```
!pip -q install kaggle timm torch torchvision pillow tqdm scikit-learn

# Upload kaggle.json to /content first (Colab file pane)
!mkdir -p ~/.kaggle
!cp /content/kaggle.json ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json

DATA_DIR="/content/fruitveg_data"
!mkdir -p $DATA_DIR

# Download THIS dataset
!kaggle datasets download -d kritikseth/fruit-and-vegetable-image-recognition

!ls -lah $DATA_DIR
!find $DATA_DIR -maxdepth 6 -type d | head -n 120

Dataset URL: https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-ir
License(s): CC0-1.0
Downloading fruit-and-vegetable-image-recognition.zip to /content/fruitveg_da
 97% 1.92G/1.98G [00:16<00:01, 39.7MB/s]
100% 1.98G/1.98G [00:16<00:00, 132MB/s]
total 20K
drwxr-xr-x  5 root root 4.0K Dec 15 00:27 .
drwxr-xr-x  1 root root 4.0K Dec 15 00:26 ..
drwxr-xr-x 38 root root 4.0K Dec 15 00:27 test
drwxr-xr-x 38 root root 4.0K Dec 15 00:27 train
drwxr-xr-x 38 root root 4.0K Dec 15 00:27 validation
/content/fruitveg_data
/content/fruitveg_data/validation
/content/fruitveg_data/validation/tomato
/content/fruitveg_data/validation/soy beans
/content/fruitveg_data/validation/garlic
/content/fruitveg_data/validation/raddish
/content/fruitveg_data/validation/onion
/content/fruitveg_data/validation/spinach
/content/fruitveg_data/validation/cauliflower
/content/fruitveg_data/validation/potato
/content/fruitveg_data/validation/pomegranate
/content/fruitveg_data/validation/eggplant
/content/fruitveg_data/validation/paprika
/content/fruitveg_data/validation/watermelon
/content/fruitveg_data/validation/sweetpotato
/content/fruitveg_data/validation/apple
/content/fruitveg_data/validation/lemon
/content/fruitveg_data/validation/chilli pepper
/content/fruitveg_data/validation/carrot
/content/fruitveg_data/validation/jalepeno
/content/fruitveg_data/validation/lettuce
/content/fruitveg_data/validation/banana
```

/content/fruitveg_data/validation/kiwi
/content/fruitveg_data/validation/beetroot
/content/fruitveg_data/validation/orange
/content/fruitveg_data/validation/turnip
/content/fruitveg_data/validation/ginger
/content/fruitveg_data/validation/bell pepper
/content/fruitveg_data/validation/mango
/content/fruitveg_data/validation/pineapple
/content/fruitveg_data/validation/cabbage
/content/fruitveg_data/validation/peas
/content/fruitveg_data/validation/cucumber
/content/fruitveg_data/validation/pear
/content/fruitveg_data/validation/sweetcorn
/content/fruitveg_data/validation/capsicum
/content/fruitveg_data/validation/corn
/content/fruitveg_data/validation/grapes
/content/fruitveg_data/test
/content/fruitveg_data/test/tomato
/content/fruitveg_data/test/soy beans
/content/fruitveg_data/test/garlic
/content/fruitveg_data/test/raddish
/content/fruitveg_data/test/onion
/content/fruitveg_data/test/spinach
/content/fruitveg_data/test/cauliflower
/content/fruitveg_data/test/potato
/content/fruitveg_data/test/pomegranate
/content/fruitveg_data/test/eggplant
/content/fruitveg_data/test/paprika
/content/fruitveg_data/test/watermelon
/content/fruitveg_data/test/sweetpotato
/content/fruitveg_data/test/apple
/content/fruitveg_data/test/lemon
/content/fruitveg_data/test/chilli pepper
/content/fruitveg_data/test/carrot
/content/fruitveg_data/test/jalepeno
/content/fruitveg_data/test/lettuce
/content/fruitveg_data/test/banana
/content/fruitveg_data/test/kiwi
/content/fruitveg_data/test/beetroot
/content/fruitveg_data/test/orange
/content/fruitveg_data/test/turnip
/content/fruitveg_data/test/ginger
/content/fruitveg_data/test/bell pepper
/content/fruitveg_data/test/mango
/content/fruitveg_data/test/pineapple
/content/fruitveg_data/test/cabbage
/content/fruitveg_data/test/peas
/content/fruitveg_data/test/cucumber

```
/content/fruitveg_data/test/pear
/content/fruitveg_data/test/sweetcorn
/content/fruitveg_data/test/capsicum
/content/fruitveg_data/test/corn
/content/fruitveg_data/test/grapes
/content/fruitveg_data/train
/content/fruitveg_data/train/tomato
/content/fruitveg_data/train/soy beans
/content/fruitveg_data/train/garlic
/content/fruitveg_data/train/raddish
/content/fruitveg_data/train/onion
/content/fruitveg_data/train/spinach
/content/fruitveg_data/train/cauliflower
/content/fruitveg_data/train/potato
/content/fruitveg_data/train/pomegranate
/content/fruitveg_data/train/eggplant
/content/fruitveg_data/train/paprika
/content/fruitveg_data/train/watermelon
/content/fruitveg_data/train/sweetpotato
/content/fruitveg_data/train/apple
/content/fruitveg_data/train/lemon
/content/fruitveg_data/train/chilli pepper
/content/fruitveg_data/train/carrot
/content/fruitveg_data/train/jalepeno
/content/fruitveg_data/train/lettuce
/content/fruitveg_data/train/banana
/content/fruitveg_data/train/kiwi
/content/fruitveg_data/train/beetroot
/content/fruitveg_data/train/orange
/content/fruitveg_data/train/turnip
/content/fruitveg_data/train/ginger
/content/fruitveg_data/train/bell pepper
/content/fruitveg_data/train/mango
/content/fruitveg_data/train/pineapple
/content/fruitveg_data/train/cabbage
/content/fruitveg_data/train/peas
/content/fruitveg_data/train/cucumber
/content/fruitveg_data/train/pear
/content/fruitveg_data/train/sweetcorn
/content/fruitveg_data/train/capsicum
/content/fruitveg_data/train/corn
/content/fruitveg_data/train/grapes
```

```
import os, glob, json, random
import numpy as np
from PIL import Image

import torch
import torch.nn as nn
```

```
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision import transforms

import timm
from tqdm.auto import tqdm


# CONFIG

DATA_DIR = "/content/fruitveg_data"
EXPORT_DIR = "/content/export_fruitveg_model"
os.makedirs(EXPORT_DIR, exist_ok=True)

SEED = 42
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 12
LR = 3e-4
BACKBONE = "efficientnet_b0"
NUM_WORKERS = 0
PIN_MEMORY = True

IMG_EXTS = (".jpg", ".jpeg", ".png", ".bmp", ".webp")

def seed_all(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)

seed_all(SEED)
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", device)

Device: cuda


# DETECT TRAIN / TEST FOLDERS

def looks_like_imagefolder_root(path: str, min_classes=2, min_images_total=50
    if not os.path.isdir(path):
        return False
    class_dirs = [d for d in glob.glob(os.path.join(path, "*")) if os.path.is
    if len(class_dirs) < min_classes:
        return False
    total = 0
    for cd in class_dirs[:20]:
        try:
```

```
            total += sum(1 for f in os.listdir(cd) if f.lower().endswith(IMG_
        except FileNotFoundError:
            pass
        if total >= min_images_total:
            return True
    return False

train_candidates = []
test_candidates = []
val_candidates = []

for d in glob.glob(os.path.join(DATA_DIR, "**"), recursive=True):
    if not os.path.isdir(d):
        continue
    base = os.path.basename(d).lower()

    if base in ["train", "training"]:
        if looks_like_imagefolder_root(d):
            train_candidates.append(d)

    if base in ["test", "testing"]:
        if looks_like_imagefolder_root(d):
            test_candidates.append(d)

    if base in ["val", "valid", "validation"]:
        if looks_like_imagefolder_root(d):
            val_candidates.append(d)

print("Train candidates:", train_candidates[:5])
print("Val candidates  :", val_candidates[:5])
print("Test candidates :", test_candidates[:5])

assert train_candidates, "No train/training folder found. Check printed folde
train_dir = sorted(train_candidates, key=lambda p: len(p.split(os.sep)))[-1]

eval_dir = None
if test_candidates:
    eval_dir = sorted(test_candidates, key=lambda p: len(p.split(os.sep)))[-1
elif val_candidates:
    eval_dir = sorted(val_candidates, key=lambda p: len(p.split(os.sep)))[-1]
else:
    raise AssertionError("No test/val folder found. Check dataset structure ou

print("Using train_dir:", train_dir)
print("Using eval_dir :", eval_dir)

Train candidates: ['/content/fruitveg_data/train']
Val candidates  : ['/content/fruitveg_data/validation']
Test candidates : ['/content/fruitveg_data/test']
```

```
Using train_dir: /content/fruitveg_data/train
Using eval_dir : /content/fruitveg_data/test


# TRANSFORMS

train_tfms = transforms.Compose([
    transforms.RandomResizedCrop(IMG_SIZE, scale=(0.7, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(0.2, 0.2, 0.2, 0.1),
    transforms.ToTensor(),
])

eval_tfms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(IMG_SIZE),
    transforms.ToTensor(),
])

# DATASETS / LOADERS

train_ds = ImageFolder(train_dir, transform=train_tfms)
eval_ds  = ImageFolder(eval_dir,  transform=eval_tfms)

classes = train_ds.classes
num_classes = len(classes)

print("Num classes:", num_classes)
print("Sample classes:", classes[:15])

train_loader = DataLoader(
    train_ds, batch_size=BATCH_SIZE, shuffle=True,
    num_workers=NUM_WORKERS, pin_memory=PIN_MEMORY
)
eval_loader = DataLoader(
    eval_ds, batch_size=BATCH_SIZE, shuffle=False,
    num_workers=NUM_WORKERS, pin_memory=PIN_MEMORY
)

Num classes: 36
Sample classes: ['apple', 'banana', 'beetroot', 'bell pepper', 'cabbage', 'ca

# MODEL

model = timm.create_model(BACKBONE, pretrained=True, num_classes=num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=EPOCH
```

```
@torch.no_grad()
def eval_loss(loader):
    model.eval()
    total_loss = 0.0
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        logits = model(x)
        loss = criterion(logits, y)
        total_loss += loss.item() * x.size(0)
    return total_loss / len(loader.dataset)

@torch.no_grad()
def topk_accuracy(loader, k=1):
    model.eval()
    correct, total = 0, 0
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        logits = model(x)
        topk = logits.topk(k, dim=1).indices  # (B,k)
        correct += (topk == y.unsqueeze(1)).any(dim=1).sum().item()
        total += y.size(0)
    return correct / max(total, 1)

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
  warnings.warn(



model.safetensors:   0%|          | 0.00/21.4M [00:00<?, ?B/s]

# TRAIN LOOP

best_top1 = 0.0
best_path = os.path.join(EXPORT_DIR, "best_weights.pt")

for epoch in range(1, EPOCHS + 1):
    model.train()
    running = 0.0
    pbar = tqdm(train_loader, desc=f"Epoch {epoch}/{EPOCHS}")

    for x, y in pbar:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        logits = model(x)
```

```
        loss = criterion(logits, y)
        loss.backward()
        optimizer.step()

        running += loss.item() * x.size(0)
        pbar.set_postfix(loss=float(loss.item()))

    scheduler.step()

    train_loss = running / len(train_loader.dataset)
    ev_loss = eval_loss(eval_loader)
    top1 = topk_accuracy(eval_loader, k=1)
    top3 = topk_accuracy(eval_loader, k=3)
    top5 = topk_accuracy(eval_loader, k=5)

    print(f"Epoch {epoch}: train_loss={train_loss:.4f} | eval_loss={ev_loss:.4

    if top1 > best_top1:
        best_top1 = top1
        torch.save(model.state_dict(), best_path)

print("Best Top-1 accuracy:", best_top1)
print("Best weights saved to:", best_path)
```

```
Epoch 1/12:   0%|          | 0/49 [00:00<?, ?it/s]


/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palet
  warnings.warn(


Epoch 1: train_loss=1.9014 | eval_loss=0.3193 | Top1=0.9192 Top3=0.9694 Top5=0



Epoch 2/12:   0%|          | 0/49 [00:00<?, ?it/s]


Epoch 2: train_loss=0.4571 | eval_loss=0.1967 | Top1=0.9415 Top3=0.9833 Top5=0



Epoch 3/12:   0%|          | 0/49 [00:00<?, ?it/s]


Epoch 3: train_loss=0.2484 | eval_loss=0.1654 | Top1=0.9499 Top3=0.9861 Top5=0
```

```
Epoch 4/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 4: train_loss=0.1376 | eval_loss=0.1322 | Top1=0.9582 Top3=0.9861 Top5=0


Epoch 5/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 5: train_loss=0.0894 | eval_loss=0.1540 | Top1=0.9526 Top3=0.9833 Top5=0


Epoch 6/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 6: train_loss=0.0738 | eval_loss=0.1341 | Top1=0.9610 Top3=0.9861 Top5=0


Epoch 7/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 7: train_loss=0.0577 | eval_loss=0.1131 | Top1=0.9638 Top3=0.9861 Top5=0


Epoch 8/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 8: train_loss=0.0440 | eval_loss=0.1069 | Top1=0.9694 Top3=0.9861 Top5=0


Epoch 9/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 9: train_loss=0.0367 | eval_loss=0.1068 | Top1=0.9694 Top3=0.9861 Top5=0


Epoch 10/12:   0%|          | 0/49 [00:00<?, ?it/s]

Epoch 10: train_loss=0.0363 | eval_loss=0.1125 | Top1=0.9694 Top3=0.9861 Top5=
```

```
Epoch 11/12:   0%|          | 0/49 [00:00<?, ?it/s]


Epoch 11: train_loss=0.0342 | eval_loss=0.1069 | Top1=0.9694 Top3=0.9861 Top5=



Epoch 12/12:   0%|          | 0/49 [00:00<?, ?it/s]


Epoch 12: train_loss=0.0287 | eval_loss=0.1044 | Top1=0.9694 Top3=0.9861 Top5=
Best Top-1 accuracy: 0.9693593314763231
Best weights saved to: /content/export_fruitveg_model/best_weights.pt

# FINAL EVAL WITH BEST WEIGHTS

model.load_state_dict(torch.load(best_path, map_location=device))

ev_loss = eval_loss(eval_loader)
top1 = topk_accuracy(eval_loader, k=1)
top3 = topk_accuracy(eval_loader, k=3)
top5 = topk_accuracy(eval_loader, k=5)

print(f"FINAL EVAL: loss={ev_loss:.4f} | Top-1 Acc={top1:.4f} | Top-3 Acc={top

FINAL EVAL: loss=0.1069 | Top-1 Acc=0.9694 | Top-3 Acc=0.9861 | Top-5 Acc=0.98

# EXPORT ARTIFACTS

with open(os.path.join(EXPORT_DIR, "classes.json"), "w") as f:
    json.dump(classes, f, indent=2)

with open(os.path.join(EXPORT_DIR, "model_config.json"), "w") as f:
    json.dump({"backbone": BACKBONE, "img_size": IMG_SIZE}, f, indent=2)

print("Exported files:", os.listdir(EXPORT_DIR))

Exported files: ['best_weights.pt', 'classes.json', 'model_config.json']

# DEMO INFERENCE ON A FEW EVAL IMAGES

eval_image_paths = []
for root, _, files in os.walk(eval_dir):
    for fn in files:
        if fn.lower().endswith(IMG_EXTS):
            eval_image_paths.append(os.path.join(root, fn))
    if len(eval_image_paths) >= 5:
        break
```

```
model.eval()

def predict_image(path):
    img = Image.open(path).convert("RGB")
    x = eval_tfms(img).unsqueeze(0).to(device)
    with torch.no_grad():
        logits = model(x)
        prob = torch.softmax(logits, dim=1)[0]
        idx = int(prob.argmax().item())
        return classes[idx], float(prob[idx].item())

print("\nDemo predictions:")
for p in eval_image_paths[:5]:
    label, conf = predict_image(p)
    print(os.path.basename(p), "->", label, f"({conf:.3f})")

Demo predictions:
Image_3.jpg -> tomato (0.999)
Image_2.jpg -> tomato (1.000)
Image_1.jpg -> tomato (1.000)
Image_9.jpg -> tomato (0.975)
Image_8.jpg -> tomato (0.998)
```