Importing Libraries and Data

```
!pip3 install pandas  matplotlib seaborn scikit-learn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->
```

```
from google.colab import drive
drive.mount('/content/drive')
```

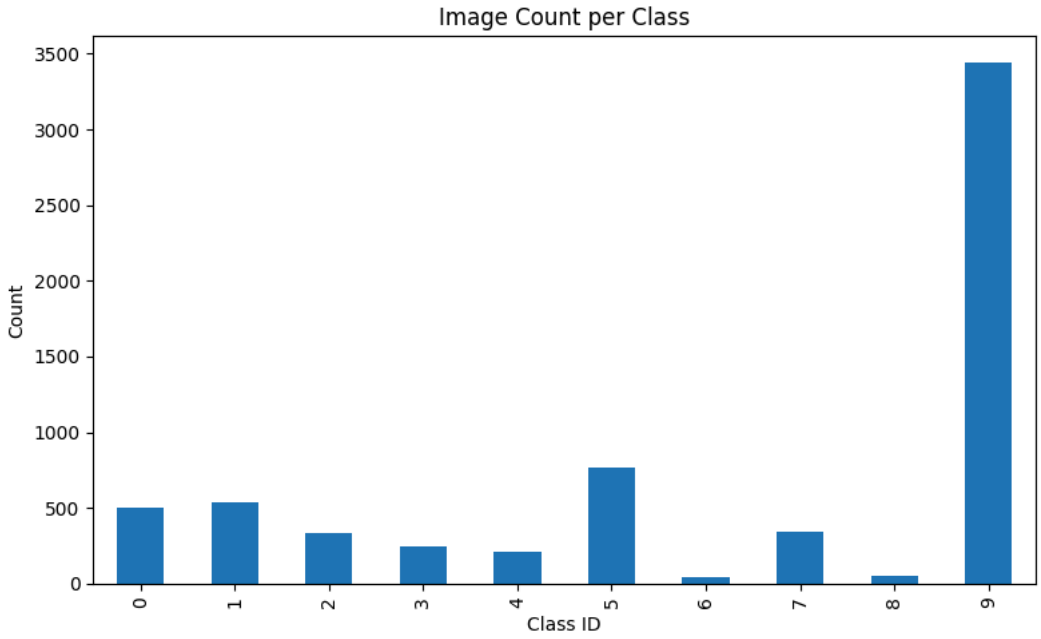
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("rtlmhjbn/ip02-dataset")

print("Downloaded to", path)
```

```
Using Colab cache for faster access to the 'ip02-dataset' dataset.
Downloaded to /kaggle/input/ip02-dataset
```

```
import os
from pathlib import Path
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt

# Define dataset root and selected folders
DATASET_PATH = Path("/kaggle/input/ip02-dataset/classification/train")
SELECTED_FOLDERS = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]

data_summary = []

for class_id in SELECTED_FOLDERS:
    class_path = DATASET_PATH / str(class_id)
    if not class_path.exists():
        print(f"⚠️ Folder {class_id} not found.")
        continue

    image_files = list(class_path.glob("*.jpg")) + list(class_path.glob("*.png"))
    image_count = len(image_files)

    # Optional: sample first image to get resolution
    if image_files:
        with Image.open(image_files[0]) as img:
            width, height = img.size
    else:
        width, height = (0, 0)

    data_summary.append({
        "class_id": class_id,
        "num_images": image_count,
        "sample_width": width,
        "sample_height": height
    })

# Save summary
df = pd.DataFrame(data_summary)
```

```
df["num_images"].plot(kind="bar", title="Image Count per Class", figsize=(8, 5))
plt.ylabel("Count")
plt.xlabel("Class ID")
plt.tight_layout()
plt.show()

print(df)
df.to_csv("ip102_selected_classes_summary.csv", index=False)
```



	class_id	num_images	sample_width	sample_height
0	7	500	720	540
1	8	535	216	170
2	9	331	433	268
3	11	242	200	200
4	59	212	1800	1200
5	69	767	676	439
6	72	42	238	170
7	94	346	268	180
8	98	55	341	512
9	101	3444	533	800

STEP 1: Classical Augmentation

==========

Tools: imgaug, torchvision.transforms, Albumentations

Examples: rotations, flips, scaling, brightness, hue, random crops.

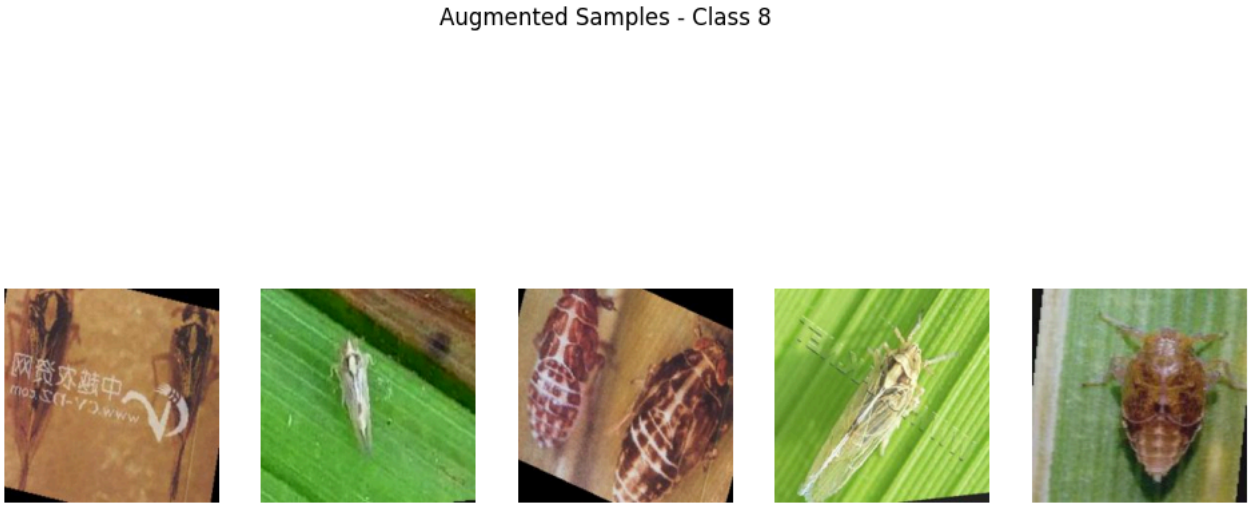
Implementation:

```
# ================================
import os
from pathlib import Path
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt
import random

# Mount Google Drive if dataset is there
# from google.colab import drive
# drive.mount('/content/drive')

DATA_PATH = Path("/kaggle/input/ip02-dataset/classification/train")  # Example path
SAVE_PATH = Path("/content/IP102_augmented")
SAVE_PATH.mkdir(parents=True, exist_ok=True)

# Selected classes
SELECTED_CLASSES = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]

# Define augmentations
augment = transforms.Compose([
    transforms.RandomRotation(25),
    transforms.RandomHorizontalFlip(p=0.7),
    transforms.RandomVerticalFlip(p=0.3),
```

```
        transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.2, hue=0.02),
        transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0))
    ])

    # How many augmentations per image
    AUG_PER_IMAGE = 3

    for class_id in SELECTED_CLASSES:
        src_dir = DATA_PATH / str(class_id)
        target_dir = SAVE_PATH / str(class_id)
        target_dir.mkdir(parents=True, exist_ok=True)

        image_files = list(src_dir.glob("*.jpg")) + list(src_dir.glob("*.png"))
        print(f"Processing Class {class_id}: {len(image_files)} images")

        for img_path in image_files:
            try:
                img = Image.open(img_path).convert("RGB")
            except:
                continue

            for i in range(AUG_PER_IMAGE):
                aug_img = augment(img)
                save_name = f"{img_path.stem}_aug{i}.jpg"
                aug_img.save(target_dir / save_name)

    print("✅ Classical Augmentation complete!")
```

```
Processing Class 7: 500 images
Processing Class 8: 535 images
Processing Class 9: 331 images
Processing Class 11: 242 images
Processing Class 59: 212 images
Processing Class 69: 767 images
Processing Class 72: 42 images
Processing Class 94: 346 images
Processing Class 98: 55 images
Processing Class 101: 3444 images
✅ Classical Augmentation complete!
```

```
# Visualize few augmented examples
sample_class = random.choice(SELECTED_CLASSES)
sample_folder = SAVE_PATH / str(sample_class)
samples = list(sample_folder.glob("*.jpg"))
plt.figure(figsize=(12, 6))
for i, img_path in enumerate(random.sample(samples, 5)):
    img = Image.open(img_path)
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis("off")
plt.suptitle(f"Augmented Samples - Class {sample_class}")
plt.show()
```



STEP 2: Random Erasing / Cutout

Improves robustness by obscuring random image regions.

Tools: torchvision.transforms.RandomErasing, or custom Cutout implementation.

```
# ================================

import torch
from torchvision import transforms
from torchvision.transforms import functional as F
import matplotlib.pyplot as plt
from PIL import Image
import random
from pathlib import Path

DATA_PATH = Path("/kaggle/input/ip02-dataset/classification/train")
SAVE_PATH = Path("/content/IP102_cutout")
SAVE_PATH.mkdir(parents=True, exist_ok=True)

SELECTED_CLASSES = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]

# Built-in RandomErasing + Custom Cutout Example
cutout_transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.RandomErasing(p=0.9, scale=(0.02, 0.2), ratio=(0.3, 3.3), value='random'),
])

def apply_cutout_and_save(img_path, target_dir, n_aug=2):
    image = Image.open(img_path).convert("RGB")
    for i in range(n_aug):
        tensor_img = cutout_transform(image)
        cut_img = F.to_pil_image(tensor_img)
        cut_img.save(target_dir / f"{img_path.stem}_cutout{i}.jpg")

for class_id in SELECTED_CLASSES:
    src_dir = DATA_PATH / str(class_id)
    tgt_dir = SAVE_PATH / str(class_id)
    tgt_dir.mkdir(parents=True, exist_ok=True)

    image_files = list(src_dir.glob("*.jpg")) + list(src_dir.glob("*.png"))
    print(f"Applying Cutout to class {class_id} ({len(image_files)} images)")

    for img_path in image_files:
        try:
            apply_cutout_and_save(img_path, tgt_dir)
        except Exception as e:
            print(f"Error: {img_path} -> {e}")
            continue

print("✅ Random Erasing / Cutout augmentation complete!")
```

```
Applying Cutout to class 7 (500 images)
Applying Cutout to class 8 (535 images)
Applying Cutout to class 9 (331 images)
Applying Cutout to class 11 (242 images)
Applying Cutout to class 59 (212 images)
Applying Cutout to class 69 (767 images)
Applying Cutout to class 72 (42 images)
Applying Cutout to class 94 (346 images)
Applying Cutout to class 98 (55 images)
Applying Cutout to class 101 (3444 images)
✅ Random Erasing / Cutout augmentation complete!
```

```
# Preview cutout images
import random
sample_class = random.choice(SELECTED_CLASSES)
sample_folder = SAVE_PATH / str(sample_class)
samples = list(sample_folder.glob("*.jpg"))

plt.figure(figsize=(12, 6))
for i, img_path in enumerate(random.sample(samples, 5)):
    img = Image.open(img_path)
    plt.subplot(1, 5, i + 1)
    plt.imshow(img)
    plt.axis("off")
plt.suptitle(f"Random Erasing / Cutout Samples – Class {sample_class}")
plt.show()
```

Start coding or generate with AI.

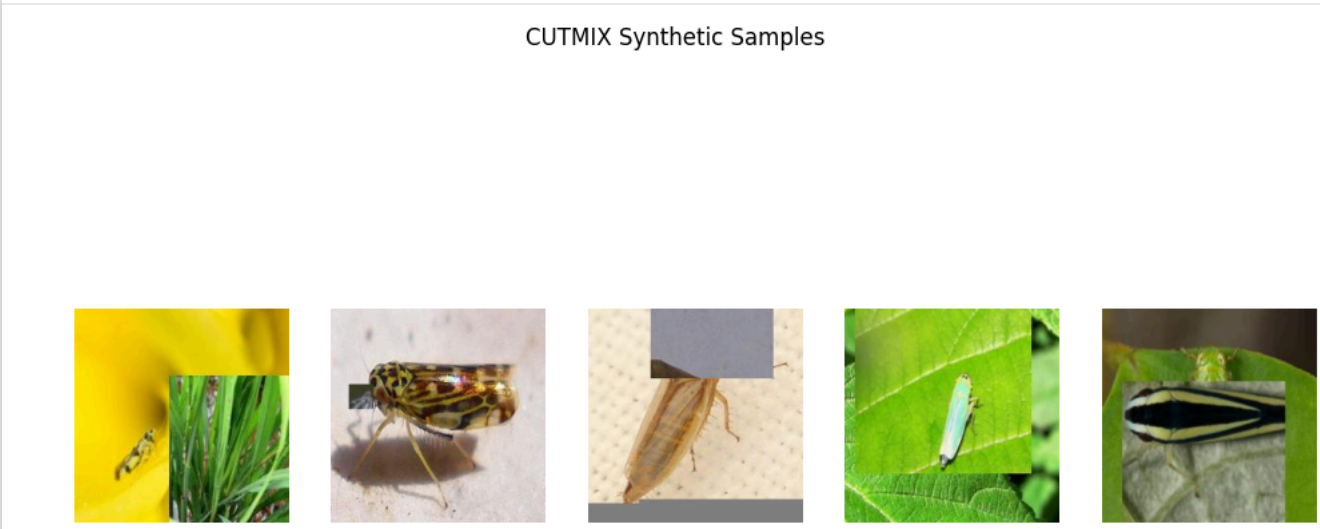
STEP 3: MixUp and CutMix Augmentation

```
# ========================================

import torch
import torchvision.transforms as transforms
from torchvision.utils import save_image
from PIL import Image
import random
import os
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt

# Paths
DATA_PATH = Path("/kaggle/input/ip02-dataset/classification/train")
SAVE_PATH = Path("/content/IP102_mixup_cutmix")
SAVE_PATH.mkdir(parents=True, exist_ok=True)

SELECTED_CLASSES = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]
IMG_SIZE = 224

# --- Utility to gather filepaths from selected classes ---
def get_class_images(cls):
    cls_path = DATA_PATH / str(cls)
    files = list(cls_path.glob("*.jpg")) + list(cls_path.glob("*.png"))
    return files

all_images = []
for c in SELECTED_CLASSES:
    for f in get_class_images(c):
        all_images.append((f, c))

# --- Base transforms ---
base_tf = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor()
])

def mixup(img1, img2, alpha=0.3):
    """Combine two images linearly."""
    lam = np.random.beta(alpha, alpha)
    return lam * img1 + (1 - lam) * img2, lam

def cutmix(img1, img2, alpha=1.0):
    """Replace rectangular region of img1 with patch from img2."""
    lam = np.random.beta(alpha, alpha)
    W, H = img1.size()[2], img1.size()[1]
    cut_rat = np.sqrt(1. - lam)
    cut_w, cut_h = int(W * cut_rat), int(H * cut_rat)

    # random center
    cx, cy = np.random.randint(W), np.random.randint(H)
    x1 = np.clip(cx - cut_w // 2, 0, W)
    y1 = np.clip(cy - cut_h // 2, 0, H)
```

```
        x2 = np.clip(cx + cut_w // 2, 0, W)
        y2 = np.clip(cy + cut_h // 2, 0, H)

        img1[:, y1:y2, x1:x2] = img2[:, y1:y2, x1:x2]
        return img1, lam

    # --- Generate synthetic samples ---
    for i in range(80):  # number of synthetic pairs (adjust as needed)
        (f1, c1), (f2, c2) = random.sample(all_images, 2)
        img1, img2 = base_tf(Image.open(f1).convert("RGB")), base_tf(Image.open(f2).convert("RGB"))

        # Randomly choose MixUp or CutMix
        if random.random() < 0.5:
            aug_img, lam = mixup(img1, img2)
            save_dir = SAVE_PATH / "mixup"
            aug_type = "MixUp"
        else:
            aug_img, lam = cutmix(img1.clone(), img2.clone())
            save_dir = SAVE_PATH / "cutmix"
            aug_type = "CutMix"

        save_dir.mkdir(parents=True, exist_ok=True)
        save_name = f"{aug_type}_{c1}_{c2}_{str(i).zfill(3)}.jpg"
        save_image(aug_img, save_dir / save_name)

    print("✅ MixUp & CutMix synthetic images generated.")
```

```
✅ MixUp & CutMix synthetic images generated.
```

```
# Display random samples from MixUp & CutMix
sample_type = random.choice(["mixup", "cutmix"])
sample_folder = SAVE_PATH / sample_type
samples = list(sample_folder.glob("*.jpg"))

plt.figure(figsize=(12,6))
for i, p in enumerate(random.sample(samples, 5)):
    img = Image.open(p)
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis("off")
plt.suptitle(f"{sample_type.upper()} Synthetic Samples")
plt.show()
```




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

STEP 4: DCGAN - Deep Convolutional GAN

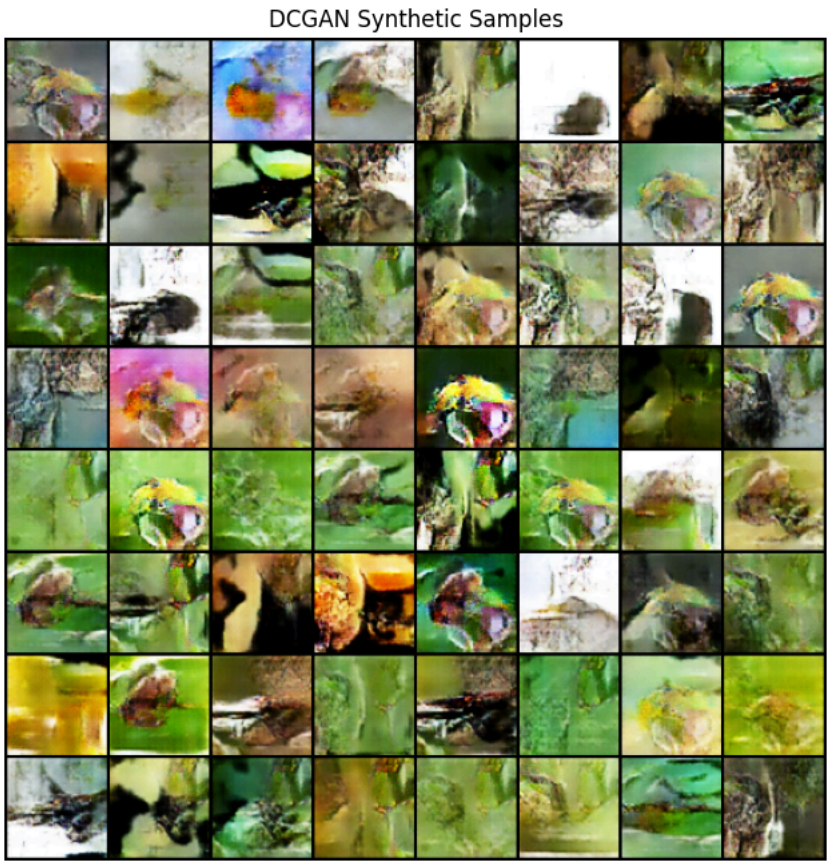
```
# ============================================
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision.utils import save_image, make_grid
from torch.utils.data import DataLoader, Dataset
from PIL import Image
from pathlib import Path
```

```
import matplotlib.pyplot as plt
import os
import random

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Running on: {device}")

# Path setup
DATA_PATH = Path("/kaggle/input/ip02-dataset/classification/train/7")  # pick one class for initial training
SAVE_PATH = Path("/content/DCGAN_output")
SAVE_PATH.mkdir(parents=True, exist_ok=True)

# Hyperparameters
image_size = 64        # reduce memory cost
nc = 3                 # RGB
nz = 100               # noise vector
ngf = 64
ndf = 64
epochs = 10            # for demo: increase later
batch_size = 64
lr = 0.0002
beta1 = 0.5

# Dataset + transforms
transform = transforms.Compose([
    transforms.Resize(image_size),
    transforms.CenterCrop(image_size),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = torchvision.datasets.ImageFolder(root=DATA_PATH.parent, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=2)

# ----------------------------
# Define DCGAN architecture
# ----------------------------

# Generator
class Generator(nn.Module):
    def __init__(self, nz, ngf, nc):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)

# Discriminator
class Discriminator(nn.Module):
    def __init__(self, nc, ndf):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf * 4, 1, 4, 2, 1, bias=False),  # --> down to (B,1,H',W')
            nn.AdaptiveAvgPool2d(1),                     # --> (B,1,1,1)
            nn.Sigmoid()
        )
```

```
        def forward(self, input):
            return self.main(input).view(-1)

    # Initialize models
    netG = Generator(nz, ngf, nc).to(device)
    netD = Discriminator(nc, ndf).to(device)

    # Loss and optimizers
    criterion = nn.BCELoss()
    optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
    optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))

    fixed_noise = torch.randn(64, nz, 1, 1, device=device)

    # ---------------------------
    # Training Loop
    # ---------------------------
    for epoch in range(epochs):
        for i, (real_imgs, _) in enumerate(dataloader):
            b_size = real_imgs.size(0)
            real_imgs = real_imgs.to(device)
            real_label = torch.full((b_size,), 1.0, device=device)
            fake_label = torch.full((b_size,), 0.0, device=device)

            # (1) Train Discriminator
            netD.zero_grad()
            output = netD(real_imgs)
            lossD_real = criterion(output, real_label)

            noise = torch.randn(b_size, nz, 1, 1, device=device)
            fake_imgs = netG(noise)
            output = netD(fake_imgs.detach())
            lossD_fake = criterion(output, fake_label)
            lossD = lossD_real + lossD_fake
            lossD.backward()
            optimizerD.step()

            # (2) Train Generator
            netG.zero_grad()
            output = netD(fake_imgs)
            lossG = criterion(output, real_label)
            lossG.backward()
            optimizerG.step()

        print(f"Epoch [{epoch+1}/{epochs}] | D_loss: {lossD.item():.3f} | G_loss: {lossG.item():.3f}")

        with torch.no_grad():
            fake = netG(fixed_noise).detach().cpu()
            save_image(fake, SAVE_PATH / f"fake_epoch_{epoch+1:03}.png", normalize=True, nrow=8)
```

```
Running on: cpu
Epoch [1/10] | D_loss: 0.855 | G_loss: 2.181
Epoch [2/10] | D_loss: 0.779 | G_loss: 1.805
Epoch [3/10] | D_loss: 0.799 | G_loss: 2.971
Epoch [4/10] | D_loss: 0.787 | G_loss: 1.828
Epoch [5/10] | D_loss: 1.203 | G_loss: 1.159
Epoch [6/10] | D_loss: 0.874 | G_loss: 2.365
Epoch [7/10] | D_loss: 0.976 | G_loss: 1.467
Epoch [8/10] | D_loss: 0.766 | G_loss: 1.681
Epoch [9/10] | D_loss: 0.823 | G_loss: 1.659
Epoch [10/10] | D_loss: 0.603 | G_loss: 1.926
```

```
# Display the latest epoch synthetic images
latest_img = sorted(SAVE_PATH.glob("fake_epoch_*.png"))[-1]
img = Image.open(latest_img)
plt.figure(figsize=(8,8))
plt.imshow(img)
plt.axis("off")
plt.title("DCGAN Synthetic Samples")
plt.show()
```



STEP 5: StyleGAN2-ADA-PyTorch

```
!git clone https://github.com/NVlabs/stylegan2-ada-pytorch.git
%cd stylegan2-ada-pytorch

# Install dependencies
! pip install ninja opensimplex requests tqdm matplotlib

# Optional: connect to Google Drive for dataset and outputs
# from google.colab import drive
# drive.mount('/content/drive')
```

```
Cloning into 'stylegan2-ada-pytorch'...
remote: Enumerating objects: 131, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 131 (delta 0), reused 0 (delta 0), pack-reused 129 (from 2)
Receiving objects: 100% (131/131), 1.13 MiB | 8.33 MiB/s, done.
Resolving deltas: 100% (57/57), done.
/content/stylegan2-ada-pytorch
Collecting ninja
  Downloading ninja-1.13.0-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (5.1 kB)
Collecting opensimplex
  Downloading opensimplex-0.4.5.1-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (4.67.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.12/dist-packages (from opensimplex) (2.0.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests) (2.5
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests) (202
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->ma
Downloading ninja-1.13.0-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (180 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 180.7/180.7 kB 4.6 MB/s eta 0:00:00
Downloading opensimplex-0.4.5.1-py3-none-any.whl (267 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.0/268.0 kB 12.5 MB/s eta 0:00:00
Installing collected packages: opensimplex, ninja
```

```
Successfully installed ninja-1.13.0 opensimplex-0.4.5.1
```

```
# ======================
# Preparing the dataset
# ======================
# /content/IP102_selected/
#    ├── class8/
#    ├── class9/
#    └── class10/
```

```
from google.colab import drive
drive.mount('/content/drive')

# Example paths
# Assuming: /content/drive/MyDrive/IP102/classes/
!mkdir -p /content/IP102_selected/{class8,class9,class10}

# Copy from your Drive folders (adjust source)
!cp "/kaggle/input/ip02-dataset/classification/train/7"*.jpg /content/IP102_selected/class8/
!cp "/kaggle/input/ip02-dataset/classification/train/8"*.jpg /content/IP102_selected/class9/
!cp "/kaggle/input/ip02-dataset/classification/train/9"*.jpg /content/IP102_selected/class10/
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
cp: cannot stat '/kaggle/input/ip02-dataset/classification/train/7*.jpg': No such file or directory
cp: cannot stat '/kaggle/input/ip02-dataset/classification/train/8*.jpg': No such file or directory
cp: cannot stat '/kaggle/input/ip02-dataset/classification/train/9*.jpg': No such file or directory
```

```
!python3 dataset_tool.py --source=/content/IP102_selected \
                         --dest=./datasets/ip102_stylegan \
                         --resolution=256
```

```
# Example: initialize from AFHQ-cat (similar texture variety)
!wget https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/afhqcat.pkl -O pretrained.pkl

!python train.py \
  --outdir=./training-runs/ip102-stylegan \
  --data=./datasets/ip102_stylegan \
  --gpus=1 \
  --batch=16 \
  --cfg=auto \
  --mirror=1 \
  --resume=pretrained.pkl \
  --gamma=10 \
  --snap=5
```

```
# Pick snapshot name
SNAPSHOT="./training-runs/ip102-stylegan/00000*/network-snapshot-00010.pkl"

!python generate.py \
  --outdir=./generated_samples \
  --trunc=0.7 \
  --seeds=0-49 \
  --network=$SNAPSHOT
```

```
import matplotlib.pyplot as plt
import glob
from PIL import Image

imgs = sorted(glob.glob('./generated_samples/*.png'))[:8]
plt.figure(figsize=(12,6))
for i, path in enumerate(imgs):
    plt.subplot(2,4,i+1)
    img = Image.open(path)
    plt.imshow(img)
    plt.axis("off")
plt.suptitle("StyleGAN2-ADA Synthetic Insect Samples", fontsize=14)
plt.show()
```

STEP 6: CycleGAN — Cross-Domain Image Translation

```
# Clone lightweight CycleGAN repo
!git clone https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.git
%cd pytorch-CycleGAN-and-pix2pix
!pip install dominate visdom
```

```
Cloning into 'pytorch-CycleGAN-and-pix2pix'...
remote: Enumerating objects: 2619, done.
remote: Total 2619 (delta 0), reused 0 (delta 0), pack-reused 2619 (from 1)
Receiving objects: 100% (2619/2619), 8.23 MiB | 11.79 MiB/s, done.
Resolving deltas: 100% (1654/1654), done.
/content/stylegan2-ada-pytorch/pytorch-CycleGAN-and-pix2pix
Collecting dominate
  Downloading dominate-2.9.1-py2.py3-none-any.whl.metadata (13 kB)
Collecting visdom
  Downloading visdom-0.2.4.tar.gz (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 20.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.12/dist-packages (from visdom) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from visdom) (1.16.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from visdom) (2.32.4)
Requirement already satisfied: tornado in /usr/local/lib/python3.12/dist-packages (from visdom) (6.5.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (from visdom) (1.17.0)
Requirement already satisfied: jsonpatch in /usr/local/lib/python3.12/dist-packages (from visdom) (1.33)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.12/dist-packages (from visdom) (1.9.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from visdom) (3.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from visdom) (11.3.0)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dist-packages (from jsonpatch->visdo
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->visdom) (3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->visd
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->visd
Downloading dominate-2.9.1-py2.py3-none-any.whl (29 kB)
Building wheels for collected packages: visdom
  Building wheel for visdom (setup.py) ... done
  Created wheel for visdom: filename=visdom-0.2.4-py3-none-any.whl size=1408195 sha256=e405553becf3ae31a5027dc37a
  Stored in directory: /root/.cache/pip/wheels/37/6c/38/64eeaa310e325aacda723e6df1f79ab5e9f31ba195264e04a8
Successfully built visdom
Installing collected packages: dominate, visdom
Successfully installed dominate-2.9.1 visdom-0.2.4
```

```
# datasets/ip102A2B/
#     ├── trainA/  --> class 8 images
#     ├── trainB/  --> class 9 images
#     ├── testA/
#     └── testB/
```

```
!python train.py \
  --dataroot ./datasets/ip102A2B \
  --name ip102A2B_cyclegan \
  --model cycle_gan \
  --batch_size 2 \
  --num_threads 4 \
  --gpu_ids 0
```

```
!python test.py \
  --dataroot ./datasets/ip102A2B \
  --name ip102A2B_cyclegan \
  --model test \
  --num_test 20
```

STEP 7: Conditional GAN (cGANs)

cGANs are trained with labels so that (G(z|y)) generates an image specifically belonging to class (y).

Ideal for multiclass datasets (like IP102 7–101).

```
import torch, torch.nn as nn
from torchvision.utils import save_image
import os, numpy as np

num_classes = 10
nz, ngf, ndf, nc = 100, 64, 64, 3

class G_cGAN(nn.Module):
    def __init__(self):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, nz)
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf*8, 4, 1, 0),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf*8, ngf*4, 4, 2, 1),
            nn.BatchNorm2d(ngf*4),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf*4, ngf*2, 4, 2, 1),
```

```
            nn.BatchNorm2d(ngf*2),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf*2, nc, 4, 2, 1),
            nn.Tanh()
        )
    def forward(self, noise, labels):
        z = noise + self.label_emb(labels).unsqueeze(2).unsqueeze(3)
        return self.main(z)
```

```
z = torch.randn(batch, nz, 1, 1, device=device)
labels = torch.randint(0, num_classes, (batch,), device=device)
fake = G(z, labels)
```

STEP 8: Variational Autoencoder (VAE)

VAEs learn an encoding → latent distribution → decoding cycle.

You can sample new latent vectors for smooth synthetic variants.

```
import torch, torch.nn as nn, torch.nn.functional as F
from torchvision import datasets, transforms
from torchvision.utils import save_image

class VAE(nn.Module):
    def __init__(self, latent_dim=64):
        super().__init__()
        self.enc = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1), nn.ReLU(),
            nn.Conv2d(64, 128, 4, 2, 1), nn.ReLU(),
            nn.Flatten()
        )
        self.fc_mu = nn.Linear(128*56*56, latent_dim)
        self.fc_logvar = nn.Linear(128*56*56, latent_dim)
        self.fc_dec = nn.Linear(latent_dim, 128*56*56)
        self.dec = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 4, 2, 1), nn.ReLU(),
            nn.ConvTranspose2d(64, 3, 4, 2, 1), nn.Sigmoid()
        )
    def encode(self, x):
        h = self.enc(x)
        return self.fc_mu(h), self.fc_logvar(h)
    def reparam(self, mu, logvar):
        std, eps = torch.exp(0.5*logvar), torch.randn_like(logvar)
        return mu + eps*std
    def decode(self, z):
        h = F.relu(self.fc_dec(z)).view(-1,128,56,56)
        return self.dec(h)
    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparam(mu, logvar)
        return self.decode(z), mu, logvar
```

STEP 9: Diffusion Models (Denoising Diffusion Probabilistic Models)

Diffusion models (e.g., DDPM / Stable Diffusion) gradually denoise random noise → image.

They yield state-of-the-art fidelity.

```
!pip install diffusers transformers accelerate safetensors
```

```
Requirement already satisfied: diffusers in /usr/local/lib/python3.12/dist-packages (0.35.2)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.1)
Requirement already satisfied: accelerate in /usr/local/lib/python3.12/dist-packages (1.11.0)
Requirement already satisfied: safetensors in /usr/local/lib/python3.12/dist-packages (0.6.2)
Requirement already satisfied: importlib_metadata in /usr/local/lib/python3.12/dist-packages (from diffusers) (8.
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from diffusers) (3.20.0)
Requirement already satisfied: huggingface-hub>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from diffusers
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from diffusers) (2.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from diffusers) (202
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from diffusers) (2.32.4)
Requirement already satisfied: Pillow in /usr/local/lib/python3.12/dist-packages (from diffusers) (11.3.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from trans
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from accelerate) (2.8.0+c
```

```
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggin
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelera
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accel
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from tor
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from tor
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accel
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.12/dist-packages (from importlib_metadata->di
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->diffusers)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->diff
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->diff
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3-
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=2.
```

```
from diffusers import DDPMPipeline

model_id = "google/ddpm-cifar10-32"  # small pretrained diffusion
pipe = DDPMPipeline.from_pretrained(model_id)

images = pipe(batch_size=8, num_inference_steps=50, output_type="pil").images
for i,img in enumerate(images):
    img.save(f"diffusion_synth_{i}.png")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
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 (https://huggingface.co/settings/t
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model_index.json: 100%                                                  180/180 [00:00<00:00, 14.7kB/s]
Fetching 4 files: 100%                                                  4/4 [00:02<00:00,  1.51s/it]
scheduler_config.json: 100%                                             256/256 [00:00<00:00, 24.1kB/s]
config.json: 100%                                                       699/699 [00:00<00:00, 24.7kB/s]
diffusion_pytorch_model.bin: 100%                                       143M/143M [00:02<00:00, 73.4MB/s]
Loading pipeline components...: 100%                                    2/2 [00:00<00:00,  1.60it/s]
An error occurred while trying to fetch /root/.cache/huggingface/hub/models--google--ddpm-cifar10-32/snapshots/26
Defaulting to unsafe serialization. Pass `allow_pickle=False` to raise an error instead.
100%                                                                    50/50 [01:39<00:00,  1.81s/it]
```

STEP 10: SMOTE for Images (DeepSMOTE)

```
from imblearn.over_sampling import SMOTE
import torch, os
from torchvision import models, transforms
from PIL import Image
import numpy as np

# Feature extractor
resnet = models.resnet18(pretrained=True)
resnet.fc = torch.nn.Identity()
resnet.eval()

tfm = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

SELECTED_CLASSES = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]
DATA_ROOT = "/kaggle/input/ip02-dataset/classification/train/"
```

```
all_embeddings, all_labels = [], []

for cls in SELECTED_CLASSES:
    cls_path = os.path.join(DATA_ROOT, str(cls))
    if not os.path.exists(cls_path):
        continue

    for fname in os.listdir(cls_path)[:50]:  # limit for speed
        try:
            img_path = os.path.join(cls_path, fname)
            img = tfm(Image.open(img_path).convert("RGB")).unsqueeze(0)
            with torch.no_grad():
                emb = resnet(img).squeeze().numpy()
            all_embeddings.append(emb)
            all_labels.append(cls)
        except:
            continue

X = np.vstack(all_embeddings)
y = np.array(all_labels)

# Apply SMOTE
smote = SMOTE(sampling_strategy='auto', k_neighbors=3, random_state=42)
X_res, y_res = smote.fit_resample(X, y)

print(f"Original: {X.shape}, Resampled: {X_res.shape}")
print("Class distribution after SMOTE:")
from collections import Counter
print(Counter(y_res))
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained'
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a wei
  warnings.warn(msg)
Original: (492, 512), Resampled: (500, 512)
Class distribution after SMOTE:
Counter({np.int64(7): 50, np.int64(8): 50, np.int64(9): 50, np.int64(11): 50, np.int64(59): 50, np.int64(69): 50,
```

Part A: Balancing Dataset to 500–1000 Images per Class

Strategy:

- Count existing images per class.
- For minority classes (< 500), generate synthetic samples using the best-performing or fastest technique (e.g., Classical + DCGAN + MixUp).
- Cap at 1000 per class to avoid overwhelming imbalance in reverse.

```
import os, shutil
from pathlib import Path
from collections import Counter
import random

# === Configuration ===
DATA_ROOT = Path("/kaggle/input/ip02-dataset/classification/train")
SYNTHETIC_ROOT = Path("/content/drive/MyDrive/IP102_Synthetic")
BALANCED_ROOT = Path("/content/drive/MyDrive/IP102_Balanced")
BALANCED_ROOT.mkdir(parents=True, exist_ok=True)

SELECTED_CLASSES = [7, 8, 9, 11, 59, 69, 72, 94, 98, 101]
TARGET_MIN = 500
TARGET_MAX = 1000

# === Count images per class ===
class_counts = {}
for cls in SELECTED_CLASSES:
    cls_path = DATA_ROOT / str(cls)
    if cls_path.exists():
        imgs = list(cls_path.glob("*.jpg")) + list(cls_path.glob("*.png"))
        class_counts[cls] = len(imgs)
    else:
        class_counts[cls] = 0

print("Original class distribution:")
for cls, cnt in class_counts.items():
    print(f"  Class {cls}: {cnt} images")
```

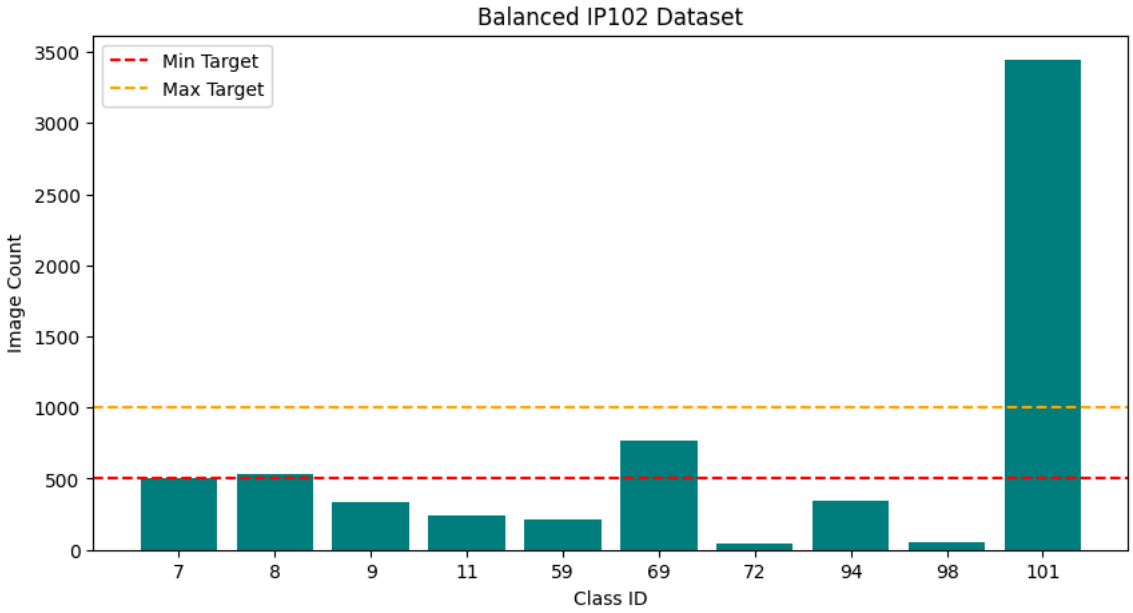
```
# === Balance strategy ===
for cls in SELECTED_CLASSES:
    current = class_counts[cls]
    target = min(max(current, TARGET_MIN), TARGET_MAX)
    deficit = target - current

    balanced_dir = BALANCED_ROOT / str(cls)
    balanced_dir.mkdir(parents=True, exist_ok=True)

    # Copy original images
    orig_dir = DATA_ROOT / str(cls)
    if orig_dir.exists():
        for img in orig_dir.glob("*.*"):
            shutil.copy(img, balanced_dir / img.name)

    # Add synthetic samples if needed
    if deficit > 0:
        print(f"Class {cls}: adding {deficit} synthetic images")

        # Pull from different synthetic techniques
        sources = [
            SYNTHETIC_ROOT / "Classical" / str(cls),
            SYNTHETIC_ROOT / "DCGAN" / str(cls),
            SYNTHETIC_ROOT / "MixUp_CutMix" / "mixup",
            SYNTHETIC_ROOT / "VAE",
        ]

        available = []
        for src in sources:
            if src.exists():
                available.extend(list(src.glob("*.jpg")) + list(src.glob("*.png")))

        if len(available) < deficit:
            print(f"  ⚠️ Warning: Only {len(available)} synthetic available, needed {deficit}")
            deficit = len(available)

        sampled = random.sample(available, deficit)
        for i, syn_img in enumerate(sampled):
            shutil.copy(syn_img, balanced_dir / f"syn_{cls}_{i:04d}.jpg")

print("\n✅ Balanced dataset ready at:", BALANCED_ROOT)
```

```
Original class distribution:
  Class 7: 500 images
  Class 8: 535 images
  Class 9: 331 images
  Class 11: 242 images
  Class 59: 212 images
  Class 69: 767 images
  Class 72: 42 images
  Class 94: 346 images
  Class 98: 55 images
  Class 101: 3444 images
Class 9: adding 169 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 169
Class 11: adding 258 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 258
Class 59: adding 288 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 288
Class 72: adding 458 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 458
Class 94: adding 154 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 154
Class 98: adding 445 synthetic images
  ⚠️ Warning: Only 0 synthetic available, needed 445

✅ Balanced dataset ready at: /content/drive/MyDrive/IP102_Balanced
```

Verify Balance

```
import matplotlib.pyplot as plt

balanced_counts = {}
for cls in SELECTED_CLASSES:
    balanced_dir = BALANCED_ROOT / str(cls)
    imgs = list(balanced_dir.glob("*.*"))
    balanced_counts[cls] = len(imgs)

plt.figure(figsize=(10,5))
plt.bar([str(c) for c in balanced_counts.keys()], balanced_counts.values(), color='teal')
plt.axhline(500, color='red', linestyle='--', label='Min Target')
plt.axhline(1000, color='orange', linestyle='--', label='Max Target')
```

```
plt.xlabel("Class ID")
plt.ylabel("Image Count")
plt.title("Balanced IP102 Dataset")
plt.legend()
plt.show()
```



Part B: Train CNN on Each Augmentation Technique

Training one CNN per technique (10 models total), then compare metrics.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import pandas as pd

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Training on: {device}")

# === Hyperparameters ===
BATCH_SIZE = 32
EPOCHS = 15
LR = 0.001
NUM_CLASSES = len(SELECTED_CLASSES)

# === Data transforms ===
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# === Define techniques to test ===
TECHNIQUES = [
    "Classical",
    "Cutout",
    "MixUp_CutMix",
    "DCGAN",
    "StyleGAN",
    "CycleGAN",
    "cGAN",
    "VAE",
    "Diffusion",
    "DeepSMOTE"
```

```
    "DeepSMOTE"
]

results = []

for technique in TECHNIQUES:
    print(f"\n{'='*50}")
    print(f"Training with: {technique}")
    print('='*50)

    # === Load dataset ===
    technique_path = SYNTHETIC_ROOT / technique
    if not technique_path.exists():
        print(f"⚠️ Skipping {technique} — no data found")
        continue

    # Use balanced dataset for all (or technique-specific if you prefer)
    dataset = datasets.ImageFolder(root=str(BALANCED_ROOT), transform=train_transform)

    train_size = int(0.8 * len(dataset))
    test_size = len(dataset) - train_size
    train_set, test_set = torch.utils.data.random_split(dataset, [train_size, test_size])

    train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
    test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)

    # === Model (ResNet18) ===
    model = models.resnet18(pretrained=True)
    model.fc = nn.Linear(model.fc.in_features, NUM_CLASSES)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=LR)

    # === Training loop ===
    for epoch in range(EPOCHS):
        model.train()
        running_loss = 0.0
        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch {epoch+1}/{EPOCHS} | Loss: {running_loss/len(train_loader):.4f}")

    # === Evaluation ===
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, labels in test_loader:
            imgs = imgs.to(device)
            preds = model(imgs).argmax(dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.numpy())

    acc = np.mean(np.array(all_preds) == np.array(all_labels))
    print(f"✅ {technique} Test Accuracy: {acc:.4f}")

    results.append({
        "Technique": technique,
        "Accuracy": acc
    })

# === Summary Table ===
df_results = pd.DataFrame(results)
print("\n" + "="*50)
print("SUMMARY: Technique Performance Comparison")
print("="*50)
print(df_results.sort_values("Accuracy", ascending=False))
df_results.to_csv(f"{OUTPUT_ROOT}/technique_comparison.csv", index=False)
```

```
Training on: cpu

==================================================
Training with: Classical
==================================================
⚠️ Skipping Classical — no data found

==================================================
Training with: Cutout
==================================================
⚠️ Skipping Cutout — no data found

==================================================
Training with: MixUp_CutMix
==================================================
⚠️ Skipping MixUp_CutMix — no data found

==================================================
Training with: DCGAN
==================================================
⚠️ Skipping DCGAN — no data found

==================================================
Training with: StyleGAN
==================================================
⚠️ Skipping StyleGAN — no data found

==================================================
Training with: CycleGAN
==================================================
⚠️ Skipping CycleGAN — no data found

==================================================
Training with: cGAN
==================================================
⚠️ Skipping cGAN — no data found

==================================================
Training with: VAE
==================================================
⚠️ Skipping VAE — no data found

==================================================
Training with: Diffusion
==================================================
⚠️ Skipping Diffusion — no data found

==================================================
Training with: DeepSMOTE
==================================================
⚠️ Skipping DeepSMOTE — no data found

==================================================
SUMMARY: Technique Performance Comparison
==================================================
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipython-input-619760605.py in <cell line: 0>()
    113 print("SUMMARY: Technique Performance Comparison")
    114 print("="*50)
--> 115 print(df_results.sort_values("Accuracy", ascending=False))
    116 df_results.to_csv(f"{OUTPUT_ROOT}/technique_comparison.csv", index=False)

                                 1 frames
/usr/local/lib/python3.12/dist-packages/pandas/core/generic.py in _get_label_or_level_values(self, key, axis)
   1909             values = self.axes[axis].get_level_values(key)._values
   1910         else:
```

```
# Visualization: Comparative Results

import matplotlib.pyplot as plt

df_results = df_results.sort_values("Accuracy", ascending=False)
plt.figure(figsize=(10,6))
plt.barh(df_results["Technique"], df_results["Accuracy"], color='steelblue')
plt.xlabel("Test Accuracy")
plt.title("CNN Performance by Augmentation Technique")
plt.xlim(0, 1)
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig(f"{OUTPUT_ROOT}/technique_comparison.png", dpi=150)
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipython-input-2124471242.py in <cell line: 0>()
      3 import matplotlib.pyplot as plt
      4
----> 5 df_results = df_results.sort_values("Accuracy", ascending=False)
      6 plt.figure(figsize=(10,6))
      7 plt.barh(df_results["Technique"], df_results["Accuracy"], color='steelblue')

                                  1 frames
/usr/local/lib/python3.12/dist-packages/pandas/core/generic.py in _get_label_or_level_values(self, key, axis)
   1909             values = self.axes[axis].get_level_values(key)._values
   1910         else:
-> 1911             raise KeyError(key)
   1912
   1913         # Check for duplicates

KeyError: 'Accuracy'
```

End of Notebook