

عشان نقدر نحمل الداتا سيت Kaggle تثبيت مكتبة

```
!pip install -q kaggle
```

```
# بيحتوي على (API Key) رفع ملف kaggle.json  
from google.colab import files  
files.upload()
```

Choose Files no files selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle (2).json to kaggle (2).json

```
# تغيير اسم الملف ليكون بالاسم الصحيح  
!mv "kaggle (2).json" kaggle.json
```

```
# Kaggle إنشاء فولدر مخفي خاص بـ  
!mkdir -p ~/.kaggle  
# إلى الفولدر API نسخ ملف الـ  
!cp kaggle.json ~/.kaggle/  
# إعطاء صلاحيات للملف (حماية وأمان)  
!chmod 600 ~/.kaggle/kaggle.json
```

```
# Kaggle من Dataset Food11 تحميل  
!kaggle datasets download -d vermaavi/food11  
# food11 فك الضغط داخل فولدر اسمه  
!unzip -q food11.zip -d food11
```

Dataset URL: <https://www.kaggle.com/datasets/vermaavi/food11>
License(s): other
Downloading food11.zip to /content
99% 2.15G/2.17G [00:21<00:00, 251MB/s]
100% 2.17G/2.17G [00:21<00:00, 108MB/s]

```
import os
import h5py

# دالة لعرض معلومات أساسية عن الداتا
def food11_summary(path):
    # قراءة كل الصور
    images = [f for f in os.listdir(path) if f.endswith(".jpg")]
    # استخراج الكلاسات من اسم الصورة (قبل _)
    classes = set(img.split("_")[0] for img in images)

    print("Dataset: Food11")
    print("Dataset Path:", path)
    print("Number of Classes:", len(classes))
    print("Total Images:", len(images))
    print("-"*40)

# تشغيل الدوال
food11_summary("/content/food11/training")
```

```
Dataset: Food11
Dataset Path: /content/food11/training
Number of Classes: 11
Total Images: 9866
-----
```

```

import matplotlib.pyplot as plt
from PIL import Image
import random
import os
# دالة لعرض صور عشوائية من الداتا
def show_samples_food11(path, n=5):
    # قراءة الصور
    images = [f for f in os.listdir(path) if f.endswith(".jpg")]

    plt.figure(figsize=(12,6))
    # اختيار صورة عشوائية

    for i in range(n):
        img_name = random.choice(images)
        img_path = os.path.join(path, img_name)
        # استخراج الكلاس من اسم الصورة
        cls = img_name.split("_")[0]
        # فتح الصورة
        img = Image.open(img_path)
        # عرض الصورة

        plt.subplot(1,n,i+1)
        plt.imshow(img)
        plt.title(f"Class {cls}")
        plt.axis("off")

    plt.show()
# عرض 5 صور

show_samples_food11("/content/food11/training")

```



```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# تحديد حجم الصور بعد المعالجة
IMG_SIZE = (224,224)
# أثناء التدريب عدد الصور في كل
BATCH_SIZE = 32
```

```
import os
import shutil

# المسار الأساسي للصور
base_path = "/content/food11/training"

# قراءة كل الصور
images = [f for f in os.listdir(base_path) if f.endswith(".jpg")]

# تنظيم الصور داخل فولدرات حسب الكلاس
for img in images:
    # استخراج اسم الكلاس
    class_label = img.split("_")[0]

    # إنشاء فولدر لكل كلاس
    class_folder = os.path.join(base_path, class_label)
    os.makedirs(class_folder, exist_ok=True)

    # نقل الصورة إلى الفولدر المناسب
    shutil.move(
        os.path.join(base_path, img),
        os.path.join(class_folder, img)
    )

print("Food11 reorganized successfully ")
```

Food11 reorganized successfully

```
# إنشاء ImageDataGenerator لعمل preprocessing و data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,          # تحويل قيم البكسل من 0-255 إلى 0-1
    rotation_range=20,       # تدوير الصور عشوائيًا حتى 20 درجة
    horizontal_flip=True,    # قلب الصورة أفقيًا
    zoom_range=0.2,          # تكبير/تصغير الصورة
    validation_split=0.2     # تخصيص 20% من البيانات للتحقق
)
```

```

# تحميل بيانات التدريب من الفولدر بعد التنظيم
train_data = train_datagen.flow_from_directory(
    "/content/food11/training",
    target_size=IMG_SIZE,          # 224x224 تغيير حجم الصور إلى
    batch_size=BATCH_SIZE,         # عدد الصور في كل batch
    class_mode="categorical",       # تصنيف متعدد الفئات
    subset="training"              # اختيار بيانات التدريب فقط
)

# تحميل بيانات التحقق (20% من البيانات)
val_data = train_datagen.flow_from_directory(
    "/content/food11/training",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="validation"            # اختيار بيانات التحقق
)

```

Found 7896 images belonging to 11 classes.
 Found 1970 images belonging to 11 classes.

```

# algorithm 1 : ResNet50
# الجاهز ResNet50 استيراد موديل
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models

base_model = ResNet50(
    weights="imagenet",             # استخدام أوزان مدربة مسبقًا
    include_top=False,             # حذف طبقة التصنيف الأصلية
    input_shape=(224,224,3)        # حجم الصور
)

# إضافة طبقات جديدة تناسب الداتا الخاصة بنا
x = layers.GlobalAveragePooling2D()(base_model.output) # يل الأبعاد
output = layers.Dense(train_data.num_classes, activation="softmax")

# إنشاء الموديل النهائي
model_resnet = models.Model(base_model.input, output)

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels/94765736/94765736 ————— 0s 0us/step

```
# algorithm 2 : MobileNetV2
# استيراد MobileNetV2
from tensorflow.keras.applications import MobileNetV2

# تحميل الموديل
base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224,224,3)
)

# إضافة طبقات مخصصة
x = layers.GlobalAveragePooling2D()(base_model.output)
output = layers.Dense(train_data.num_classes, activation="softmax")

# إنشاء الموديل
model_mobilenet = models.Model(base_model.input, output)
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_v2_140_140_1000.h5 9406464/9406464 ————— 0s 0us/step

```
# algorithm 3 : EfficientNetB0
# استيراد EfficientNetB0
from tensorflow.keras.applications import EfficientNetB0

# تحميل الموديل
base_model = EfficientNetB0(
    weights="imagenet",
    include_top=False,
    input_shape=(224,224,3)
)

# إضافة طبقات جديدة
x = layers.GlobalAveragePooling2D()(base_model.output)
output = layers.Dense(train_data.num_classes, activation="softmax")

# إنشاء الموديل
model_efficient = models.Model(base_model.input, output)
```

```

# للموديل (تجهيزه للتدريب) دالة لعمل
def compile_model(model):
    model.compile(
        optimizer="adam",           # وازمية تحسين الأوزان
        loss="categorical_crossentropy", # لتصنيف متعدد الفئات
        metrics=["accuracy"]        # مقياس الأداء (الدقة)
    )

```

```

# تجهيز كل موديل للتدريب
compile_model(model_resnet)
compile_model(model_mobilenet)
compile_model(model_efficient)

```

```

# تدريب موديل ResNet50
history_res = model_resnet.fit(
    train_data,           # بيانات التدريب
    validation_data=val_data, # بيانات التحقق
    epochs =5
)

```

```

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapter.py:100:
self._warn_if_super_not_called()
Epoch 1/5
247/247 ██████████ 6191s 25s/step - accuracy: 0.4342 - loss: 0.6191
Epoch 2/5
247/247 ██████████ 6155s 25s/step - accuracy: 0.6592 - loss: 0.5155
Epoch 3/5
247/247 ██████████ 6092s 25s/step - accuracy: 0.7350 - loss: 0.4092
Epoch 4/5
247/247 ██████████ 6090s 25s/step - accuracy: 0.7674 - loss: 0.4090
Epoch 5/5
247/247 ██████████ 5983s 24s/step - accuracy: 0.7787 - loss: 0.3983

```

```
# تدرب موديل MobileNetV2
history_mob = model_mobilenet.fit(
    train_data,
    validation_data=val_data,
    epochs =5
)
```

```
Epoch 1/5
247/247 ————— 1740s 7s/step - accuracy: 0.6234 - loss
Epoch 2/5
247/247 ————— 1689s 7s/step - accuracy: 0.7882 - loss
Epoch 3/5
247/247 ————— 1689s 7s/step - accuracy: 0.8207 - loss
Epoch 4/5
247/247 ————— 1699s 7s/step - accuracy: 0.8390 - loss
Epoch 5/5
247/247 ————— 1711s 7s/step - accuracy: 0.8585 - loss
```

```
# تدرب موديل EfficientNetB0
history_eff = model_efficient.fit(
    train_data,
    validation_data=val_data,
    epochs =5
)
```

```
Epoch 1/5
247/247 ————— 2522s 10s/step - accuracy: 0.7256 - loss
Epoch 2/5
247/247 ————— 2487s 10s/step - accuracy: 0.8493 - loss
Epoch 3/5
247/247 ————— 2875s 12s/step - accuracy: 0.8813 - loss
Epoch 4/5
247/247 ————— 2464s 10s/step - accuracy: 0.8974 - loss
Epoch 5/5
247/247 ————— 2521s 10s/step - accuracy: 0.9082 - loss
```

