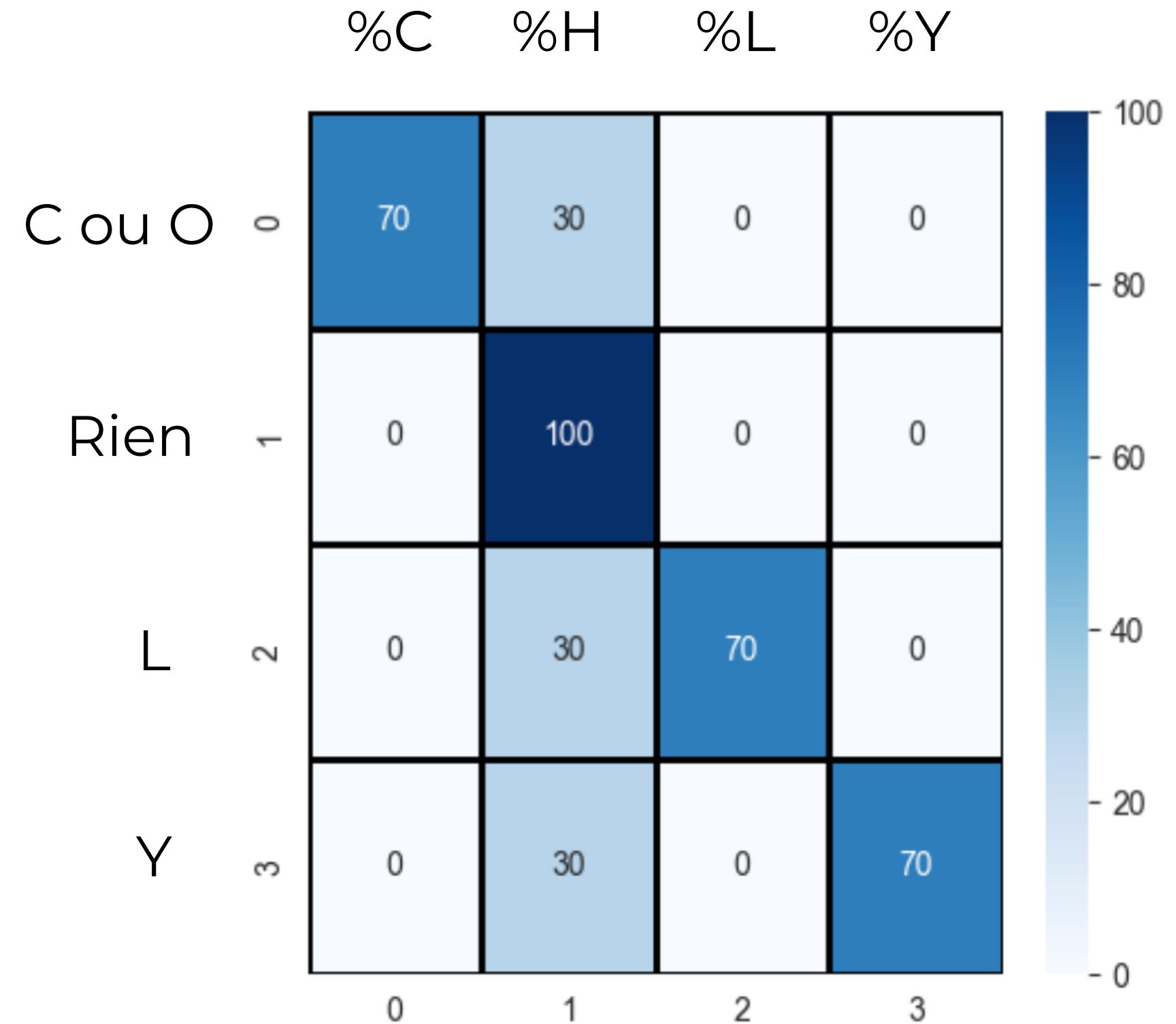
RÉSULTATS D'ENTRAINEMENT 2 (~8 MIN)



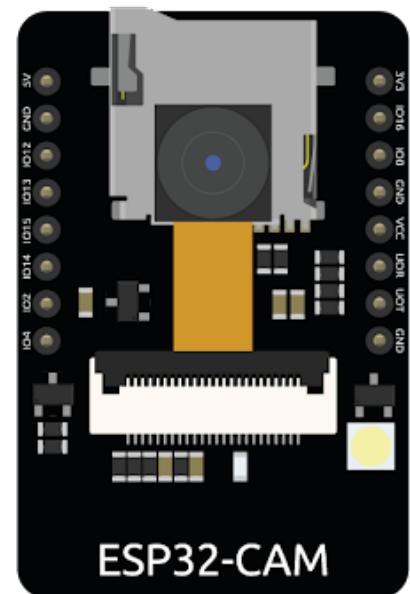
RÉSULTATS DE TEST 2



RÉSULTATS FINAUX SUR ESP32



ANALYSE DE LA PERTE DE PRÉCISION

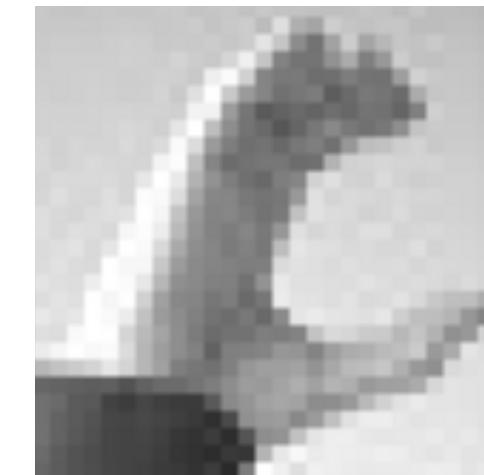


Camera +
ESP32 SDK



Perte 2 :
Qualité de la caméra
et du formatage

ESP32 SDK



Perte 3 :
Qualité du
redimensionnement



Perte 1 :
Qualité des données
d'entraînement

