FOOD CLASSIFICATION AND CALORIES ESTIMATION

MASTERING YOUR DIET

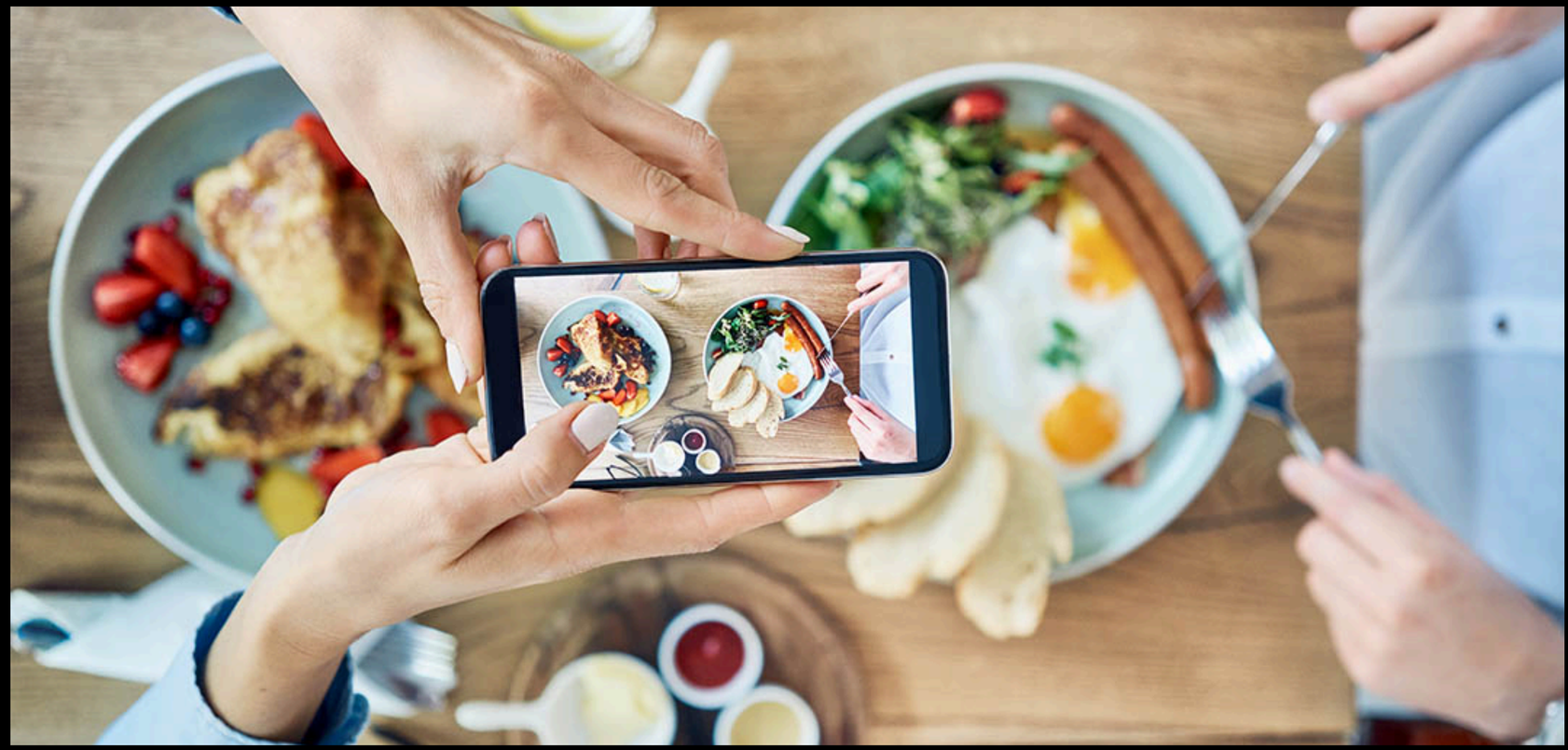
PRESENTER PAR : SALAHEDDINE KAYOUH
YASSINE CHMIRROU
ZAKARIA TIBTIBA

ENCADRER PAR: MR ANAS BELCAID

PLANNING

1. INTRODUCTION & RAPPEL
2. PRESENTATION DATASET FINAL
3. APPROCHE D’ETUDES & CREATION DES MODELS
4. CHALLENGES
5. APPLICATION WEB

INTRODUCTION ET RAPPEL

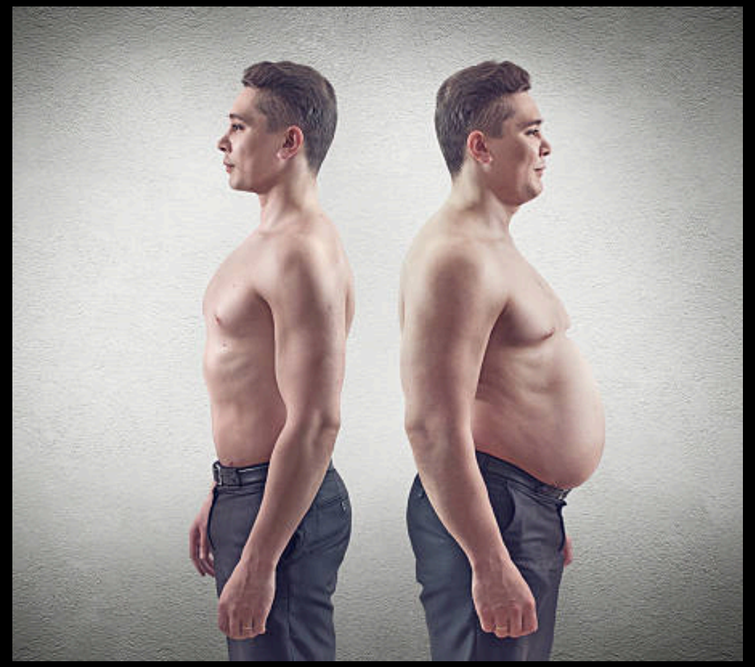


POURQUOI LE FOOD CLASSIFICATION ET CALORIES ESTIMATION ?

INTRODUCTION ET RAPPEL

?

RISING OBESITY AND CHRONIC DISEASES



INCREASING DEMAND FOR PERSONALIZED NUTRITION

BUSY AND STRESSFUL LIFESTYLES

ENVIRONMENTAL SUSTAINABILITY AND DIVERSE FOOD CULTURES

PRESENTATION DATASET FINAL (ECUSTFD)

ECUST FOOD DATASET (ECUSTFD)

- NUMBER OF CLASSES: 19
- NUMBER OF IMAGES: 2,978

FOR EACH IMAGES, OUR DATASET STILL PROVIDE OTHER INFORMATIONS AS FOLLOWS: ANNOTATION,MASS,VOLUME,DENSITY AND ENERGY

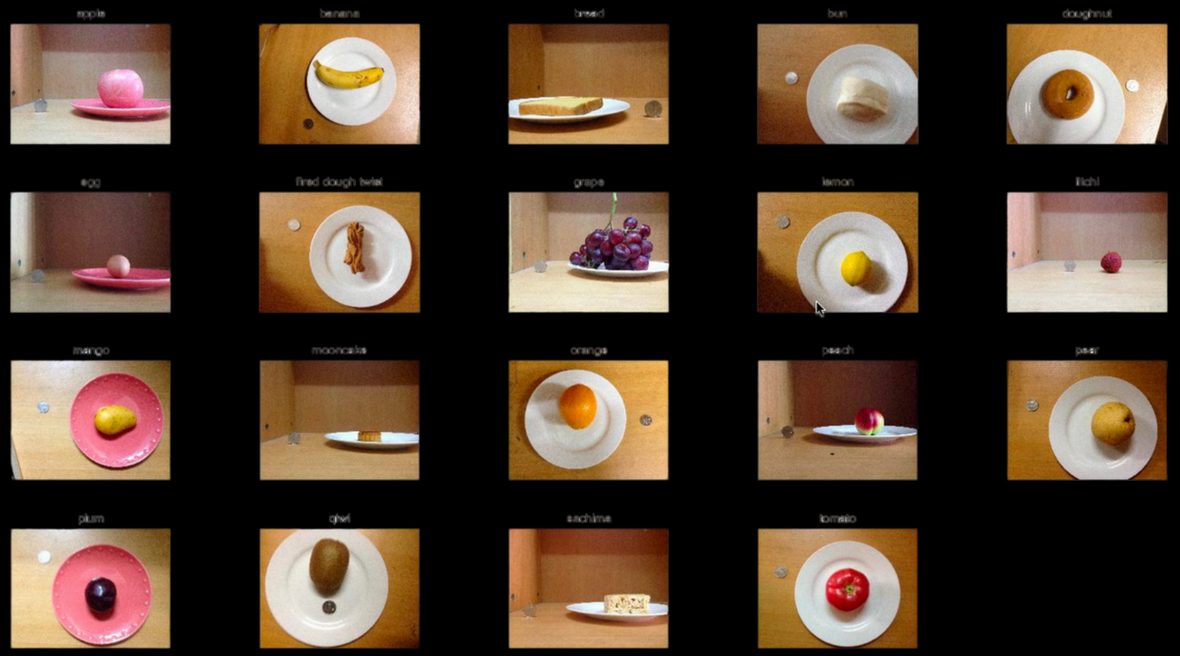
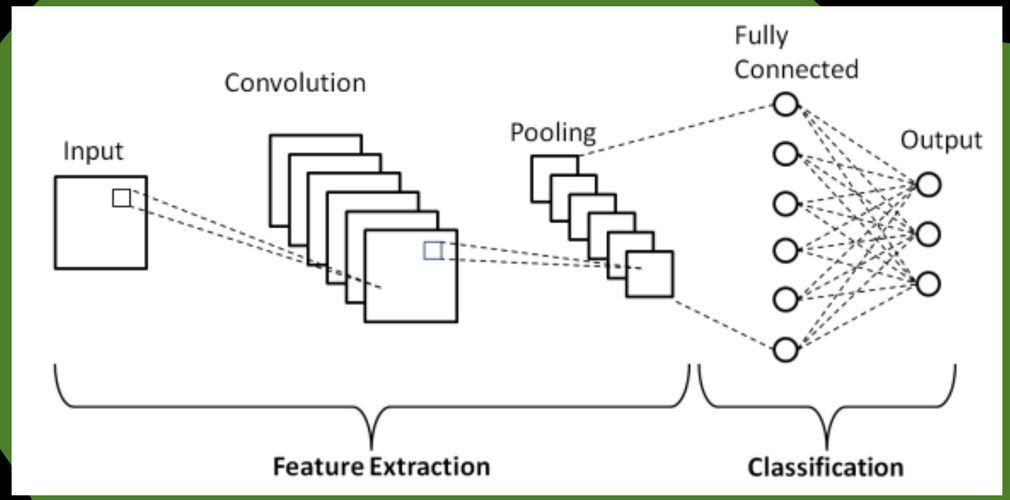


Table 1. ECUSTFD

Food Type	The number of images	The number of objects	Density (g/cm^3)	Energy $(kcal/g)$
apple	296	19	0.78	0.52
banana	178	15	0.91	0.89
bread	66	7	0.18	3.15
bun	90	8	0.34	2.23
doughnut	210	9	0.31	4.34
egg	104	7	1.03	1.43
fired dough twist	124	7	0.58	24.16
grape	58	2	0.97	0.69
lemon	148	4	0.96	0.29
litchi	78	5	1.00	0.66
mango	220	10	1.07	0.60
mix	108	14	/	/
mooncake	134	6	0.96	18.83
orange	254	15	0.90	0.63
peach	126	5	0.96	0.57
pear	166	6	1.02	0.39
plum	176	4	1.01	0.46
qiwi	120	8	0.97	0.61
sachima	150	5	0.22	21.45
tomato	172	4	0.98	0.27

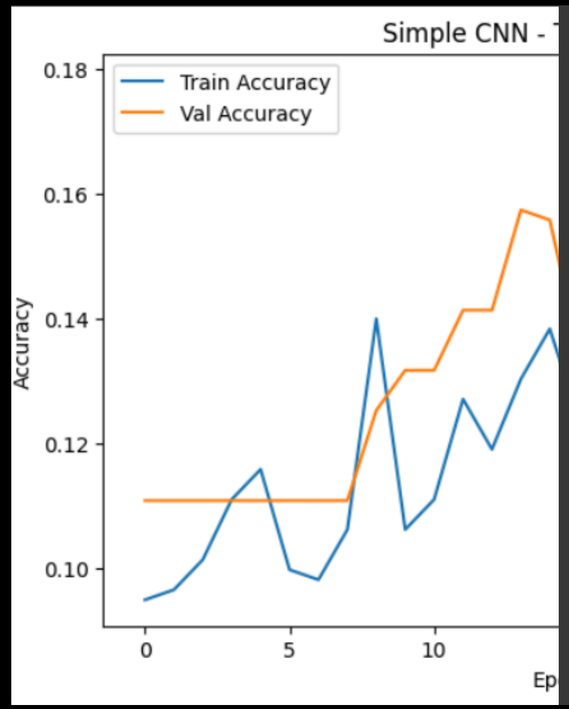
APPROCHE D'ÉTUDES & CRÉATION DES MODÈLES

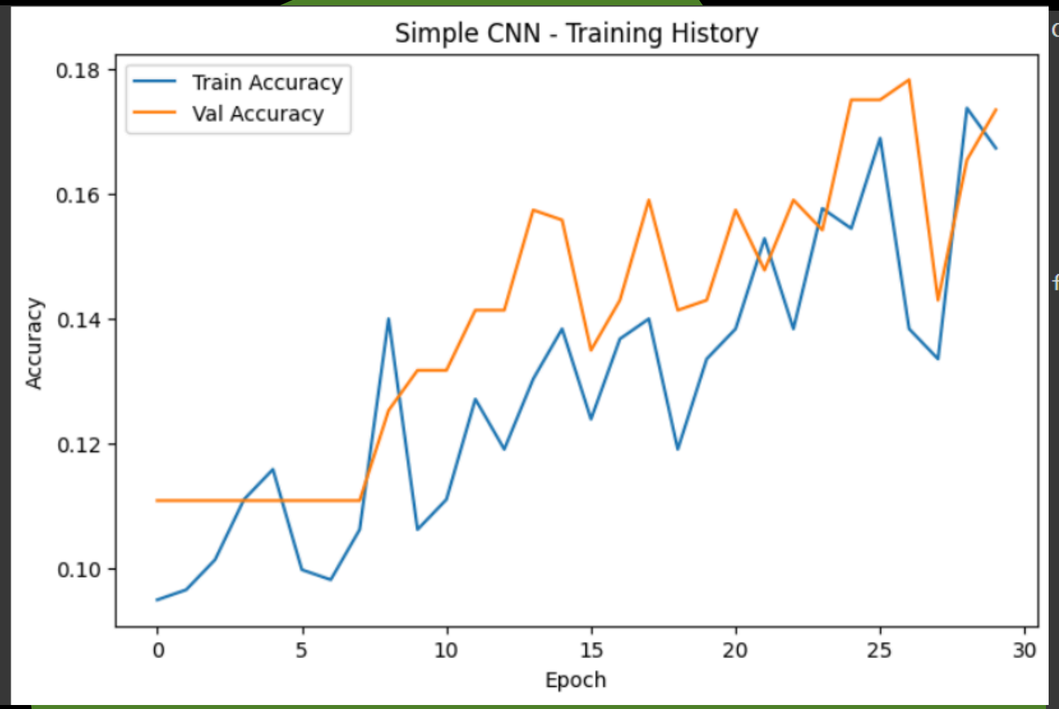
CNN FROM SCRATCH BASIQUE




```python
def create_simple_cnn_model(input_shape=(224, 224, 3), num_classes=19):
    """Crée un simple modèle CNN avec quelques couches pour la classification."""
    inputs = layers.Input(shape=input_shape)

    # Convolution + Pooling Block 1
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.MaxPooling2D((2, 2))(x)

    # Convolution + Pooling Block 2
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)

    # Global Average Pooling
    x = layers.GlobalAveragePooling2D()(x)

    # Fully Connected Layer
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    # Output Layer
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs, name='simple_cnn')
    return model
```

CNN FROM SCRATCH BASIQUE



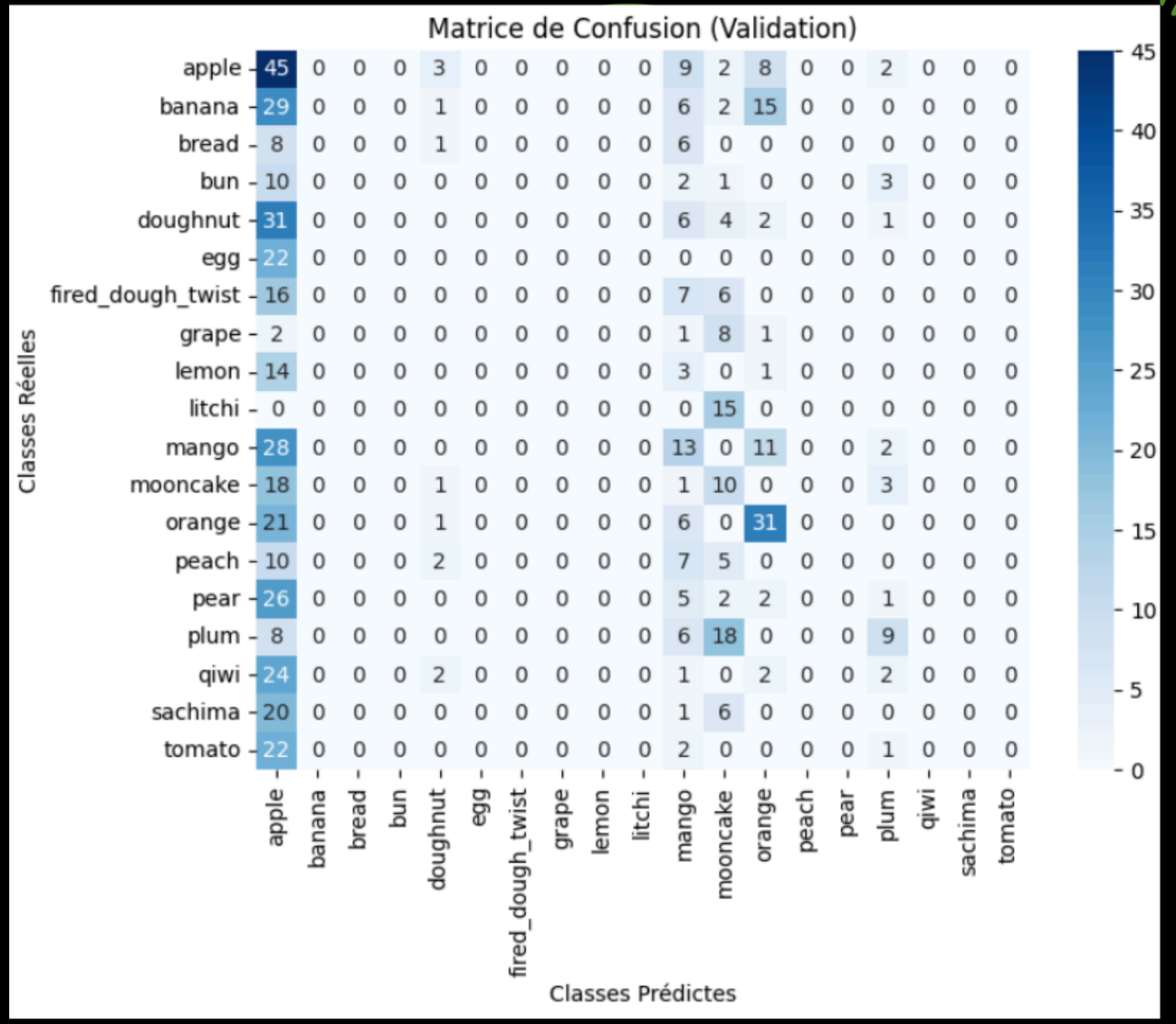


Classification Report (Precision, Recall, F1) :

	precision	recall	f1-score	support
apple	0.13	0.65	0.21	69
banana	0.00	0.00	0.00	53
bread	0.00	0.00	0.00	15
bun	0.00	0.00	0.00	16
doughnut	0.00	0.00	0.00	44
egg	0.00	0.00	0.00	22
fired_dough_twist	0.00	0.00	0.00	29
grape	0.00	0.00	0.00	12
lemon	0.00	0.00	0.00	18
litchi	0.00	0.00	0.00	15
mango	0.16	0.24	0.19	54
mooncake	0.13	0.30	0.18	33
orange	0.42	0.53	0.47	59
peach	0.00	0.00	0.00	24
pear	0.00	0.00	0.00	36
plum	0.38	0.22	0.28	41
qiwi	0.00	0.00	0.00	31
sachima	0.00	0.00	0.00	27
tomato	0.00	0.00	0.00	25
accuracy			0.17	623
macro avg	0.06	0.10	0.07	623
weighted avg	0.10	0.17	0.11	623

ACCURACY DE 17% DANS LA VALIDATION

CNN FROM SCRATCH BASIQUE



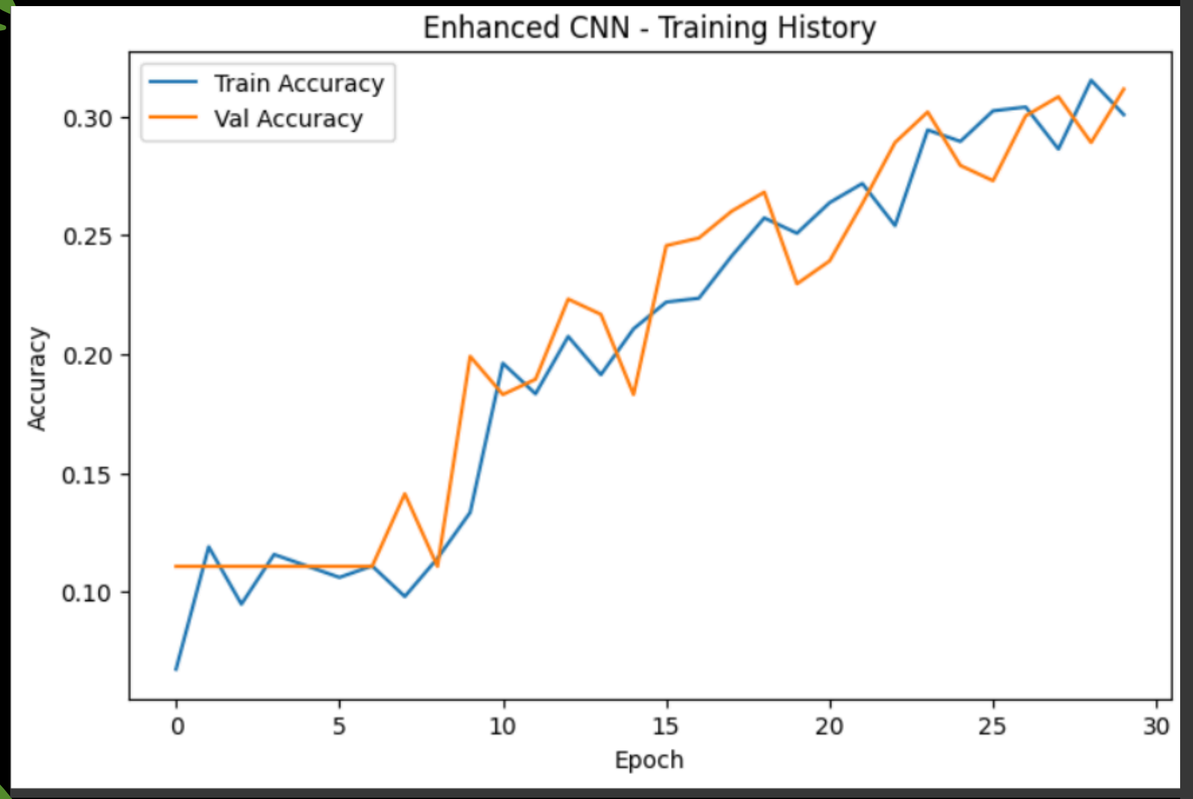
MODEL FROM SCRATCH AUGMENTER

```python
def create_enhanced_cnn_model(input_shape=(224, 224, 3), num_classes=19):
    """
    Un CNN légèrement plus profond que le 'simple_cnn_model'.
    - 3 blocs de convolution + pooling
    - Nombre de filtres en augmentation
    - Puis GlobalAveragePooling2D + 1 Dense cachée
    """
    inputs = layers.Input(shape=input_shape)

    # Bloc 1
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)

    # Bloc 2
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)
```

+1 BLOC DE CONVOLUTION
+1 COUCHE DE CONVOLUTION DANS CHAQUE BLOCS
(6 CONVOLUTIONS (2 PAR BLOC × 3 BLOCS).
DENSITER AUGMENTER :32 → 32, 64 → 64, 128 → 128
(DOUBLEZ À CHAQUE BLOC ET ALLEZ PLUS LOIN JUSQU'À 128)
DENSE(256), DONC UN RÉSEAU PLUS EXPRESSIF AU NIVEAU DE LA COUCHE FINALE

MODEL FROM SCRATCH AUGMENTER

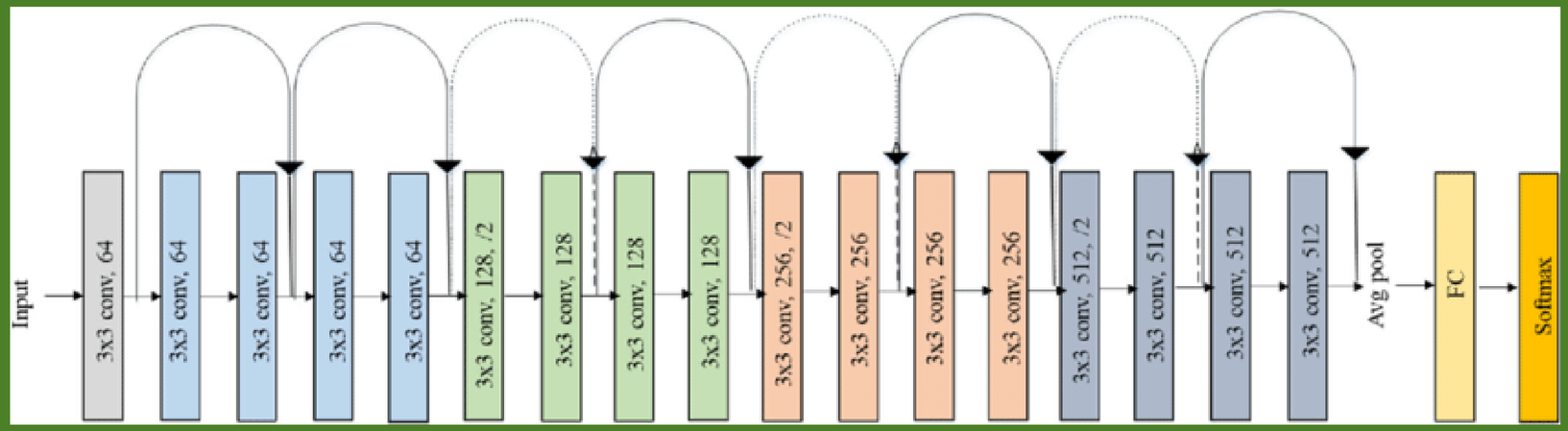
AUGMENTATION DE +17% DANS L'ACCURACY
DE VALIDATION DE 17% À 31%



Classification Report (Precision, Recall, F1) :

	precision	recall	f1-score	support
apple	0.31	0.94	0.47	69
banana	0.29	0.81	0.43	53
bread	0.00	0.00	0.00	15
bun	0.00	0.00	0.00	16
doughnut	0.19	0.09	0.12	44
egg	0.00	0.00	0.00	22
fired_dough_twist	0.00	0.00	0.00	29
grape	0.00	0.00	0.00	12
lemon	0.00	0.00	0.00	18
litchi	0.00	0.00	0.00	15
mango	0.24	0.17	0.20	54
mooncake	0.09	0.24	0.13	33
orange	0.89	0.81	0.85	59
peach	0.00	0.00	0.00	24
pear	0.00	0.00	0.00	36
plum	0.23	0.37	0.28	41
qiwi	0.00	0.00	0.00	31
sachima	0.00	0.00	0.00	27
tomato	0.00	0.00	0.00	25
accuracy			0.31	623
macro avg	0.12	0.18	0.13	623
weighted avg	0.20	0.31	0.22	623

RESNET18



1.CONVOLUTION LAYER : 1 COUCHE

2.BATCH NORMALIZATION : APRÈS CHAQUE CONVOLUTION POUR NORMALISER LES ACTIVATIONS.

3.RELU ACTIVATION : APRÈS CHAQUE BATCHNORM

4.RESIDUAL BLOCKS: 8 BLOCS AU TOTAL, REGROUPÉS EN 4 GROUPES

5.POOLING LAYERS: 2 COUCHES (MAXPOOLING ET AVERAGEPOOLING)

6.FULLY CONNECTED LAYER: POUR PRODUIRE LA SORTIE FINALE (CLASSIFICATION)

RESNET18

```
Epoch [1/10], Loss: 0.8999
Epoch [2/10], Loss: 0.1134
Epoch [3/10], Loss: 0.1129
Epoch [4/10], Loss: 0.0498
Epoch [5/10], Loss: 0.0250
Epoch [6/10], Loss: 0.0717
Epoch [7/10], Loss: 0.1490
Epoch [8/10], Loss: 0.0839
Epoch [9/10], Loss: 0.0930
Epoch [10/10], Loss: 0.1221
Accuracy: 57.79%
Precision: 0.67
Recall: 0.58
F1-Score: 0.54
```

PERFORMANCES GLOBALE

Classification Report:

	precision	recall	f1-score	support
Apple	1.00	0.18	0.31	166
Banana	0.92	0.62	0.74	94
Bread	0.00	0.00	0.00	36
Bun	0.85	0.29	0.44	58
Doughnut	0.59	0.70	0.64	122
Egg	0.81	0.50	0.62	60
Grape	0.00	0.00	0.00	34
Lemon	0.90	0.89	0.90	112
Litchi	0.97	0.73	0.83	48
Mango	0.63	0.72	0.67	118
Mooncake	0.34	1.00	0.51	68
Orange	0.47	0.80	0.59	140
Peach	0.53	0.12	0.19	78
Pear	0.34	0.31	0.32	94
Plum	0.64	1.00	0.78	94
Sachima	0.47	1.00	0.64	96
Tomato	0.95	0.34	0.50	122
accuracy			0.58	1540
macro avg	0.61	0.54	0.51	1540
weighted avg	0.67	0.58	0.54	1540

Confusion Matrix:

```
[[ 30   0   0   0  26   4   0   0   0   1   8  29   8  38   3  19   0]
 [  0  58   0   0   6   0   0   0   0   2   8   0   0   0   0  20   0]
 [  0   4   0   0   0   0   0   0   0   0  18   0   0   0   0  14   0]
 [  0   0  17  17   0   0   0   0   0   0   0   0   0   4   8  12   0]
 [  0   0   0   0  86   0   0   0   0   0  24   3   0   0   0   9   0]
 [  0   0   0   3   0  30   0   0   0   9   4   8   0   0   0   6   0]
 [  0   0   0   0   5   0   0   0   1   0   0   0   0   0  28   0   0]
 [  0   0   0   0   0   0   0 100   0  10   2   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  35   0   2   0   0   0  10   1   0]
 [  0   0   0   0   0   0   0  10   0  85  23   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  68   0   0   0   0   0   0]
 [  0   0   0   0  10   0   0   1   0   3  12 112   0   0   2   0   0]
 [  0   0   0   0   1   3   0   0   0   5  26   6   9  15   0  11   2]
 [  0   1   8   0  12   0   0   0   0  20   6   0   0  29   1  17   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  94   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  96   0]
 [  0   0   0   0   0   0   0   0   0   0   0  80   0   0   0   1  41]]
```

MODEL FROM SCRATCH AUGMENTER V2 (ALEXNET)

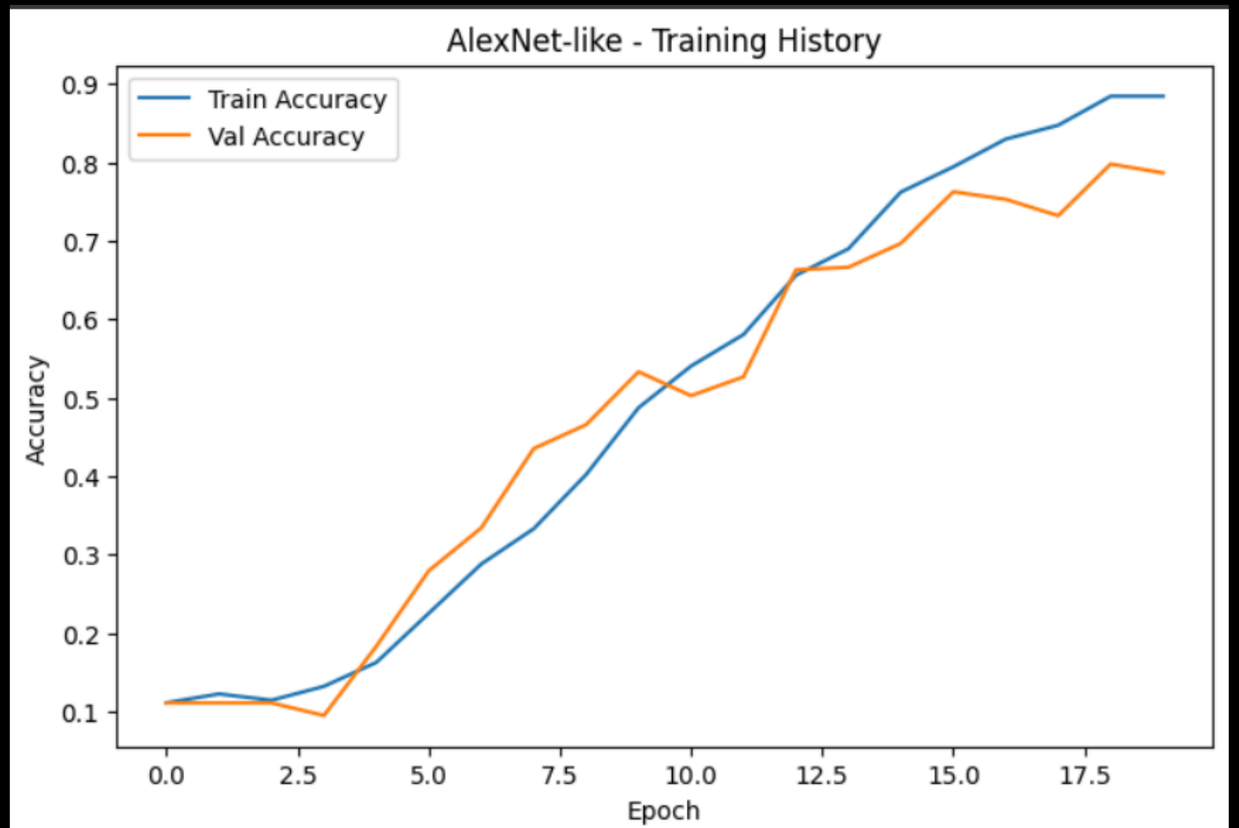
+3 BLOC DE CONVOLUTION DE 3 À 5
3 GROS BLOCS DE CONVOLUTION), AVEC DES COUCHES FULLY-CONNECTED (DENSES) BEAUCOUP PLUS PROFONDES (2 * 4096 NEURONES).
DENSITER AUGMENTER DE BASE : 96 → 256 → 384/384 → 256 CAPTURANT PLUS DE DETAILLES (DOUBLEZ À CHAQUE BLOC ET ALLEZ PLUS LOIN JUSQU'À 128)
2 DENSE(4096), CHACUNE SUIVIE D'UN DROPOUT(0.5).

```python
def create_alexnet_model(input_shape=(224, 224, 3), num_classes=19):
    """
    Implémentation inspirée d'AlexNet pour 19 classes.
    Architecture de base (simplifiée & adaptée) :
      1) Conv(96 filtres, kernel=11, stride=4) + ReLU + MaxPool
      2) Conv(256 filtres, kernel=5) + ReLU + MaxPool
      3) Conv(384 filtres, kernel=3) + ReLU
      4) Conv(384 filtres, kernel=3) + ReLU
      5) Conv(256 filtres, kernel=3) + ReLU + MaxPool
      6) Flatten -> Dense(4096) -> ReLU -> Dropout
      7) Dense(4096) -> ReLU -> Dropout
      8) Dense(num_classes) -> Softmax
    """
    inputs = layers.Input(shape=input_shape)

    # 1) Premier bloc
    x = layers.Conv2D(96, kernel_size=11, strides=4, padding='same', activation='relu')(in
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=2)(x)

    # 2) Deuxième bloc
    x = layers.Conv2D(256, kernel_size=5, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=2)(x)
```

```python
    inputs = layers.Input(shape=input_shape)

    # 1) Premier bloc
    x = layers.Conv2D(96, kernel_size=11, strides=4, padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=2)(x)

    # 2) Deuxième bloc
    x = layers.Conv2D(256, kernel_size=5, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=2)(x)

    # 3) Troisième bloc de convolutions
    x = layers.Conv2D(384, kernel_size=3, padding='same', activation='relu')(x)
    x = layers.Conv2D(384, kernel_size=3, padding='same', activation='relu')(x)
    x = layers.Conv2D(256, kernel_size=3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=2)(x)

    # 4) Passage en Dense
    x = layers.Flatten()(x)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    # 5) Output
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs, outputs, name='alexnet_like')
    return model
```

MODEL FROM SCRATCH AUGMENTER V2 (ALEXNET)

AUGMENTATION DE +49% DANS L'ACCURACY
DE VALIDATION DE 31% À 80%



Classification Report (Precision, Recall, F1) :

	precision	recall	f1-score	support
apple	0.86	0.81	0.84	69
banana	0.88	0.81	0.84	53
bread	0.80	0.80	0.80	15
bun	0.84	1.00	0.91	16
doughnut	0.86	0.86	0.86	44
egg	0.61	1.00	0.76	22
fired_dough_twist	0.86	0.66	0.75	29
grape	1.00	1.00	1.00	12
lemon	0.88	0.78	0.82	18
litchi	0.94	1.00	0.97	15
mango	0.68	0.81	0.74	54
mooncake	0.66	0.94	0.78	33
orange	0.92	0.83	0.88	59
peach	0.58	0.46	0.51	24
pear	0.82	0.78	0.80	36
plum	0.98	0.98	0.98	41
qiwi	0.68	0.68	0.68	31
sachima	0.55	0.59	0.57	27
tomato	1.00	0.40	0.57	25
accuracy			0.80	623
macro avg	0.81	0.80	0.79	623
weighted avg	0.81	0.80	0.80	623

Data Augmentation

Augmenter la taille et la diversité des données pour améliorer la généralisation du modèle CNN.

Pipeline d'Augmentation

Utilisation d'Albumentations pour une approche modulaire et efficace.

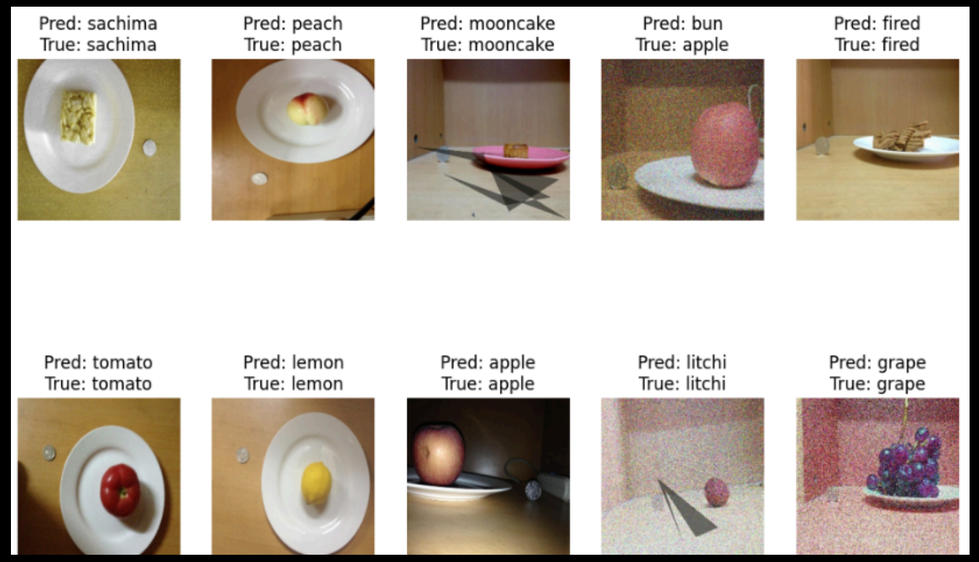
```python
def create_augmentation_pipeline():
    """Create an augmentation pipeline with various transforms."""
    return A.Compose([
        A.OneOf([
            A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
            A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=0.5)
        ], p=0.8),
        A.OneOf([
            A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
            A.ISONoise(color_shift=(0.01, 0.05), intensity=(0.1, 0.5), p=0.5),
        ], p=0.5),
        A.OneOf([
            A.RandomRotate90(p=0.5),
            A.Rotate(limit=15, p=0.5),
        ], p=0.5),
        A.OneOf([
            A.MotionBlur(blur_limit=3, p=0.5),
            A.MedianBlur(blur_limit=3, p=0.5),
            A.GaussianBlur(blur_limit=3, p=0.5),
        ], p=0.3),
        A.OneOf([
            A.OpticalDistortion(distort_limit=0.05, shift_limit=0.05, p=0.5),
            A.GridDistortion(distort_limit=0.1, p=0.5),
        ], p=0.3),
        A.OneOf([
            A.RandomShadow(p=0.5)
```

Impact sur le Dataset

Amélioration de la robustesse du modèle face à des variations réelles.

MODÈLE CNN POUR LA CLASSIFICATION AVEC DATA AUGMENTES

```
Epoch 1/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 114s 164ms/step - accuracy: 0.0816 - auc: 0.5611 - loss: 3.6877 - top_3_accuracy: 0.2011 - val_accuracy: 0.1012 - val_auc: 0.6157 - val_loss: 3.2348 - val_top_3_accuracy: 0.2182
Epoch 2/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 79s 139ms/step - accuracy: 0.1851 - auc: 0.7176 - loss: 3.0141 - top_3_accuracy: 0.3749 - val_accuracy: 0.1347 - val_auc: 0.7121 - val_loss: 3.1624 - val_top_3_accuracy: 0.3148
Epoch 3/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 79s 139ms/step - accuracy: 0.2683 - auc: 0.8140 - loss: 2.7205 - top_3_accuracy: 0.5061 - val_accuracy: 0.1600 - val_auc: 0.7432 - val_loss: 3.1102 - val_top_3_accuracy: 0.3617
Epoch 4/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 78s 137ms/step - accuracy: 0.3533 - auc: 0.8709 - loss: 2.4705 - top_3_accuracy: 0.6210 - val_accuracy: 0.1940 - val_auc: 0.7784 - val_loss: 2.9419 - val_top_3_accuracy: 0.4217
Epoch 5/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 78s 137ms/step - accuracy: 0.4351 - auc: 0.9055 - loss: 2.2663 - top_3_accuracy: 0.7024 - val_accuracy: 0.1489 - val_auc: 0.7532 - val_loss: 3.4372 - val_top_3_accuracy: 0.3884
Epoch 6/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 136ms/step - accuracy: 0.5068 - auc: 0.9284 - loss: 2.0960 - top_3_accuracy: 0.7654 - val_accuracy: 0.1876 - val_auc: 0.8024 - val_loss: 3.0756 - val_top_3_accuracy: 0.5045
Epoch 7/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 78s 137ms/step - accuracy: 0.5776 - auc: 0.9476 - loss: 1.9263 - top_3_accuracy: 0.8232 - val_accuracy: 0.4441 - val_auc: 0.9191 - val_loss: 2.1724 - val_top_3_accuracy: 0.7533
Epoch 8/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 136ms/step - accuracy: 0.6417 - auc: 0.9591 - loss: 1.8026 - top_3_accuracy: 0.8600 - val_accuracy: 0.2166 - val_auc: 0.7997 - val_loss: 3.2529 - val_top_3_accuracy: 0.5505
Epoch 9/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 136ms/step - accuracy: 0.6725 - auc: 0.9661 - loss: 1.7161 - top_3_accuracy: 0.8852 - val_accuracy: 0.2972 - val_auc: 0.8617 - val_loss: 2.6673 - val_top_3_accuracy: 0.6263
Epoch 10/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 137ms/step - accuracy: 0.7223 - auc: 0.9749 - loss: 1.6072 - top_3_accuracy: 0.9082 - val_accuracy: 0.3520 - val_auc: 0.8919 - val_loss: 2.4235 - val_top_3_accuracy: 0.6666
Epoch 11/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 136ms/step - accuracy: 0.7365 - auc: 0.9757 - loss: 1.5824 - top_3_accuracy: 0.9075 - val_accuracy: 0.2816 - val_auc: 0.8390 - val_loss: 2.8473 - val_top_3_accuracy: 0.5747
Epoch 12/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 77s 136ms/step - accuracy: 0.7713 - auc: 0.9812 - loss: 1.5002 - top_3_accuracy: 0.9275 - val_accuracy: 0.3963 - val_auc: 0.9037 - val_loss: 2.3647 - val_top_3_accuracy: 0.7490
Epoch 13/50
...
Epoch 49/50
553/553 ━━━━━━━━━━━━━━━━━━━━ 83s 138ms/step - accuracy: 0.9428 - auc: 0.9974 - loss: 1.0469 - top_3_accuracy: 0.9872 - val_accuracy: 0.8493 - val_auc: 0.9923 - val_loss: 1.2039 - val_top_3_accuracy: 0.9733
Epoch 50/50
```



	precision	recall	f1-score	support
apple	0.94	0.83	0.88	259
banana	0.91	0.90	0.90	236
bread	1.00	0.89	0.94	213
bun	0.41	1.00	0.58	218
doughnut	0.95	0.83	0.89	242
egg	0.98	0.81	0.89	221
fired	0.99	0.92	0.96	225
grape	1.00	0.98	0.99	212
lemon	0.98	0.90	0.94	230
litchi	0.92	1.00	0.96	216
mango	0.84	0.92	0.88	244
mix	1.00	0.09	0.17	43
mooncake	0.99	0.82	0.90	227
orange	0.87	0.97	0.92	251
peach	0.94	0.81	0.87	225
pear	0.93	0.82	0.87	233
plum	0.99	0.90	0.94	235
qiwi	0.88	0.73	0.80	224
sachima	0.99	0.90	0.94	230
tomato	0.99	0.94	0.97	234
accuracy			0.88	4418
macro avg	0.92	0.85	0.86	4418
weighted avg	0.92	0.88	0.89	4418

Comparaison

CNN From Scratch Basic

- Accuracy : 17%
- Original Images

ResNet18

- Accuracy : 58%
- Original Images
- Pre-trained

Model From Scratch Augmenter v2 (Alexnet)

- Accuracy : 80%
- Original Images

Modèle CNN avec données augmentées

- Accuracy : 88%
- Original Images + Augmented Images

PIPELINE D’ESTIMATION

PIPELINE D'ESTIMATION DE CALORIE

CHOIX D'APPROCHE ?

DETECTRON2

MODELE OBJECT RECOGNITION ET SEGMENTATION

PAR LOI PROBABILISTIQUE

PIPELINE D'ESTIMATION DE CALORIE

CHOIX D'APPROCHE: DETECTRON2

INCONVENIENT:

- INSTALLATION COMPLEXE
- EXIGEANT
- COURBE D'APPRENTISSAGE

AVANTAGE:

- PRÉCIS
- MULTI-CLASSES
- PERSONNALISABLE

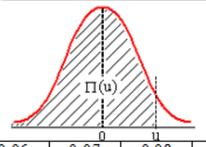
PIPELINE D'ESTIMATION DE CALORIE

CHOIX D'APPROCHE: LOI NORMAL

INCONVENIENT:

- SENSIBILITÉ AUX VALEURS EXTRÊMES
- NON-ADAPTÉE AUX DISTRIBUTIONS ASYMÉTRIQUES
- HYPOTHÈSE PARFOIS IRRÉALISTE

Table de Loi Normale
P(x<u)



	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0,0	0,5000	0,5040	0,5080	0,5120	0,5160	0,5199	0,5239	0,5279	0,5319	0,5359
0,1	0,5398	0,5438	0,5478	0,5517	0,5557	0,5596	0,5636	0,5675	0,5714	0,5753
0,2	0,5793	0,5832	0,5871	0,5910	0,5948	0,5987	0,6026	0,6064	0,6103	0,6141
0,3	0,6179	0,6217	0,6255	0,6293	0,6331	0,6368	0,6406	0,6443	0,6480	0,6517
0,4	0,6554	0,6591	0,6628	0,6664	0,6700	0,6736	0,6772	0,6808	0,6844	0,6879
0,5	0,6915	0,6950	0,6985	0,7019	0,7054	0,7088	0,7123	0,7157	0,7190	0,7224
0,6	0,7257	0,7291	0,7324	0,7357	0,7389	0,7422	0,7454	0,7486	0,7517	0,7549
0,7	0,7580	0,7611	0,7642	0,7673	0,7704	,0,7734	0,7764	0,7794	0,7823	0,7852
0,8	0,7881	0,7910	0,7939	0,7967	0,7995	0,8023	0,8051	0,8078	0,8106	0,8133
0,9	0,8159	0,8186	0,8212	0,8238	0,8254	0,8289	0,8315	0,8340	0,8365	0,8389
1,0	0,8413	0,8438	0,8461	0,8485	0,8508	0,8531	0,8554	0,8577	0,8599	0,8621
1,1	0,8643	0,8665	0,8686	0,8708	0,8729	0,8749	0,8770	0,8790	0,8810	0,8830
1,2	0,8849	0,8869	0,8888	0,8907	0,8925	0,8944	0,8962	0,8980	0,8997	0,9015
1,3	0,9032	0,9049	0,9066	0,9082	0,9099	0,9115	0,9131	0,9147	0,9162	0,9177
1,4	0,9192	0,9207	0,9222	0,9236	0,9251	0,9265	0,9279	0,9292	0,9306	0,9319
1,5	0,9332	0,9345	0,9357	0,9370	0,9382	0,9394	0,9406	0,9418	0,9429	0,9441
1,6	0,9452	0,9463	0,9474	0,9484	0,9495	0,9505	0,9515	0,9525	0,9535	0,9545
1,7	0,9554	0,9564	0,9573	,0,9582	0,9591	0,9599	0,9608	0,9616	0,9625	0,9633
1,8	0,9641	0,9649	0,9656	0,9664	0,9671	0,9678	0,9686	0,9693	0,9699	0,9706
1,9	0,9713	0,9719	0,9726	0,9732	0,9738	0,9744	0,9750	0,9756	6,9761	0,9767
2,0	0,9772	0,9778	0,9783	0,9788	0,9793	0,9798	0,9803	0,9808	0,9812	0,9817
2,1	0,9821	0,9826	0,9830	0,9834	0,9838	0,9842	0,9846	0,9850	0,9854	0,9857
2,2	0,9861	0,9864	0,9868	0,9871	0,9875	0,9878	0,9881	0,9884	0,9887	0,9890
2,3	0,9893	0,9896	0,9898	0,9901	0,9904	0,9906	0,9909	0,9911	0,9913	0,9916
2,4	0,9918	0,9920	0,9922	0,9925	0,9927	0,9929	0,9931	0,9932	0,9934	0,9936
2,5	0,9938	0,9940	0,9941	0,9943	0,9945	0,9946	0,9948	0,9949	0,9951	0,9952
2,6	0,9953	0,9955	0,9956	0,9957	0,9959	0,9960	0,9961	0,9962	0,9963	0,9964
2,7	0,9965	0,9966	0,9967	0,9968	0,9969	0,9970	0,9971	0,9972	0,9973	0,9974
2,8	0,9974	0,9975	0,9976	0,9977	0,9977	0,9978	0,9979	0,9979	0,9980	0,9981
2,9	0,9981	0,9982	0,9982	0,9983	0,9984	0,9984	0,9985	0,9985	0,9986	0,9986
3,0	0,9987	0,9987	0,9987	0,9988	0,9988	0,9989	0,9989	,0,9989	0,9990	0,9990
3,1	0,9990	0,9991	0,9991	0,9991	0,9992	0,9992	0,9992	0,9992	0,9993	0,9993
3,2	0,9993	0,9993	0,9994	0,9994	0,9994	0,9994	0,9994	0,9995	0,9995	0,9995
3,3	0,9995	0,9995	0,9995	0,9996	0,9996	0,9996	0,9996	0,9996	0,9996	0,9997
3,4	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9998

AVANTAGE:

- RÉPARTITION RÉALISTE
- FACILITÉ D'AJUSTEMENT
- APPLICABILITÉ UNIVERSELLE

PIPELINE D’ESTIMATION DE CALORIE

CHOIX D’APPROCHE

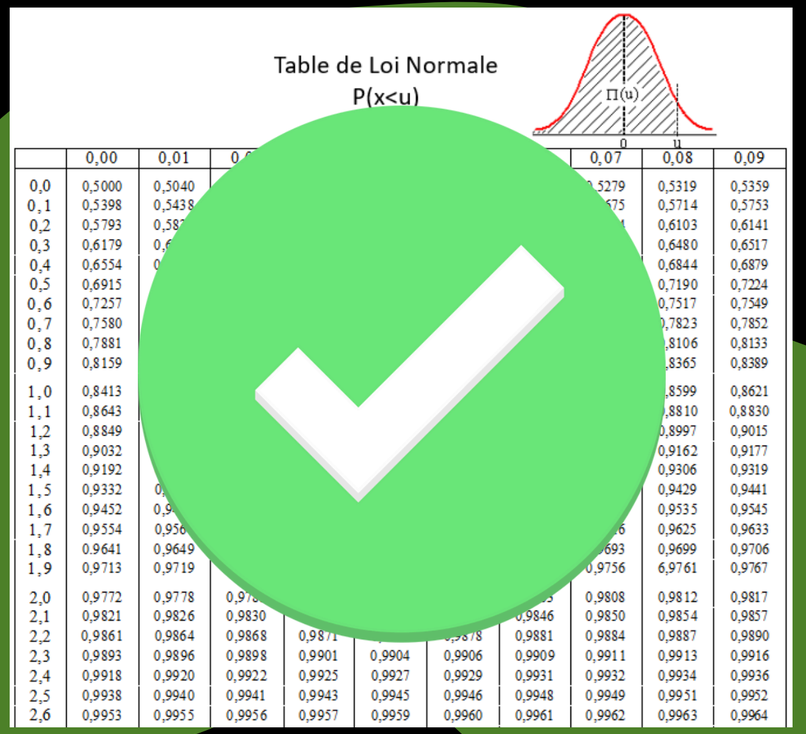
DETECTRON2

LIMITATION APRES CLONAGE DANS L’INSTALATION ET LE HARDWARE

PIPELINE D'ESTIMATION DE CALORIE

CHOIX D'APPROCHE

LOI NORMAL



	0,00	0,01	0,07	0,08	0,09
0,0	0,5000	0,5040	0,5279	0,5319	0,5359
0,1	0,5398	0,5438	0,5675	0,5714	0,5753
0,2	0,5793	0,583	0,6103	0,6141	
0,3	0,6179	0,6	0,6480	0,6517	
0,4	0,6554	0	0,6844	0,6879	
0,5	0,6915		0,7190	0,7224	
0,6	0,7257		0,7517	0,7549	
0,7	0,7580		0,7823	0,7852	
0,8	0,7881		0,8106	0,8133	
0,9	0,8159		0,8365	0,8389	
1,0	0,8413		0,8599	0,8621	
1,1	0,8643		0,8810	0,8830	
1,2	0,8849		0,8997	0,9015	
1,3	0,9032		0,9162	0,9177	
1,4	0,9192		0,9306	0,9319	
1,5	0,9332	0	0,9429	0,9441	
1,6	0,9452	0,9	0,9535	0,9545	
1,7	0,9554	0,956	0,9625	0,9633	
1,8	0,9641	0,9649	0,9699	0,9706	
1,9	0,9713	0,9719	0,9756	6,9761	0,9767
2,0	0,9772	0,9778	0,9808	0,9812	0,9817
2,1	0,9821	0,9826	0,9830	0,9846	0,9850	0,9854	0,9857
2,2	0,9861	0,9864	0,9868	0,9871	0,9881	0,9884	0,9887	0,9890
2,3	0,9893	0,9896	0,9898	0,9901	0,9904	0,9906	0,9909	0,9911	0,9913	0,9916
2,4	0,9918	0,9920	0,9922	0,9925	0,9927	0,9929	0,9931	0,9932	0,9934	0,9936
2,5	0,9938	0,9940	0,9941	0,9943	0,9945	0,9946	0,9948	0,9949	0,9951	0,9952
2,6	0,9953	0,9955	0,9956	0,9957	0,9959	0,9960	0,9961	0,9962	0,9963	0,9964

CRÉATION D'UN DICTIONNAIRE POUR CHAQUE CLASSE, AVEC DES ÉCHANTILLONS DE VOLUMES ET DE POIDS PROCHES DES OBJETS ENTRAÎNÉS

PIPELINE D'ESTIMATION DE CALORIE

LOI NORMAL

$$\text{Calories Estimées} = \mu + \epsilon,$$
$$\text{où } \epsilon \sim \mathcal{N}(0, \sigma^2).$$

1	id	type	volume(mm^3)	weight(g)	calorie (cl)
2	orange001	orange	160	220.5	103.359375
3	orange002	orange	280	197.9	92.765625
4	orange003	orange	210	202.5	94.921875
5	orange004	orange	260	218	102.1875
6	orange005	orange	150	138.9	65.109375
7	orange006	orange	200	179.3	84.046875
8	orange007	orange	220	184.8	86.625
9	orange008	orange	270	232.5	108.984375
10	orange009	orange	220	209.5	98.203125
11	orange010	orange	260	234	109.6875
12	orange011	orange	250	222	104.0625
13	orange012	orange	200	177.3	83.109375
14	orange013	orange	190	162.3	76.078125
15	orange014	orange	160	167.5	78.515625
16	orange015	orange	130	179.7	84.234375

- CRÉATION DU DICTIONNAIRE DES CLASSES
- COLLECTE DES DONNÉES DE CALORIES
- APPLICATION DE LA LOI NORMALE
- INTÉGRATION DES VARIATIONS ALÉATOIRES

DÉFIS

PROBLÈME DE SURAPPRENTISSAGE (OVERFITTING)

Lors de l'entraînement des modèles sur des ensembles de données relativement petits, le modèle a tendance à mémoriser les exemples d'entraînement au lieu de généraliser. Cela se manifeste par une faible précision sur les données de test malgré une haute performance sur les données d'entraînement.

PROBLÈME DE CONVERGENCE LENTE

En particulier pour les modèles construits from scratch, la convergence a été lente en raison d'une mauvaise initialisation des poids ou de choix sous-optimaux des hyperparamètres (learning rate, batch size, etc.).

PROBLÈME DE DÉSÉQUILIBRE DES CLASSES

Certaines classes alimentaires étaient surreprésentées, ce qui a conduit à un biais du modèle favorisant ces classes. Cela a impacté les métriques comme la précision, le rappel et le F1-score(Resnet18)

Application Web

Déploiement avec Streamlit

FINAL THOUGHTS

IN THIS PROJECT, WE AIM TO BUILD A FOOD CLASSIFICATION AND CALORIE ESTIMATION MODEL USING COMPUTER VISION, DEEP LEARNING, AND MATHEMATICAL FORMULAS. BY INTEGRATING THESE TECHNOLOGIES, WE'LL CREATE A SYSTEM THAT SIMPLIFIES DIET TRACKING AND SUPPORTS HEALTHCARE, FITNESS, AND FOOD SERVICES.

THANK YOU FOR YOUR ATTENTION