Sorting and Recycling of Waste using CNN and Bayesian Optimization

- Sai Swayam Pradhan - 23BAI1432
- Preeti Yadav - 23BAI1456
- Abhay Tiwari - 23BAI1077


```
import numpy as np
import pandas as pd
%pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.23.2 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from pandas) (1.2
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\appdata\roaming\python\python311\site-packages (from pandas) (2.9.0
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2025
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from pandas) (20
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pand
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 23.2.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
!pip install torch torchvision scikit-learn numpy matplotlib seaborn tqdm optuna
```

```
Requirement already satisfied: torch in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (2.6.0)
Requirement already satisfied: torchvision in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (0.21.0)
Requirement already satisfied: scikit-learn in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (1.6.1)
Requirement already satisfied: numpy in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (1.26.0)
Requirement already satisfied: matplotlib in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (3.10.1)
Requirement already satisfied: seaborn in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (0.13.2)
Requirement already satisfied: tqdm in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (4.67.1)
Requirement already satisfied: optuna in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (4.2.1)
Requirement already satisfied: filelock in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torch) (3.17.0)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\hp\appdata\roaming\python\python311\site-packages (from torch) (4.1
Requirement already satisfied: networkx in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torch) (2025.2.0)
Requirement already satisfied: sympy==1.13.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torch) (1.13
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from sympy==
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from torc
Requirement already satisfied: scipy>=1.6.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn)
Requirement already satisfied: joblib>=1.2.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from sciki
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from matplotli
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from matplotlib) (
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from matplotl
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from matplotl
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\roaming\python\python311\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from matplotli
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\appdata\roaming\python\python311\site-packages (from matplotlib) (2.9
Requirement already satisfied: pandas>=1.2 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from seaborn) (2.2.
Requirement already satisfied: colorama in c:\users\hp\appdata\roaming\python\python311\site-packages (from tqdm) (0.4.6)
Requirement already satisfied: alembic>=1.5.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from optuna) (1.
Requirement already satisfied: colorlog in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from optuna) (6.9.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from optuna)
Requirement already satisfied: PyYAML in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from optuna) (6.0.2)
Requirement already satisfied: Mako in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from alembic>=1.5.0->optun
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from pandas>=1.2->
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from pandas>=1.2
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplo
Requirement already satisfied: greenlet!=0.4.17 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from sqlalchem
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from jinja2->to

[notice] A new release of pip is available: 23.2.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.models as models
import torchvision.datasets as datasets
from torch.utils.data import DataLoader
from sklearn.utils.class_weight import compute_class_weight
```

```
from sklearn.metrics import (
    classification_report, confusion_matrix, f1_score,
    precision_recall_curve, average_precision_score,
    roc_curve, roc_auc_score, precision_score, recall_score
)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from time import time
from tqdm import tqdm
import optuna  # For Bayesian optimization
```

```
c:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update j
  from .autonotebook import tqdm as notebook_tqdm
```

```
# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.manual_seed(42)
np.random.seed(42)
print(device)
```

```
cpu
```

```
%pip install os
```

```
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement os (from versions: none)
ERROR: No matching distribution found for os

[notice] A new release of pip is available: 23.2.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
# Dataset Paths
import os
dataset_path = "DATASET"
train_dir = os.path.join(dataset_path, "TRAIN")
val_dir = os.path.join(dataset_path, "TEST")  # Using TEST as validation set
```

```
# Data Transformations
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to 224x224
    transforms.RandomRotation(30),  # Random rotation between -30 to +30 degrees
    transforms.RandomHorizontalFlip(p=0.5),  # Flip image with 50% probability
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # Vary colors
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1)),  # Random affine transformations
    transforms.ToTensor(),  # Convert image to PyTorch tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize using ImageNet stats
])

val_transforms = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to 224x224
    transforms.ToTensor(),  # Convert image to tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize
])
```

```
# Load Data
train_dataset = datasets.ImageFolder(train_dir, transform=train_transforms)
val_dataset = datasets.ImageFolder(val_dir, transform=val_transforms)
```

Image Visualization

```
# Helps identify class imbalance.
import matplotlib.pyplot as plt
import seaborn as sns

# Get class names and counts
class_counts = [len(os.listdir(os.path.join(train_dir, cls))) for cls in train_dataset.classes]

# Plot the class distribution
plt.figure(figsize=(10, 5))
```

```
sns.barplot(x=train_dataset.classes, y=class_counts, palette="viridis")
plt.xticks(rotation=45)
plt.xlabel("Class")
plt.ylabel("Number of Images")
plt.title("Class Distribution in Training Set")
plt.show()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_30584\3839377508.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x=train_dataset.classes, y=class_counts, palette="viridis")
```


```
# Displays a few random images with their labels.
import torchvision.utils as vutils

# Function to display sample images
def show_images(dataset, num_images=10):
    fig, axes = plt.subplots(1, num_images, figsize=(20, 5))
    indices = np.random.randint(0, len(dataset), size=num_images)

    for i, idx in enumerate(indices):
        img, label = dataset[idx]
        img = img.permute(1, 2, 0).numpy()  # Convert tensor to numpy
        img = img * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]  # Denormalization
        img = np.clip(img, 0, 1)  # Clip values between 0 and 1

        axes[i].imshow(img)
        axes[i].set_title(f"Class: {train_dataset.classes[label]}")
        axes[i].axis("off")

    plt.show()

# Show random images from the training set
show_images(train_dataset)
```


```
import os
import numpy as np
import matplotlib.pyplot as plt
```

```
from PIL import Image

train_dir = r"C:\Users\hp\OneDrive\Desktop\AI WASTE MANGE\DATASET\TRAIN"  # Update this path if needed

image_shapes = []
for img_name in os.listdir(train_dir):
    img_path = os.path.join(train_dir, img_name)
    if os.path.isdir(img_path):
        continue

    try:
        with Image.open(img_path) as img:
            image_shapes.append(img.size)
    except Exception as e:
        print(f"Skipping {img_name}: {e}")
# Convert to NumPy array (only if images exist)
if image_shapes:
    image_shapes = np.array(image_shapes)

    # Plot image dimensions
    plt.figure(figsize=(10, 5))
    plt.scatter(image_shapes[:, 0], image_shapes[:, 1], alpha=0.5)
    plt.xlabel("Width")
    plt.ylabel("Height")
    plt.title("Image Size Distribution")
    plt.show()
else:
    print("No valid images found in the directory.")
```

```
No valid images found in the directory.
```

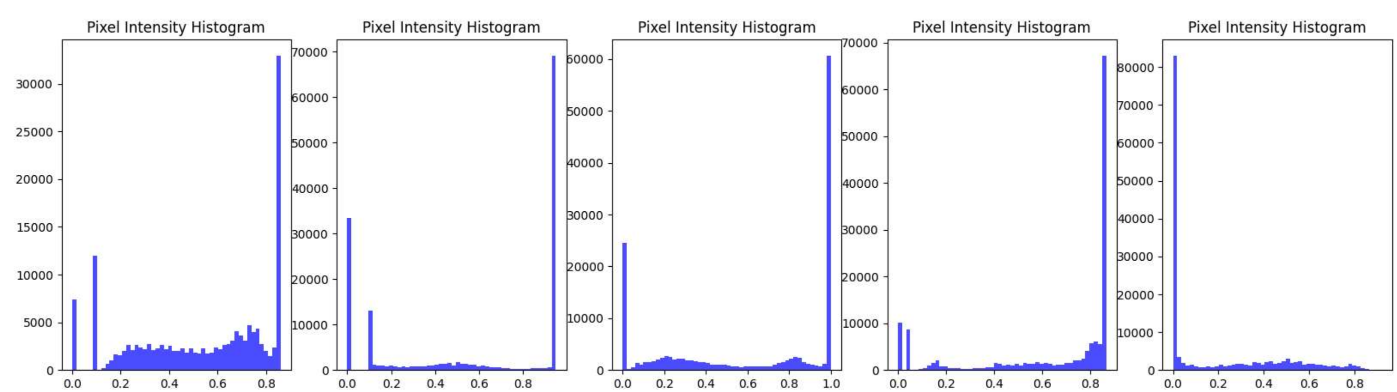
```
# Function to plot pixel intensity distribution
def plot_pixel_distribution(dataset, num_samples=5):
    fig, axes = plt.subplots(1, num_samples, figsize=(20, 5))

    for i in range(num_samples):
        img, _ = dataset[i]
        img = img.permute(1, 2, 0).numpy()  # Convert tensor to NumPy
        img = img * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]  # Denormalization
        img = np.clip(img, 0, 1)

        axes[i].hist(img.ravel(), bins=50, color="blue", alpha=0.7)
        axes[i].set_title("Pixel Intensity Histogram")

    plt.show()

# Show pixel distributions
plot_pixel_distribution(train_dataset)
```



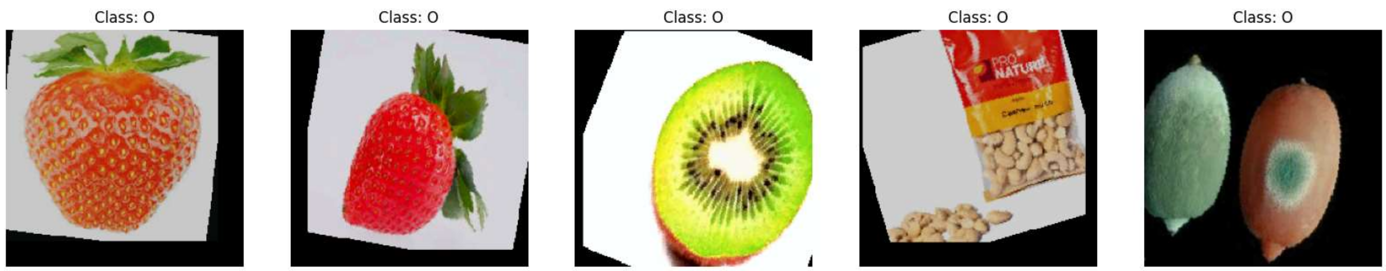
```
#Ensures that augmentations are applied correctly.
def visualize_augmentations(dataset, num_images=5):
    fig, axes = plt.subplots(1, num_images, figsize=(20, 5))

    for i in range(num_images):
        img, label = dataset[i]
        img = img.permute(1, 2, 0).numpy()  # Convert tensor to NumPy
        img = img * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]  # Denormalization
        img = np.clip(img, 0, 1)

        axes[i].imshow(img)
        axes[i].set_title(f"Class: {dataset.classes[label]}")
        axes[i].axis("off")

    plt.show()

# Show augmented images
visualize_augmentations(train_dataset)
```




```
batch_size = 32  # Adjust batch size to avoid memory issues
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=True)
```

```
import optuna

# Define the objective function for optimization
def objective(trial):
    # Search space for hyperparameters
    lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)
    batch_size = trial.suggest_categorical("batch_size", [16, 32, 64])
    dropout_rate = trial.suggest_float("dropout_rate", 0.2, 0.5)
    num_filters = trial.suggest_categorical("num_filters", [32, 64, 128])
    optimizer_name = trial.suggest_categorical("optimizer", ["Adam", "SGD", "RMSprop"])

    # Data loaders (Re-create with different batch size)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=True)

    # Define a new model with trial parameters
    class OptimizedCNN(nn.Module):
        def __init__(self, num_classes):
            super(OptimizedCNN, self).__init__()
            self.conv_layers = nn.Sequential(
                nn.Conv2d(3, num_filters, kernel_size=3, stride=1, padding=1),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),

                nn.Conv2d(num_filters, num_filters * 2, kernel_size=3, stride=1, padding=1),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),

                nn.Conv2d(num_filters * 2, num_filters * 4, kernel_size=3, stride=1, padding=1),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2)
            )
            self.fc_layers = nn.Sequential(
                nn.Flatten(),
```

```
                nn.Linear(num_filters * 4 * 28 * 28, 512),
                nn.ReLU(),
                nn.Dropout(dropout_rate),
                nn.Linear(512, num_classes)
            )

        def forward(self, x):
            x = self.conv_layers(x)
            x = self.fc_layers(x)
            return x

    # Initialize model
    model = OptimizedCNN(len(train_dataset.classes)).to(device)

    # Define loss function
    criterion = nn.CrossEntropyLoss(weight=class_weights)

    # Choose optimizer
    if optimizer_name == "Adam":
        optimizer = optim.Adam(model.parameters(), lr=lr)
    elif optimizer_name == "SGD":
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
    else:
        optimizer = optim.RMSprop(model.parameters(), lr=lr)

    # Train the model for a few epochs (to speed up Optuna)
    for epoch in range(3):  # Fewer epochs for efficiency
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

    # Evaluate the model
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return accuracy  # Optuna will try to maximize accuracy
```

```
from sklearn.utils.class_weight import compute_class_weight
import torch

# Get class labels from dataset
labels = [label for _, label in train_dataset]

# Compute class weights
class_weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(labels),
    y=labels
)

# Convert to tensor for PyTorch
class_weights = torch.tensor(class_weights, dtype=torch.float32).to(device)

# Now define the loss function
criterion = nn.CrossEntropyLoss(weight=class_weights)
```

```
# Create a study and optimize
study = optuna.create_study(direction="maximize")  # We want to maximize accuracy
study.optimize(objective, n_trials=20)  # Run 20 trials to find the best hyperparameters
# Print the best parameters
```

```
print("Best Hyperparameters:", study.best_params)
```

Show hidden output

```
# Compute Class Weights
train_targets = np.array(train_dataset.targets)
class_weights = compute_class_weight('balanced', classes=np.unique(train_targets), y=train_targets)
class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)
```

Show hidden output

```
import torch
import torch.nn as nn

class WasteClassifierCNN(nn.Module):
    def __init__(self, num_classes):
        super(WasteClassifierCNN, self).__init__()

        # Convolutional layers for feature extraction
        self.conv_layers = nn.Sequential(
            # First convolutional block: 3 input channels (RGB), 32 output channels, 3x3 kernel, stride 1, padding 1
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),  # ReLU activation function
            nn.MaxPool2d(kernel_size=2, stride=2),  # Max pooling to reduce spatial dimensions

            # Second convolutional block: 32 input channels, 64 output channels, 3x3 kernel, stride 1, padding 1
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # Third convolutional block: 64 input channels, 128 output channels, 3x3 kernel, stride 1, padding 1
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        # Fully connected layers for classification
        self.fc_layers = nn.Sequential(
            nn.Flatten(),  # Flatten the feature maps into a 1D tensor
            nn.Linear(128 * 28 * 28, 512),
            nn.ReLU(),
            nn.Dropout(0.5),  # Dropout for regularization
            nn.Linear(512, num_classes)  # Output layer: 512 input features, num_classes output features
        )

    def forward(self, x):
        x = self.conv_layers(x)  # Pass input through convolutional layers
        x = self.fc_layers(x)  # Pass output through fully connected layers
        return x


# Initialize Model
num_classes = len(train_dataset.classes)
model = WasteClassifierCNN(num_classes).to(device)

# Define Loss and Optimizer
criterion = nn.CrossEntropyLoss(weight=class_weights)
optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

```
import torch
from tqdm import tqdm

def train_epoch(model, train_loader, criterion, optimizer, epoch, device, batch_size):
    """
    Trains the model for one epoch.
        model (nn.Module): The neural network model to train.
        train_loader (DataLoader): DataLoader for the training dataset.
        criterion (nn.Module): Loss function (e.g., CrossEntropyLoss).
        optimizer (Optimizer): Optimizer (e.g., Adam).
        epoch (int): Current epoch number.
        device (str): Device to use (e.g., 'cuda' or 'cpu').
        batch_size (int): Size of batches used in the training loop.
    """
    model.train()  # Set the model to training mode
```

```
    running_loss = 0.0
    correct, total = 0, 0

    progress_bar = tqdm(train_loader, desc=f'Epoch {epoch+1}')

    for inputs, labels in progress_bar:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

        # Update the progress bar with current loss and accuracy
        progress_bar.set_postfix({'loss': f'{running_loss/(total/batch_size):.3f}', 'acc': f'{100.*correct/total:.2f}%'})

    # Calculate average loss and accuracy for the epoch
    average_loss = running_loss / len(train_loader)
    epoch_accuracy = 100. * correct / total

    return average_loss, epoch_accuracy
```

```
import torch
import numpy as np
from tqdm import tqdm

def evaluate_model(model, val_loader, criterion):
    """Evaluates the model on the validation set."""
    model.eval()  # Set to evaluation mode
    running_loss = 0.0
    all_preds, all_labels, all_probs = [], [], []

    with torch.no_grad(): # Disable gradient calculation
        for inputs, labels in tqdm(val_loader, desc='Validation'):
            inputs, labels = inputs.to(device), labels.to(device) # Move data to device
            outputs = model(inputs) # Forward pass
            loss = criterion(outputs, labels) # Calculate loss
            running_loss += loss.item() # Accumulate loss

            probs = torch.softmax(outputs, dim=1) # Get probabilities
            _, preds = torch.max(outputs, 1) # Get predictions

            all_preds.extend(preds.cpu().numpy()) # Store predictions
            all_labels.extend(labels.cpu().numpy()) # Store true labels
            all_probs.extend(probs[:, 1].cpu().numpy()) # Store probabilities (binary case)

    return running_loss / len(val_loader), np.array(all_preds), np.array(all_labels), np.array(all_probs) # Return results
```

```
# Training Loop
num_epochs = 10
best_val_loss = float('inf')

for epoch in range(num_epochs):
    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, epoch)
    val_loss, val_preds, val_labels, val_probs = evaluate_model(model, val_loader, criterion)

    print(f"Epoch {epoch+1} - Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%")
    print(f"Validation Loss: {val_loss:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 'best_custom_cnn.pth')
        print("New best model saved!")
```

Show hidden output

```
# Evaluation & Metrics Plotting
def plot_metrics(all_labels, all_preds, all_probs, classes):
    plt.figure(figsize=(10, 8))
    conf_matrix = confusion_matrix(all_labels, all_preds)
```

```
    conf_matrix = confusion_matrix(all_labels, all_preds)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()

    plt.figure(figsize=(10, 8))
    fpr, tpr, _ = roc_curve(all_labels, all_probs)
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc_score(all_labels, all_probs):.3f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()


def evaluate_model(model, val_loader, criterion):
    model.eval()
    running_loss = 0.0
    all_preds = []
    all_labels = []
    all_probs = []

    print("\nEvaluating model...")
    with torch.no_grad():
        for inputs, labels in tqdm(val_loader, desc='Validation'):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item()

            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            all_probs.extend(probs[:, 1].cpu().numpy())  # Probability of positive class

    val_loss = running_loss / len(val_loader)
    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)
    all_probs = np.array(all_probs)

    return val_loss, all_preds, all_labels, all_probs


# Final Evaluation
val_loss, final_preds, final_labels, final_probs = evaluate_model(model, val_loader, criterion)

print("\nFinal Classification Report:")
print(classification_report(final_labels, final_preds, target_names=train_dataset.classes, digits=4))

plot_metrics(final_labels, final_preds, final_probs, train_dataset.classes)


import pickle
# Save the model to a file using pickle
with open('my_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```