```
Start coding or generate with AI.
```

📌 Setup and Installation (প্যাকেজ ইন্সটল ও লাইব্রেরি ইম্পোর্ট)

```
# !pip install torchsummary seaborn --quiet

# 🔍 এই লাইনে দুইটি লাইব্রেরি ইন্সটল করছি:
# - torchsummary: মডেলের প্রতিটি লেয়ার ও প্যারামিটার দেখতে কাজে লাগে
# - seaborn: ডেটা ও গ্রাফ সুন্দর করে প্লট করার জন্য
```

🔧 Imports এবং Seed ফিক্স করা

```
import os
import random
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from sklearn.metrics import f1_score, classification_report, confusion_matrix, accuracy_
import matplotlib.pyplot as plt
import seaborn as sns
from torchsummary import summary

# 🎯 Reproducibility: প্রতিবার কোড চালালে যেন একই রেজাল্ট পাই সেই জন্য seed সেট করা
def set_seed(seed=42):
    random.seed(seed)               # Python এর random seed
    np.random.seed(seed)            # NumPy এর random seed
    torch.manual_seed(seed)         # PyTorch (CPU) seed
    torch.cuda.manual_seed_all(seed)  # PyTorch (GPU) seed

set_seed(42)

# ✅ ডিভাইস চেক (GPU থাকলে GPU ব্যবহার করবো)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

🗂️ Dataset Preparation (ডেটাসেট লোড, প্রসেসিং ও ভাগ করা)

```
# 🔗 ডেটাসেটের পাথ
DATASET_PATH = '/kaggle/input/fruitvision-a-benchmark-dataset-for-fresh/Fruits Original'

# 🎨 Data Augmentation (ডেটা বৈচিত্র্য বাড়ানোর জন্য)
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(128),       # র‍্যান্ডম ভাবে ইমেজ কাটছেঃ ইনপুটের আকার 128
    transforms.RandomHorizontalFlip(),       # ডান-বামে উল্টে দিচ্ছে (randomly)
    transforms.RandomRotation(15),           # ১৫ ডিগ্রি ঘুরিয়ে দিচ্ছে
    transforms.ColorJitter(brightness=0.2, contrast=0.2), # উজ্জ্বলতা ও কনট্রাস্ট হালকা করে
    transforms.ToTensor(),                   # পিক্সেল ভ্যালুকে টেনসর বানিয়ে দিচ্ছে (0-1 স্কেলে
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])        # ইমেজ নরমালাইজ করে দিচ্ছে (
])

# ✅ Validation ও Test ডেটা শুধু Resize + Normalize
val_test_transforms = transforms.Compose([
    transforms.Resize(140),        # প্রথমে রিসাইজ করা হচ্ছে
    transforms.CenterCrop(128),    # তারপর মাঝখান থেকে কেটে নিচ্ছে
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# 📁 ডেটাসেট লোড করা হচ্ছে ImageFolder থেকে (auto লেবেল নেবে ফোল্ডার নাম দেখে)
full_dataset = datasets.ImageFolder(root=DATASET_PATH, transform=train_transforms)

# 🔢 ডেটা এলোমেলো করে ভাগ করছি
dataset_size = len(full_dataset)
indices = list(range(dataset_size))
random.shuffle(indices)

# 🍼 ভাগ করে নিচ্ছি → 70% Train, 15% Validation, 15% Test
train_end = int(0.7 * dataset_size)
val_end = train_end + int(0.15 * dataset_size)

train_indices = indices[:train_end]
val_indices = indices[train_end:val_end]
test_indices = indices[val_end:]
```

```
# 🔍 Subset তৈরি করছি → index অনুযায়ী
train_dataset = Subset(full_dataset, train_indices)
val_dataset = Subset(full_dataset, val_indices)
test_dataset = Subset(full_dataset, test_indices)

# 🖌️ ভ্যালিডেশন ও টেস্টে ট্রান্সফর্ম বদলানো হচ্ছে (augmentation বাদ)
val_dataset.dataset.transform = val_test_transforms
test_dataset.dataset.transform = val_test_transforms

# 🔁 DataLoader তৈরি করছি (batch-by-batch data process)
BATCH_SIZE = 64
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Train size: {len(train_dataset)}, Val size: {len(val_dataset)}, Test size: {len(
```

🖼️ Sample Image Visualization

```
# 📊 ছবি ঠিকমতো আসছে কিনা সেটা চেক করতে ১০টি ছবি দেখানো
images, labels = next(iter(train_loader))

# ✅ Normalization undo করার জন্য mean এবং std লাগবে
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])

fig, axs = plt.subplots(1, 10, figsize=(18, 3))
for i in range(10):
    img = images[i].permute(1, 2, 0).numpy()
    img = img * std + mean           # Normalization undo
    img = np.clip(img, 0, 1)
    axs[i].imshow(img)
    axs[i].set_title(full_dataset.classes[labels[i]])
    axs[i].axis('off')
plt.tight_layout()
plt.show()
```

🧠 Custom CNN Model Explained (Conv, Padding, etc.)

```
# 🔧 CustomCNN ক্লাস: আমরা নিজেরা মডেলের গঠন বানাচ্ছি
```

```
class CustomCNN(nn.Module):
    def __init__(self, num_classes):  # num_classes = মোট কতটি ক্লাসে ভাগ করতে হবে (যেম
        super().__init__()  # nn.Module-এর ফিচারগুলো কপি নিচ্ছে

        # 🧠 Feature extraction অংশ – এখানে ছবির থেকে গুরুত্বপূর্ণ ফিচার বের করা হয়
        self.features = nn.Sequential(  # nn.Sequential = সবগুলো লেয়ার সিরিয়ালি একটার প

            # -------- 🔹 Block 1 --------
            nn.Conv2d(3, 32, 3, padding=1),  # ইনপুট: 3 চ্যানেল (RGB), আউটপুট: 32 ফিচার
            # Conv2D = ছবি থেকে ফিচার বের করে (filter দিয়ে)
            # kernel_size=3 মানে ৩x৩ ফিল্টার
            # padding=1 মানে চারপাশে ০ বসিয়ে input ও output size একই রাখা

            nn.BatchNorm2d(32),  # ফিচার ম্যাপগুলোকে normalize করে
            nn.ReLU(inplace=True),  # নেগেটিভ ভ্যালু ০ করে দিয়ে মডেলকে নন-লিনিয়ার বানায়

            nn.Conv2d(32, 32, 3, padding=1),  # আরেকটা Conv → ফিচার আরও deep
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),

            nn.MaxPool2d(2),  # ২x২ এর মধ্যে সবচেয়ে বড়টা নেয় → resolution অর্ধেক হয়
            nn.Dropout(0.1),  # কিছু neuron randomly বন্ধ করে → overfitting কমায়

            # -------- 🔹 Block 2 --------
            nn.Conv2d(32, 64, 3, padding=1),  # এখন ফিল্টারের সংখ্যা ৬৪ → বেশি ফিচার ধরা য
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Dropout(0.15),  # আগের চেয়ে একটু বেশি dropout

            # -------- 🔹 Block 3 --------
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Dropout(0.2),
```

```
            # --------  🔹 Block 4 --------
            nn.Conv2d(128, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Dropout(0.25)  # বেশি ড্রপআউট → deep layer এ বেশি generalization দরকার
        )

        # 🎯 Classification অংশ – এখান থেকে ক্লাসের নাম প্রেডিক্ট করা হয়
        self.classifier = nn.Sequential(
            nn.Flatten(),  # ফিচার ম্যাপ (3D) → 1D ভেক্টর বানায়

            nn.Linear(8*8*256, 512),  # ইনপুট ভেক্টরের সাইজ = (8x8x256) → ফিচার ম্যাপের শে
            # এই লেয়ারে ৮x৮ রেজুলুশনের ২৫৬টা ফিচারকে একটা বড় ভেক্টরে রূপান্তর করে

            nn.ReLU(inplace=True),  # ReLU → নন-লিনিয়ারিটি যোগ করে
            nn.Dropout(0.3),        # Overfitting কমানোর জন্য কিছু neuron বন্ধ

            nn.Linear(512, num_classes)  # 🔚 Final লেয়ার → যত ক্লাস তত ইউনিট
        )

    def forward(self, x):
        # x → input image (tensor)
        x = self.features(x)    # প্রথমে Conv part দিয়ে ফিচার বের করি
        x = self.classifier(x)  # তারপর Fully Connected লেয়ার দিয়ে ক্লাস প্রেডিক্ট করি
        return x  # আউটপুট রিটার্ন
```

🏋️ Training Loop (মডেল প্রশিক্ষণ দেওয়া হচ্ছে)

```python
# ✅ Loss Function → কতটা ভুল হচ্ছে তা মাপার উপায়
criterion = nn.CrossEntropyLoss()

# ✅ Optimizer → ওজন আপডেট করার নিয়ম
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 💾 Best ফাইল সংরক্ষণের জন্য ভ্যারিয়েবল
best_f1 = 0.0

# 📈 গ্রাফ আঁকার জন্য তথ্য জমা রাখা হবে
```

```
train_losses, val_losses, val_f1s = [], [], []

# 🔁 কতবার পুরো ডেটা দিয়ে ট্রেন করবো
num_epochs = 10

# 🔁 প্রত্যেক epoch এর জন্য loop
for epoch in range(num_epochs):
    model.train()  # Train Mode ON (Dropout, BatchNorm active)

    train_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()           # আগের গ্র্যাডিয়েন্ট মুছে ফেলি
        outputs = model(inputs)         # মডেল প্রেডিকশন করে
        loss = criterion(outputs, labels)  # ভুল কতটুকু তা হিসাব করি
        loss.backward()                 # গ্র্যাডিয়েন্ট বের করি
        optimizer.step()                # ওজন আপডেট করি
        train_loss += loss.item() * inputs.size(0)  # মোট ট্রেন লস যোগ করি

    train_loss /= len(train_loader.dataset)  # গড় ট্রেন লস

    # ------------------------------
    # 🔎 Validation: শেখার সময় মডেল কেমন করছে?
    # ------------------------------
    model.eval()  # Eval Mode ON (Dropout, BatchNorm OFF)
    val_loss = 0.0
    all_preds, all_labels = [], []

    with torch.no_grad():  # মেমরি বাঁচানোর জন্য গ্র্যাডিয়েন্ট বন্ধ
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * inputs.size(0)

            preds = outputs.argmax(dim=1)  # সবচেয়ে বড় logit কে প্রেডিকশন ধরা
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    val_loss /= len(val_loader.dataset)  # গড় ভ্যালিডেশন লস
    val_f1 = f1_score(all_labels, all_preds, average='weighted')  # F1 → ক্লাস ব্যালেন্সড মে
```

```
    # 📊 তথ্য জমা রাখি
    train_losses.append(train_loss)
    val_losses.append(val_loss)
    val_f1s.append(val_f1)

    # ℹ️ স্ট্যাটাস প্রিন্ট
    print(f"Epoch [{epoch+1}/{num_epochs}] Train Loss: {train_loss:.4f} Val Loss: {val_l

    # 💾 বেস্ট F1 থাকলে সেই মডেল সংরক্ষণ করি
    if val_f1 > best_f1:
        best_f1 = val_f1
        torch.save(model.state_dict(), 'custom_cnn_best.pt')
        print("Best model saved!")

print("Training complete!")
```

📈 Loss এবং F1 Score এর গ্রাফ আঁকা

```
# 🔢 x-axis: Epochs
epochs = range(1, num_epochs + 1)

plt.figure(figsize=(12,5))

# 🔹 লস গ্রাফ
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label='Train Loss', marker='o')
plt.plot(epochs, val_losses, label='Val Loss', marker='s')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curves')
plt.legend()
plt.grid(True)

# 🔸 F1 স্কোর গ্রাফ
plt.subplot(1, 2, 2)
plt.plot(epochs, val_f1s, label='Validation F1 Score', color='green', marker='^')
plt.xlabel('Epoch')
plt.ylabel('F1 Score')
plt.title('Validation F1 Score Curve')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

🧪 Final Evaluation on Test Set (টেস্ট ডেটায় মডেল পরীক্ষা)

```
# 💾 সেরা মডেল লোড করলাম
model.load_state_dict(torch.load('custom_cnn_best.pt', map_location=device))
model.eval()

all_preds, all_labels = [], []

# 🔍 টেস্ট ডেটা দিয়ে প্রেডিকশন
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        preds = outputs.argmax(dim=1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# 📋 রিপোর্ট ও একুরেসি প্রিন্ট
print("\n📊 Classification Report:\n")
print(classification_report(all_labels, all_preds, target_names=full_dataset.classes))

acc = accuracy_score(all_labels, all_preds)
print(f"✅ Test Accuracy: {acc:.4f}")

# 📉 Confusion Matrix প্লট করা
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=full_dataset.classes,
            yticklabels=full_dataset.classes)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

✅ Extra Help (Bonus): Conv2d ও Padding কী?

- `nn.Conv2d(in_channels, out_channels, kernel_size, padding)`
  → Conv লেয়ার ইমেজের ফিচার খুঁজে বের করে (edge, texture ইত্যাদি)
  → padding=1 দিলে image সাইজ ধরে রাখে (সীমার বাইরে 0 বসায়)

- `nn.MaxPool2d(2)`
  → ২x২ ব্লকের মধ্যে সবচেয়ে বড় ভ্যালু রাখে
  → image সাইজ কমায়, important ফিচার রাখে

- `Dropout`
  → training এ কিছু neuron বন্ধ করে দেয় random ভাবে
  → ওভারফিটিং কমায়

- `BatchNorm`
  → প্রত্যেক লেয়ারের ইনপুটকে স্ট্যাবল রাখে (normalization করে)