conv-recur_artist

March 25, 2025

[1]:
```
import torch
import pandas as pd
import numpy as np
from tqdm import tqdm
import wandb
```

[2]:
```
wandb.init(entity="ameyar3103-iiit-hyderabad",project="recurrent_conv_art",␣
 ↪config={
    "epochs": 5,
    "batch_size": 4,
    "learning_rate": 0.001,
    "model": "RecurrentCNN"
})
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: ameyar3103
(ameyar3103-iiit-hyderabad) to https://api.wandb.ai. Use
`wandb login --relogin` to force relogin
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[2]:
```
<wandb.sdk.wandb_run.Run at 0x76f213160f70>
```

0.1 Data loading

[3]:
```
df_train = pd.read_csv('wikiart_csv/artist_train.csv',header=None,␣
 ↪names=["image_path", "artist_id"])
df_val = pd.read_csv('wikiart_csv/artist_val.csv',header=None,␣
 ↪names=["image_path", "artist_id"])
```

[4]:
```
# get the number of classes
num_classes = 23 # from artist_class.txt
```

[5]:
```
# Gather input data
train_images = df_train['image_path'].values
train_labels = df_train['artist_id'].values

val_images = df_val['image_path'].values
val_labels = df_val['artist_id'].values
```

[6]:
```
from torchvision import transforms
import cv2
```

0.2 Preprocess data and create test and train dataset

[7]:
```
# create test and train dataset for dataloader

def get_image(image_path,image_size=224):
    try:
        img = cv2.imread('./wikiart/' + image_path)
        if img is None:
            raise ValueError(f"Image not loaded: ./wikiart/{image_path}")
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        h, w, _ = img.shape
        scale = 256 / min(h, w)
        new_w = int(w * scale)
        new_h = int(h * scale)
        img_resized = cv2.resize(img, (new_w, new_h))
        start_x = (new_w - image_size) // 2
        start_y = (new_h - image_size) // 2
        img_cropped = img_resized[start_y:start_y+image_size, start_x:
 ↪start_x+image_size]
        img_cropped = img_cropped.astype(np.float32) / 255.0
        img_tensor = torch.from_numpy(img_cropped).permute(2, 0, 1)
        mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)
        std  = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1)
        img_tensor = (img_tensor - mean) / std
        return img_tensor
    except Exception as e:
        print(f"Error processing {image_path}: {e}")
        return torch.zeros(3, image_size, image_size)

class WikiArtDataset(torch.utils.data.Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels
```

```
    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        # image_vectors = []
        # for image in self.images:
        #     image_emb = get_image(image)
        #     image_vectors.append(image_emb)
        # image = torch.stack(image_vectors)
        image = self.images[idx]
        # label should be a one-hot encoded vector
        label = torch.zeros(num_classes)
        label[self.labels[idx]] = 1

        return image, label

train_dataset = WikiArtDataset(train_images, train_labels)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,␣
 ↪shuffle=True)
val_dataset = WikiArtDataset(val_images, val_labels)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=64,␣
 ↪shuffle=False)

for i, (images, labels) in enumerate(train_loader):
    print(images)
    print(labels)
    break
```

```
('Impressionism/claude-monet_the-japanese-bridge-2-1924.jpg',
'Symbolism/nicholas-roerich_himalayas-1947-1.jpg', 'Impressionism/pierre-
auguste-renoir_bather-drying-her-leg.jpg', 'Symbolism/salvador-dali_familia-
ruth-moabitidis-1964.jpg', 'Romanticism/ivan-aivazovsky_the-bay-of-naples-by-
moonlight-1892.jpg', 'Impressionism/pierre-auguste-renoir_umbrellas-1886.jpg',
'Impressionism/edgar-degas_two-harlequins-1886.jpg',
'Naive_Art_Primitivism/martiros-saryan_illustration-to-armenian-folk-
tales-1937-9.jpg', 'Realism/pyotr-konchalovsky_the-roses-by-the-
window-1953.jpg', 'Realism/ivan-shishkin_covert-1894.jpg',
'Impressionism/childe-hassam_lower-manhattan-aka-broad-and-wall-streets.jpg',
'Impressionism/pierre-auguste-renoir_christine-lerolle-embroidering-1897.jpg',
'Romanticism/gustave-dore_schismatics-mahomet.jpg', 'Realism/ilya-
repin_moonlight-1896.jpg', 'Impressionism/edgar-degas_achille-de-gas-1872.jpg',
'Cubism/pablo-picasso_still-life.jpg', 'Realism/pyotr-konchalovsky_skating-rink-
dynamo-1948.jpg', 'Romanticism/gustave-dore_don-quixote-94.jpg',
'Impressionism/pierre-auguste-renoir_lemons-1912.jpg', 'Realism/ivan-
shishkin_winte-1890.jpg', 'Romanticism/gustave-dore_don-quixote-138.jpg',
'Realism/ivan-shishkin_willows-lit-up-by-the-sun.jpg', 'Realism/camille-
pissarro_entering-a-village.jpg', 'Realism/pyotr-konchalovsky_still-life-rowan-
```

```
on-blue-1947.jpg', 'Impressionism/camille-pissarro_boulevard-montmartre-
afternoon-in-the-rain-1897.jpg', 'Impressionism/edgar-degas_combing-the-
hair.jpg', 'Pointillism/salvador-dali_dawn-noon-sunset-and-dusk.jpg',
'Impressionism/pierre-auguste-renoir_vase-of-peonies.jpg',
'Impressionism/childe-hassam_morning-old-lyme.jpg', 'Impressionism/eugene-
boudin_camaret-three-masters-anchored-in-the-harbor-1873.jpg',
'Impressionism/claude-monet_water-lilies-35.jpg', 'Naive_Art_Primitivism/marc-
chagall_clowns-musicians-1980.jpg', 'Expressionism/pablo-picasso_portrait-of-
child-1951.jpg', 'Impressionism/claude-monet_leicester-square-at-night.jpg',
'Symbolism/salvador-dali_beatrice.jpg', 'Symbolism/nicholas-roerich_blessed-
soul-bhagavan-sri-ramakrishna-1924.jpg', 'Impressionism/pierre-auguste-
renoir_nude-study-for-the-large-bathers-1887.jpg', 'Art_Nouveau_Modern/raphael-
kirchner_boys-and-girls-at-sea.jpg', 'Impressionism/pierre-auguste-
renoir_landscape-13.jpg', 'Impressionism/pierre-auguste-renoir_young-woman-in-a-
straw-hat.jpg', 'Naive_Art_Primitivism/marc-chagall_joseph-being-seventeen-
years-old-goes-with-his-brothers-and-the-flocks-genesis-xxxvii-2.jpg',
'Symbolism/martiros-saryan_aragats-1922.jpg', 'Symbolism/salvador-dali_iosephet-
fratres-in-aegypto-1967.jpg', 'Realism/ivan-shishkin_bee-families-1884.jpg',
'Realism/ilya-repin_calvary-crucifixion-study-1869.jpg',
'Northern_Renaissance/albrecht-durer_apollo-with-the-solar-disc.jpg',
'Post_Impressionism/vincent-van-gogh_madame-roulin-rocking-the-cradle-a-
lullaby-1889-1.jpg', 'Art_Nouveau_Modern/boris-kustodiev_waiter-1920.jpg',
'Baroque/rembrandt_self-portrait-in-oriental-attire-with-poodle-1631.jpg',
'Realism/vincent-van-gogh_sand-diggers-in-dekkersduin-near-the-
hague-2-1883.jpg', 'Romanticism/gustave-dore_a-voyage-to-the-moon.jpg',
'Romanticism/ivan-aivazovsky_battle-of-steamship-vesta-and-turkish-
ironclad-1877.jpg', 'Art_Nouveau_Modern/raphael-kirchner_marionettes-1.jpg',
'Impressionism/john-singer-sargent_man-seated-by-a-stream-1912.jpg',
'Baroque/rembrandt_the-stoning-of-st-stephen-1635.jpg',
'Art_Nouveau_Modern/raphael-kirchner_small-greek-heads-1.jpg', 'Realism/ivan-
shishkin_forest-1897.jpg', 'Art_Nouveau_Modern/nicholas-
roerich_plafond-1913.jpg', 'Realism/boris-kustodiev_church-parade-of-the-
finlandsky-guard-regiment-december-12-1905-1906.jpg', 'Expressionism/pablo-
picasso_el-tinen-1906.jpg', 'Impressionism/claude-monet_the-la-rue-bavolle-at-
honfleur-2.jpg', 'Impressionism/camille-pissarro_the-countryside-in-the-
vicinity-of-conflans-saint-honorine-1874.jpg', 'Symbolism/nicholas-
roerich_charaka.jpg', 'Baroque/rembrandt_a-young-man-at-a-table-possibly-
govaert-flinck-1660.jpg')
tensor([[0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.],
        …,
        [0., 0., 1.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 1., 0., 0.]])
```

0.3 Model architecture definition

[ ]:
```
import torch.nn as nn
import torch.nn.functional as F

class RecurrentCNN(nn.Module):
    def __init__(self, num_classes, lstm_hidden_size=256, dropout_prob=0.5):
        super(RecurrentCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.adaptive_pool = nn.AdaptiveAvgPool2d((14, 56))
        self.lstm_input_size = 64 * 56
        self.lstm_hidden_size = lstm_hidden_size
        self.lstm = nn.LSTM(input_size=self.lstm_input_size,␣
 ↪hidden_size=lstm_hidden_size,
                            batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(2 * lstm_hidden_size, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = self.adaptive_pool(x)
        x = x.permute(0, 2, 1, 3).contiguous()
        batch_size, seq_len, channels, width = x.shape
        x = x.view(batch_size, seq_len, channels * width)
        lstm_out, _ = self.lstm(x)
        x = lstm_out.mean(dim=1)
        x = self.dropout(x)
        x = self.fc(x)
        return x

model = RecurrentCNN(num_classes)
model.to('cuda')

import torch.optim as optim

wandb.watch(model, log="all")
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

0.4 Model training and saving the model,optimizer weights (to use when needed)

[9]:
```
# Train the model
num_epochs = 5

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    train_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")
    for image_paths, labels in train_bar:
        image_tensors = torch.stack([get_image(image_path) for image_path in␣
 ↪image_paths])
        images = image_tensors.to('cuda')
        labels = labels.to('cuda')

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        train_bar.set_postfix(loss=loss.item())

    avg_train_loss = running_loss / len(train_loader)
    wandb.log({"epoch": epoch+1, "train_loss": avg_train_loss})

    # Validation Loop
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        val_bar = tqdm(val_loader, desc="Validation")
        for image_paths, labels in val_bar:
            image_tensors = torch.stack([get_image(image_path) for image_path␣
 ↪in image_paths])
            image_tensors = image_tensors.to('cuda')
            labels = labels.to('cuda')
            outputs = model(image_tensors)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
```

```
            total += labels.size(0)
            correct += (predicted == labels.argmax(dim=1)).sum().item()
            val_bar.set_postfix(loss=loss.item())

    avg_val_loss = val_loss / len(val_loader)
    val_accuracy = 100 * correct / total
    wandb.log({"val_loss": avg_val_loss, "val_accuracy": val_accuracy})
    print(f"Epoch {epoch+1}/{num_epochs} - Train Loss: {avg_train_loss:.4f},␣
↪Val Loss: {avg_val_loss:.4f}, Val Accuracy: {val_accuracy:.2f}%")
```

```
Epoch 1/5:  19%|          | 39/209 [00:35<02:31,  1.12it/s, loss=2.8] Corrupt
JPEG data: premature end of data segment
Epoch 1/5:  72%|       | 150/209 [02:19<00:56,  1.04it/s, loss=2.54]Corrupt
JPEG data: bad Huffman code
Epoch 1/5: 100%|      | 209/209 [03:11<00:00,  1.09it/s, loss=2.57]
Validation: 100%|      | 90/90 [01:17<00:00,  1.17it/s, loss=3.11]

Epoch 1/5 - Train Loss: 2.5810, Val Loss: 2.4019, Val Accuracy: 28.90%

Epoch 2/5:  63%|       | 132/209 [02:03<01:08,  1.13it/s, loss=2.36]Corrupt
JPEG data: bad Huffman code
Epoch 2/5:  71%|       | 148/209 [02:19<00:59,  1.02it/s, loss=2.01]Corrupt
JPEG data: premature end of data segment
Epoch 2/5: 100%|      | 209/209 [03:15<00:00,  1.07it/s, loss=2.17]
Validation: 100%|      | 90/90 [01:15<00:00,  1.19it/s, loss=2.41]

Epoch 2/5 - Train Loss: 2.2534, Val Loss: 2.1157, Val Accuracy: 38.00%

Epoch 3/5:  60%|       | 126/209 [01:59<01:19,  1.04it/s, loss=2.38]Corrupt
JPEG data: bad Huffman code
Epoch 3/5:  95%|      | 199/209 [03:04<00:09,  1.08it/s, loss=2.22]Corrupt
JPEG data: premature end of data segment
Epoch 3/5: 100%|      | 209/209 [03:13<00:00,  1.08it/s, loss=2.23]
Validation: 100%|      | 90/90 [01:14<00:00,  1.21it/s, loss=2.54]

Epoch 3/5 - Train Loss: 2.0654, Val Loss: 2.0065, Val Accuracy: 40.24%

Epoch 4/5:  25%|         | 52/209 [00:47<02:15,  1.16it/s, loss=2.26]Corrupt
JPEG data: bad Huffman code
Epoch 4/5:  71%|       | 149/209 [02:14<00:56,  1.06it/s, loss=1.99]Corrupt
JPEG data: premature end of data segment
Epoch 4/5: 100%|      | 209/209 [03:07<00:00,  1.12it/s, loss=2.26]
Validation: 100%|      | 90/90 [01:11<00:00,  1.25it/s, loss=1.92]

Epoch 4/5 - Train Loss: 1.9179, Val Loss: 1.9156, Val Accuracy: 43.02%

Epoch 5/5:  19%|          | 39/209 [00:34<02:39,  1.07it/s, loss=1.77]Corrupt
JPEG data: premature end of data segment
Epoch 5/5:  42%|        | 88/209 [01:18<01:40,  1.21it/s, loss=1.69]Corrupt
JPEG data: bad Huffman code
Epoch 5/5: 100%|      | 209/209 [03:05<00:00,  1.13it/s, loss=2.27]
```

```
Validation: 100%|      | 90/90 [01:11<00:00,  1.27it/s, loss=1.99]
Epoch 5/5 - Train Loss: 1.7949, Val Loss: 1.8356, Val Accuracy: 45.50%
```

[ ]:
```python
num_epochs = 15

for epoch in range(5,5+num_epochs):
    model.train()
    running_loss = 0.0
    train_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")
    for image_paths, labels in train_bar:
        image_tensors = torch.stack([get_image(image_path) for image_path in␣
 ↪image_paths])
        images = image_tensors.to('cuda')
        labels = labels.to('cuda')

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        train_bar.set_postfix(loss=loss.item())

    avg_train_loss = running_loss / len(train_loader)
    wandb.log({"epoch": epoch+1, "train_loss": avg_train_loss})

    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        val_bar = tqdm(val_loader, desc="Validation")
        for image_paths, labels in val_bar:
            image_tensors = torch.stack([get_image(image_path) for image_path␣
 ↪in image_paths])
            image_tensors = image_tensors.to('cuda')
            labels = labels.to('cuda')
            outputs = model(image_tensors)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
```

```
            total += labels.size(0)
            correct += (predicted == labels.argmax(dim=1)).sum().item()
            val_bar.set_postfix(loss=loss.item())

    avg_val_loss = val_loss / len(val_loader)
    val_accuracy = 100 * correct / total
    wandb.log({"val_loss": avg_val_loss, "val_accuracy": val_accuracy})
    print(f"Epoch {epoch+1}/{num_epochs} - Train Loss: {avg_train_loss:.4f},␣
↪Val Loss: {avg_val_loss:.4f}, Val Accuracy: {val_accuracy:.2f}%")
    if(epoch%5==0):
        torch.save(model.state_dict(), f"recurrent_cnn_epoch_{epoch+1}.pth")
        torch.save(optimizer.state_dict(),␣
↪f"recurrent_cnn_optimizer_epoch_{epoch+1}.pth")
```

```
Epoch 6/15:  54%|       | 113/209 [01:44<01:15,  1.26it/s, loss=1.75]Corrupt
JPEG data: bad Huffman code
Epoch 6/15:  57%|       | 119/209 [01:50<01:22,  1.10it/s, loss=1.79]Corrupt
JPEG data: premature end of data segment
Epoch 6/15: 100%|      | 209/209 [03:20<00:00,  1.04it/s, loss=1.72]
Validation: 100%|      | 90/90 [01:22<00:00,  1.09it/s, loss=2.68]

Epoch 6/15 - Train Loss: 1.6917, Val Loss: 1.7797, Val Accuracy: 47.65%

Epoch 7/15:  39%|        | 81/209 [01:14<01:53,  1.12it/s, loss=1.78]Corrupt
JPEG data: bad Huffman code
Epoch 7/15:  54%|       | 113/209 [01:45<01:27,  1.10it/s, loss=1.34]Corrupt
JPEG data: premature end of data segment
Epoch 7/15: 100%|      | 209/209 [03:12<00:00,  1.08it/s, loss=2.08]
Validation: 100%|      | 90/90 [01:15<00:00,  1.19it/s, loss=1.89]

Epoch 7/15 - Train Loss: 1.5727, Val Loss: 1.7408, Val Accuracy: 47.55%

Epoch 8/15:   5%|         | 11/209 [00:10<03:10,  1.04it/s, loss=1.51]Corrupt
JPEG data: bad Huffman code
Epoch 8/15:  46%|        | 96/209 [01:29<01:47,  1.05it/s, loss=1.49]Corrupt
JPEG data: premature end of data segment
Epoch 8/15: 100%|      | 209/209 [03:16<00:00,  1.07it/s, loss=1.48]
Validation: 100%|      | 90/90 [01:17<00:00,  1.16it/s, loss=2.06]

Epoch 8/15 - Train Loss: 1.4398, Val Loss: 1.6948, Val Accuracy: 49.89%

Epoch 9/15:   3%|         | 6/209 [00:06<03:23,  1.00s/it, loss=1.1] Corrupt
JPEG data: premature end of data segment
Epoch 9/15:  17%|        | 35/209 [00:33<02:43,  1.06it/s, loss=1.22] Corrupt
JPEG data: bad Huffman code
Epoch 9/15: 100%|      | 209/209 [03:23<00:00,  1.03it/s, loss=1.21]
Validation: 100%|      | 90/90 [01:28<00:00,  1.01it/s, loss=1.85]

Epoch 9/15 - Train Loss: 1.2744, Val Loss: 1.6817, Val Accuracy: 50.37%

Epoch 10/15:  49%|       | 103/209 [01:44<01:34,  1.12it/s,
```

```
loss=0.952]Corrupt JPEG data: bad Huffman code
Epoch 10/15:  64%|       | 133/209 [02:11<01:16,  1.00s/it, loss=1]
Corrupt JPEG data: premature end of data segment
Epoch 10/15: 100%|      | 209/209 [03:29<00:00,  1.00s/it, loss=1.19]
Validation: 100%|      | 90/90 [01:23<00:00,  1.08it/s, loss=1.57]

Epoch 10/15 - Train Loss: 1.1044, Val Loss: 1.6716, Val Accuracy: 51.14%

Epoch 11/15:  25%|         | 53/209 [00:53<03:19,  1.28s/it, loss=0.678]Corrupt
JPEG data: bad Huffman code
Epoch 11/15:  32%|        | 67/209 [01:07<02:29,  1.06s/it, loss=1.3]  Corrupt
JPEG data: premature end of data segment
Epoch 11/15: 100%|      | 209/209 [03:32<00:00,  1.02s/it, loss=0.551]
Validation: 100%|      | 90/90 [01:19<00:00,  1.14it/s, loss=2.34]

Epoch 11/15 - Train Loss: 0.9214, Val Loss: 1.7216, Val Accuracy: 51.52%

Epoch 12/15:   2%|          | 5/209 [00:04<02:54,  1.17it/s, loss=0.762]Corrupt
JPEG data: bad Huffman code
Epoch 12/15:  91%|      | 191/209 [02:45<00:13,  1.34it/s,
loss=0.714]Corrupt JPEG data: premature end of data segment
Epoch 12/15: 100%|      | 209/209 [03:00<00:00,  1.16it/s, loss=0.57]
Validation: 100%|      | 90/90 [01:10<00:00,  1.27it/s, loss=1.89]

Epoch 12/15 - Train Loss: 0.7425, Val Loss: 1.7183, Val Accuracy: 51.70%

Epoch 13/15:  42%|        | 87/209 [01:15<01:37,  1.25it/s, loss=0.743]Corrupt
JPEG data: bad Huffman code
Epoch 13/15:  44%|        | 91/209 [01:18<01:42,  1.15it/s, loss=0.57] Corrupt
JPEG data: premature end of data segment
Epoch 13/15: 100%|      | 209/209 [03:00<00:00,  1.16it/s, loss=0.584]
Validation: 100%|      | 90/90 [01:11<00:00,  1.26it/s, loss=2.14]

Epoch 13/15 - Train Loss: 0.5620, Val Loss: 1.8278, Val Accuracy: 50.70%

Epoch 14/15:  28%|         | 58/209 [00:51<02:15,  1.12it/s, loss=0.408]Corrupt
JPEG data: bad Huffman code
Epoch 14/15:  73%|       | 153/209 [02:16<00:54,  1.02it/s,
loss=0.511]Corrupt JPEG data: premature end of data segment
Epoch 14/15: 100%|      | 209/209 [03:09<00:00,  1.10it/s, loss=0.647]
Validation: 100%|      | 90/90 [01:14<00:00,  1.21it/s, loss=1.52]

Epoch 14/15 - Train Loss: 0.4245, Val Loss: 1.8726, Val Accuracy: 50.86%

Epoch 15/15:  44%|        | 91/209 [01:23<01:44,  1.13it/s, loss=0.268]Corrupt
JPEG data: bad Huffman code
Epoch 15/15:  61%|       | 127/209 [01:57<01:19,  1.04it/s, loss=0.27]
Corrupt JPEG data: premature end of data segment
Epoch 15/15: 100%|      | 209/209 [03:14<00:00,  1.07it/s, loss=0.419]
Validation: 100%|      | 90/90 [01:14<00:00,  1.21it/s, loss=1.85]

Epoch 15/15 - Train Loss: 0.3131, Val Loss: 1.9881, Val Accuracy: 50.65%
```

```
Epoch 16/15:   7%|          | 14/209 [00:12<03:01,  1.07it/s, loss=0.212]Corrupt
JPEG data: bad Huffman code
Epoch 16/15:  54%|       | 112/209 [01:41<01:29,  1.08it/s, loss=0.159]
Corrupt JPEG data: premature end of data segment
Epoch 16/15: 100%|      | 209/209 [03:08<00:00,  1.11it/s, loss=0.149]
Validation: 100%|      | 90/90 [01:17<00:00,  1.16it/s, loss=2.21]

Epoch 16/15 - Train Loss: 0.2233, Val Loss: 2.0907, Val Accuracy: 51.16%

Epoch 17/15:  27%|         | 57/209 [01:01<02:53,  1.14s/it, loss=0.12]
Corrupt JPEG data: premature end of data segment
Epoch 17/15:  43%|        | 89/209 [01:33<02:02,  1.02s/it,
loss=0.0998]Corrupt JPEG data: bad Huffman code
Epoch 17/15: 100%|      | 209/209 [03:30<00:00,  1.01s/it, loss=0.181]
Validation: 100%|      | 90/90 [01:16<00:00,  1.17it/s, loss=2.51]

Epoch 17/15 - Train Loss: 0.1674, Val Loss: 2.1928, Val Accuracy: 50.05%

Epoch 18/15:  61%|       | 128/209 [01:56<01:28,  1.09s/it, loss=0.106]
Corrupt JPEG data: bad Huffman code
Epoch 18/15:  76%|       | 159/209 [02:25<00:51,  1.04s/it, loss=0.166]
Corrupt JPEG data: premature end of data segment
Epoch 18/15: 100%|      | 209/209 [03:15<00:00,  1.07it/s, loss=0.156]
Validation: 100%|      | 90/90 [01:12<00:00,  1.24it/s, loss=3.12]

Epoch 18/15 - Train Loss: 0.1309, Val Loss: 2.2882, Val Accuracy: 50.81%

Epoch 19/15:  65%|       | 135/209 [02:01<01:05,  1.13it/s, loss=0.198]
Corrupt JPEG data: bad Huffman code
Epoch 19/15:  88%|       | 183/209 [02:44<00:23,  1.10it/s,
loss=0.0284]Corrupt JPEG data: premature end of data segment
Epoch 19/15: 100%|      | 209/209 [03:06<00:00,  1.12it/s, loss=0.109]
Validation: 100%|      | 90/90 [01:12<00:00,  1.24it/s, loss=2.08]

Epoch 19/15 - Train Loss: 0.0909, Val Loss: 2.3468, Val Accuracy: 50.37%

Epoch 20/15:  23%|         | 49/209 [00:45<02:16,  1.17it/s,
loss=0.0379]Corrupt JPEG data: premature end of data segment
Epoch 20/15:  56%|       | 118/209 [01:45<01:19,  1.15it/s, loss=0.107]
Corrupt JPEG data: bad Huffman code
Epoch 20/15: 100%|      | 209/209 [03:04<00:00,  1.13it/s, loss=0.115]
Validation: 100%|      | 90/90 [01:10<00:00,  1.28it/s, loss=1.54]

Epoch 20/15 - Train Loss: 0.0911, Val Loss: 2.2935, Val Accuracy: 50.74%
```