

```

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

 Mounted at /content/drive

1
2 import pandas as pd
3 csv_path = './content/MyDrive/drive/data/csv'
4 artist_train = pd.read_csv('/content/drive/MyDrive/data/csv/Artist/
    artist_train')
5 # lets visualize one imag
6 base_url = '/content/drive/MyDrive/data/images'
7 # lets start creating data
8 artist = '/content/drive/MyDrive/data/csv/Artist'
9 genre = '/content/drive/MyDrive/data/csv/Genre'
10 style = '/content/drive/MyDrive/data/csv/Style'
11 data_dir = '/content/drive/MyDrive/data/csv'
12
13 artist_train_path = data_dir + '/artist_train.csv'
14 artist_val_path = data_dir + '/artist_val.csv'
15 artist_class_path = data_dir + '/artist_class.txt'
16
17 genre_train_path = data_dir + '/genre_train.csv'
18 genre_val_path = data_dir + '/genre_val.csv'
19 genre_class_path = data_dir + '/genre_class.txt'
20
21 style_train_path = data_dir + '/style_train.csv'
22 style_val_path = data_dir + '/style_val.csv'
23 style_class_path = data_dir + '/style_class.txt'
24
25 artist_train = pd.read_csv(data_dir + '/artist_train.csv')
26 artist_val = pd.read_csv(data_dir + '/artist_val.csv')
27 artist_class = pd.read_csv(artist_class_path, header=None, names=
    ["artist_name"])
28
29 genre_train = pd.read_csv(data_dir + '/genre_train.csv')
30 genre_val = pd.read_csv(data_dir + '/genre_val.csv')
31 genre_class = pd.read_csv(genre_class_path, header=None, names=["genre_name"])
32
33
34 style_train = pd.read_csv(data_dir + '/style_train.csv')
35 style_val = pd.read_csv(data_dir + '/style_val.csv')
36 style_class = pd.read_csv(style_class_path, header=None, names=["style_name"])
37
38 # genre_class['genre_name'][1]
39 len(style_class)

```

 27

```

1 import os
2 import pandas as pd
3 import torch
4 from torch.utils.data import Dataset
5 from torchvision import transforms
6 from PIL import Image
7 from collections import defaultdict
8 import random
9 from tqdm import tqdm
10 import matplotlib.pyplot as plt
11
12 # Define dataset class
13 class BalancedArtDataset(Dataset):
14     def __init__(self, csv_file, img_dir, class_mapping, transform=None, images_per_class=32):
15         self.data = pd.read_csv(csv_file)
16         self.img_dir = img_dir
17         self.class_mapping = class_mapping
18         self.transform = transform
19         self.images_per_class = images_per_class
20
21     # Filter out missing images
22     print("Filtering missing images...")
23     self.data = self.data[self.data.iloc[:, 0].apply(lambda x: os.path.exists(os.path.join(img_dir, str(x))))]
24
25     # Group images by class

```

```

26     print("Grouping images by class...")
27     self.class_images = defaultdict(list)
28     for _, row in tqdm(self.data.iterrows(), total=len(self.data), desc="Processing rows"):
29         self.class_images[row.iloc[1]].append(row)
30
31     # Balance dataset with 32 images per class
32     print("Balancing dataset...")
33     self.final_data = []
34     all_images = []
35     for cls, images in tqdm(self.class_images.items(), total=len(self.class_images), desc="Processing classes"):
36         if len(images) >= images_per_class:
37             selected_images = random.sample(images, images_per_class)
38         else:
39             selected_images = images[:]
40             all_images.extend(images) # Store extra images for filling
41         self.final_data.extend(selected_images)
42
43     # Fill missing slots with extra images
44     print("Filling missing slots...")
45     needed_images = images_per_class * len(self.class_images) - len(self.final_data)
46     if needed_images > 0:
47         self.final_data.extend(random.sample(all_images, min(needed_images, len(all_images))))
48
49     # Shuffle dataset
50     print("Shuffling dataset...")
51     random.shuffle(self.final_data)
52
53     # Count images per class
54     self.class_counts = defaultdict(int)
55     for row in self.final_data:
56         self.class_counts[row.iloc[1]] += 1
57
58     def __len__(self):
59         return len(self.final_data)
60
61     def __getitem__(self, idx):
62         row = self.final_data[idx]
63         img_path = os.path.join(self.img_dir, str(row.iloc[0]))
64         label = row.iloc[1]
65         image = Image.open(img_path).convert("RGB")
66
67         if self.transform:
68             image = self.transform(image)
69
70         return image, label
71
72
73     def visualize_class_distribution(self):
74         plt.figure(figsize=(12, 6))
75         plt.bar(self.class_counts.keys(), self.class_counts.values(), color='skyblue')
76         plt.xlabel("Class")
77         plt.ylabel("Number of Images")
78         plt.title("Class Distribution in Balanced Dataset")
79         plt.xticks(rotation=45)
80         plt.show()
81
82     def visualize_samples(self, num_samples=10):
83         fig, axes = plt.subplots(1, num_samples, figsize=(40, 20))
84         for i in range(num_samples):
85             image, label = self.__getitem__(random.randint(0, len(self) - 1))
86             image = image.permute(1, 2, 0).numpy() # Convert to (H, W, C)
87             image = (image * 0.5) + 0.5 # Unnormalize
88             axes[i].imshow(image)
89             axes[i].set_title(f"Class: {label} {artist_class['artist_name'][label]}")
90             axes[i].axis("off")
91         plt.show()
92
93     # Function to compare artist and genre relationships
94
95
96
97 # Define transformations
98 transform = transforms.Compose([
99     transforms.Resize((224, 224)),
100    transforms.ToTensor(),
101    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
102 ])

```

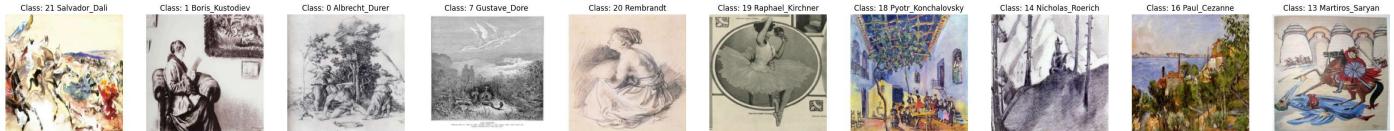
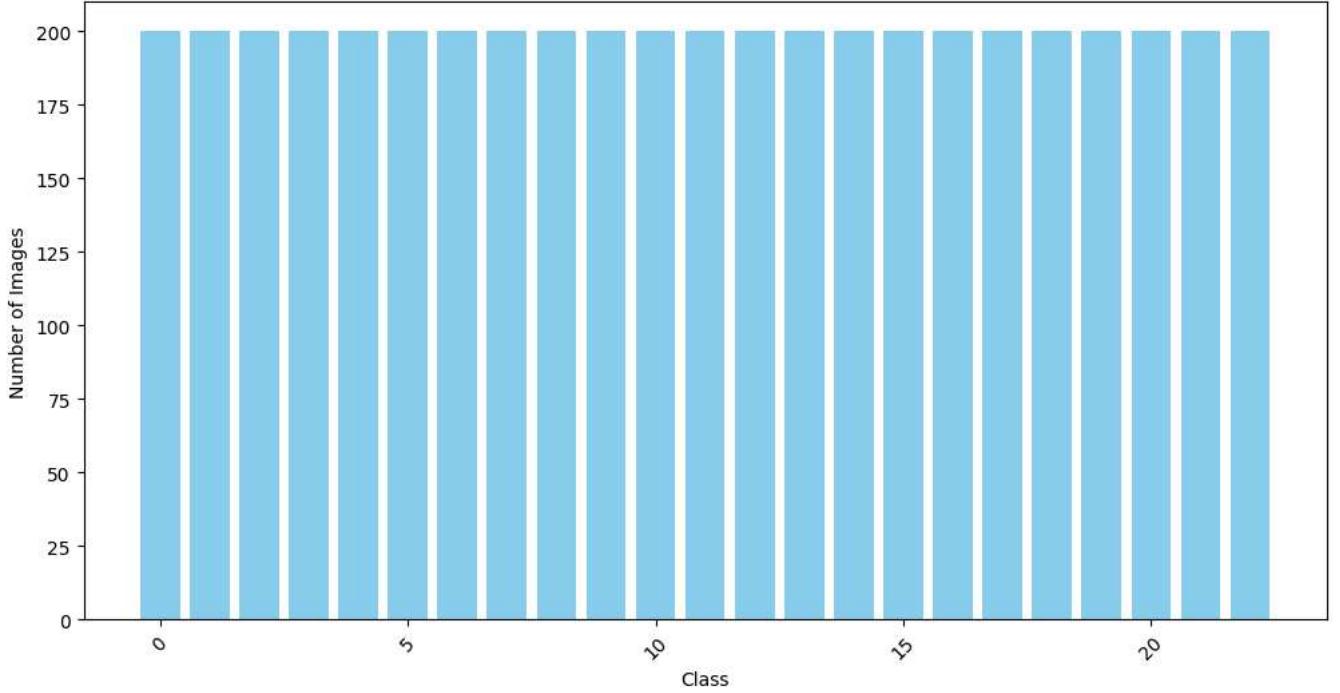
```

103
104 # Create balanced artist dataset
105 print("Creating balanced artist dataset...")
106 artist_balanced_dataset = BalancedArtDataset(artist_train_path, "/content/drive/MyDrive/data/images", artist_class_path, transform=transform)
107 artist_test_balanced_dataset = BalancedArtDataset(artist_val_path, "/content/drive/MyDrive/data/images", artist_class_path, transform=transform)
108
109 # Check dataset length
110 print("Artist balanced dataset size:", len(artist_balanced_dataset), len(artist_test_balanced_dataset))
111
112 # Visualize class distribution
113 artist_balanced_dataset.visualize_class_distribution()
114
115 # Visualize sample images
116 artist_balanced_dataset.visualize_samples(num_samples=10)
117

```

Creating balanced artist dataset...
 Filtering missing images...
 Grouping images by class...
 Processing rows: 100%|██████████| 13344/13344 [00:00<00:00, 15426.11it/s]
 Balancing dataset...
 Processing classes: 100%|██████████| 23/23 [00:00<00:00, 9064.93it/s]
 Filling missing slots...
 Shuffling dataset...
 Filtering missing images...
 Grouping images by class...
 Processing rows: 100%|██████████| 5706/5706 [00:00<00:00, 23955.41it/s]
 Balancing dataset...
 Processing classes: 100%|██████████| 23/23 [00:00<00:00, 25897.72it/s]
 Filling missing slots...
 Shuffling dataset...
 Artist balanced dataset size: 4600 1150

Class Distribution in Balanced Dataset



```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torchvision.models as models
5 from torch.utils.data import DataLoader
6 from tqdm import tqdm
7
8 # Check for GPU

```

```
9 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
10 print(f"Using device: {device}")
11
12 # Define DataLoaders
13 batch_size = 32 # Adjust based on GPU memory
14
15 train_loader = DataLoader(artist_balanced_dataset, batch_size=batch_size, shuffle=True, num_workers=4,pin_memory=True)
16 val_loader = DataLoader(artist_test_balanced_dataset, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=True)
17
18 print("Dataloaders created successfully!")
19

20 Using device: cuda
21 Dataloaders created successfully!
22 /usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes
23 warnings.warn(
24

25
26
27
28
29
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

```

60         loss = criterion(outputs, labels)
61         loss.backward() # Backpropagation
62         fc_optimizer.step() # Update FC layer weights
63
64         running_loss += loss.item()
65         _, predicted = torch.max(outputs, 1)
66         correct += (predicted == labels).sum().item()
67         total += labels.size(0)
68
69         # Update the progress bar
70         loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
71
72         print(f"Epoch [{epoch+1}/{num_epochs_fc}], Loss: {running_loss / len(train_loader):.4f}, "
73               f"Accuracy: {100 * correct / total:.2f}%")
74
75     print("⌚ Step 1 Complete: FC Layer Training Finished!")
76
77
78 # Function to fine-tune the entire ResNet-50 network
79 def fine_tune():
80     print("Starting Step 2: Fine-Tuning the Entire Network...")
81
82     # Unfreeze all layers
83     for param in model.parameters():
84         param.requires_grad = True
85
86     # Define optimizer and learning rate scheduler for fine-tuning
87     finetune_optimizer = optim.Adam(model.parameters(), lr=finetune_lr, weight_decay=weight_decay)
88     scheduler = ReduceLROnPlateau(finetune_optimizer, mode='min', factor=0.1, patience=3, verbose=True)
89
90     best_val_loss = float('inf')
91
92     for epoch in range(num_epochs_finetune):
93         model.train()
94         running_loss, correct, total = 0.0, 0, 0
95         loop = tqdm(train_loader, leave=True, desc=f"Epoch [{epoch+1}/{num_epochs_finetune}]")
96
97         for images, labels in loop:
98             images, labels = images.to(device), labels.to(device)
99
100            finetune_optimizer.zero_grad()
101
102            # Forward pass
103            outputs = model(images)
104
105            # Compute loss
106            loss = criterion(outputs, labels)
107            loss.backward()
108            finetune_optimizer.step()
109
110            running_loss += loss.item()
111            _, predicted = torch.max(outputs, 1)
112            correct += (predicted == labels).sum().item()
113            total += labels.size(0)
114
115            # Update the progress bar
116            loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
117
118     # Validation phase
119     val_loss, val_acc = evaluate(val_loader)
120     scheduler.step(val_loss)
121
122     print(f"Epoch [{epoch+1}/{num_epochs_finetune}], "
123           f"Train Loss: {running_loss / len(train_loader):.4f}, Train Acc: {100 * correct / total:.2f}%, "
124           f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
125
126     # Save the best model based on validation loss
127     if val_loss < best_val_loss:
128         best_val_loss = val_loss
129         torch.save(model.state_dict(), "/content/drive/MyDrive/art_model/best_resnet50_model.pth")
130         print("✅ Best model saved!")
131
132     print("⌚ Step 2 Complete: Fine-Tuning Finished!")
133
134
135 # Function to evaluate the model on validation data
136 def evaluate(loader):

```

```

137     model.eval()
138     val_loss, correct, total = 0.0, 0, 0
139
140     with torch.no_grad():
141         for images, labels in loader:
142             images, labels = images.to(device), labels.to(device)
143
144             outputs = model(images)
145             loss = criterion(outputs, labels)
146             val_loss += loss.item()
147             _, predicted = torch.max(outputs, 1)
148             correct += (predicted == labels).sum().item()
149             total += labels.size(0)
150
151     avg_loss = val_loss / len(loader)
152     accuracy = 100 * correct / total
153
154     return avg_loss, accuracy
155
156 # Start training
157 train_fc_layer() # Step 1: Train FC Layer only
158 fine_tune()      # Step 2: Fine-tune the entire ResNet-50 model
159
160 print("🎉 Training complete! The best model is saved as 'best_resnet50_model.pth'.")
161

```

⬇️ Downloading: "<https://download.pytorch.org/models/resnet50-11ad3fa6.pth>" to /root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth
100% [██████████] 97.8M/97.8M [00:00<00:00, 145MB/s]

Starting Step 1: Training FC Layer Only...

Epoch [1/5]: 100% [██████████] | 144/144 [09:41<00:00, 4.04s/it, acc=44.7, loss=2.16]
Epoch [1/5], Loss: 2.1568, Accuracy: 44.65%
Epoch [2/5]: 100% [██████████] | 144/144 [01:18<00:00, 1.84it/s, acc=65.2, loss=1.4]
Epoch [2/5], Loss: 1.3978, Accuracy: 65.17%
Epoch [3/5]: 100% [██████████] | 144/144 [01:18<00:00, 1.84it/s, acc=71.2, loss=1.13]
Epoch [3/5], Loss: 1.1317, Accuracy: 71.20%
Epoch [4/5]: 100% [██████████] | 144/144 [01:16<00:00, 1.88it/s, acc=75.3, loss=0.973]
Epoch [4/5], Loss: 0.9726, Accuracy: 75.26%
Epoch [5/5]: 100% [██████████] | 144/144 [01:18<00:00, 1.85it/s, acc=78.7, loss=0.857]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_warnings.warn()
Epoch [5/5], Loss: 0.8567, Accuracy: 78.67%

⌚ Step 1 Complete: FC Layer Training Finished!

Starting Step 2: Fine-Tuning the Entire Network...

Epoch [1/6]: 100% [██████████] | 144/144 [01:23<00:00, 1.73it/s, acc=81.8, loss=0.628]
Epoch [1/6], Train Loss: 0.6284, Train Acc: 81.80%, Val Loss: 0.8030, Val Acc: 75.22%
✓ Best model saved!
Epoch [2/6]: 100% [██████████] | 144/144 [01:24<00:00, 1.70it/s, acc=98.1, loss=0.122]
Epoch [2/6], Train Loss: 0.1215, Train Acc: 98.09%, Val Loss: 0.7846, Val Acc: 77.22%
✓ Best model saved!
Epoch [3/6]: 100% [██████████] | 144/144 [01:25<00:00, 1.69it/s, acc=99.6, loss=0.0396]
Epoch [3/6], Train Loss: 0.0396, Train Acc: 99.59%, Val Loss: 0.7731, Val Acc: 77.65%
✓ Best model saved!
Epoch [4/6]: 100% [██████████] | 144/144 [01:24<00:00, 1.70it/s, acc=99.9, loss=0.0171]
Epoch [4/6], Train Loss: 0.0171, Train Acc: 99.93%, Val Loss: 0.7619, Val Acc: 78.87%
✓ Best model saved!
Epoch [5/6]: 100% [██████████] | 144/144 [01:24<00:00, 1.70it/s, acc=100, loss=0.00966]
Epoch [5/6], Train Loss: 0.0097, Train Acc: 99.98%, Val Loss: 0.7586, Val Acc: 78.43%
✓ Best model saved!
Epoch [6/6]: 100% [██████████] | 144/144 [01:24<00:00, 1.69it/s, acc=100, loss=0.00559]
Epoch [6/6], Train Loss: 0.0056, Train Acc: 100.00%, Val Loss: 0.7795, Val Acc: 78.78%

⌚ Step 2 Complete: Fine-Tuning Finished!

🎉 Training complete! The best model is saved as 'best_resnet50_model.pth'.

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.optim.lr_scheduler import ReduceLROnPlateau
5 from torchvision import datasets, transforms, models
6 from torch.utils.data import DataLoader
7 from tqdm import tqdm
8 import time
9
10 # Hyperparameters
11 num_epochs_fc = 5          # Train FC layer only for 5 epochs
12 num_epochs_finetune = 6    # Fine-tune entire model for 6 more epochs
13 initial_lr = 0.001        # Learning rate for training FC layer
14 finetune_lr = 1e-4         # Lower learning rate for fine-tuning
15 batch_size = 32

```

```

16 weight_decay = 1e-4      # L2 regularization
17 hidden_size = 256        # Hidden size for LSTM
18 num_layers = 2           # Number of LSTM layers
19
20 # Define the number of output classes
21 num_classes = len(artist_class) # Replace with your actual class count
22
23 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
24
25 # Define a hybrid model combining ResNet-50 and an LSTM
26 class ResNetRNN(nn.Module):
27     def __init__(self, num_classes, hidden_size=256, num_layers=2):
28         super(ResNetRNN, self).__init__()
29         self.resnet = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
30         self.resnet.fc = nn.Identity() # Remove FC layer to get feature maps
31
32         # LSTM layer to process sequential features from ResNet's output
33         self.lstm = nn.LSTM(input_size=2048, hidden_size=hidden_size, num_layers=num_layers, batch_first=True, bidirectional=True)
34
35         # Fully connected layer after LSTM
36         self.fc = nn.Linear(hidden_size * 2, num_classes) # Bidirectional doubles the hidden size
37
38     def forward(self, x):
39         batch_size, _, _, _ = x.size()
40
41         # Extract features using ResNet
42         features = self.resnet(x) # Shape: (batch_size, 2048)
43         features = features.unsqueeze(1) # Add time dimension: (batch_size, 1, 2048)
44
45         # Pass features through LSTM
46         lstm_out, _ = self.lstm(features) # Shape: (batch_size, 1, hidden_size * 2)
47         lstm_out = lstm_out[:, -1, :] # Get the output of the last time step
48
49         # Final classification
50         output = self.fc(lstm_out)
51         return output
52
53 model = ResNetRNN(num_classes=num_classes, hidden_size=hidden_size, num_layers=num_layers).to(device)
54
55 # Define loss function and optimizer for training the FC layer
56 criterion = nn.CrossEntropyLoss()
57 fc_optimizer = optim.Adam(model.fc.parameters(), lr=initial_lr, weight_decay=weight_decay)
58
59 # Function to train the fully connected (FC) layer only
60 def train_fc_layer():
61     print("Starting Step 1: Training FC Layer Only...")
62
63     # Freeze all layers except the FC layer
64     for param in model.parameters():
65         param.requires_grad = False
66
67     # Ensure the new FC layer and LSTM are trainable
68     for param in model.fc.parameters():
69         param.requires_grad = True
70     for param in model.lstm.parameters():
71         param.requires_grad = True
72
73     model.train() # Set the model to training mode
74
75     for epoch in range(num_epochs_fc):
76         running_loss, correct, total = 0.0, 0, 0
77         loop = tqdm(train_loader, leave=True, desc=f"Epoch [{epoch+1}/{num_epochs_fc}]")
78
79         for images, labels in loop:
80             images, labels = images.to(device), labels.to(device)
81
82             # Zero the gradients
83             fc_optimizer.zero_grad()
84
85             # Forward pass
86             outputs = model(images)
87
88             # Compute loss
89             loss = criterion(outputs, labels)
90             loss.backward() # Backpropagation
91             fc_optimizer.step() # Update FC layer weights
92

```

```

93     running_loss += loss.item()
94     _, predicted = torch.max(outputs, 1)
95     correct += (predicted == labels).sum().item()
96     total += labels.size(0)
97
98     # Update the progress bar
99     loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
100
101    print(f"Epoch [{epoch+1}/{num_epochs_fc}], Loss: {running_loss / len(train_loader):.4f}, "
102          f"Accuracy: {100 * correct / total:.2f}%")
103
104   print("\ud83c\udfaf Step 1 Complete: FC Layer Training Finished!")
105
106
107 # Function to fine-tune the entire ResNet-LSTM network
108 def fine_tune():
109   print("Starting Step 2: Fine-Tuning the Entire Network...")
110
111   # Unfreeze all layers
112   for param in model.parameters():
113     param.requires_grad = True
114
115   # Define optimizer and learning rate scheduler for fine-tuning
116   finetune_optimizer = optim.Adam(model.parameters(), lr=finetune_lr, weight_decay=weight_decay)
117   scheduler = ReduceLROnPlateau(finetune_optimizer, mode='min', factor=0.1, patience=3, verbose=True)
118
119   best_val_loss = float('inf')
120
121   for epoch in range(num_epochs_finetune):
122     model.train()
123     running_loss, correct, total = 0.0, 0, 0
124     loop = tqdm(train_loader, leave=True, desc=f"Epoch [{epoch+1}/{num_epochs_finetune}]")
125
126     for images, labels in loop:
127       images, labels = images.to(device), labels.to(device)
128
129       finetune_optimizer.zero_grad()
130
131       # Forward pass
132       outputs = model(images)
133
134       # Compute loss
135       loss = criterion(outputs, labels)
136       loss.backward()
137       finetune_optimizer.step()
138
139       running_loss += loss.item()
140       _, predicted = torch.max(outputs, 1)
141       correct += (predicted == labels).sum().item()
142       total += labels.size(0)
143
144       # Update the progress bar
145       loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
146
147   # Validation phase
148   val_loss, val_acc = evaluate(val_loader)
149   scheduler.step(val_loss)
150
151   print(f"Epoch [{epoch+1}/{num_epochs_finetune}], "
152         f"Train Loss: {running_loss / len(train_loader):.4f}, Train Acc: {100 * correct / total:.2f}%, "
153         f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
154
155   # Save the best model based on validation loss
156   if val_loss < best_val_loss:
157     best_val_loss = val_loss
158     torch.save(model.state_dict(), "/content/drive/MyDrive/art_model/best_resnet_lstm_model.pth")
159     print("\u2705 Best model saved!")
160
161   print("\ud83c\udfaf Step 2 Complete: Fine-Tuning Finished!")
162
163
164 # Function to evaluate the model on validation data
165 def evaluate(loader):
166   model.eval()
167   val_loss, correct, total = 0.0, 0, 0
168
169   with torch.no_grad():

```

```

170     for images, labels in loader:
171         images, labels = images.to(device), labels.to(device)
172
173         outputs = model(images)
174         loss = criterion(outputs, labels)
175         val_loss += loss.item()
176         _, predicted = torch.max(outputs, 1)
177         correct += (predicted == labels).sum().item()
178         total += labels.size(0)
179
180     avg_loss = val_loss / len(loader)
181     accuracy = 100 * correct / total
182     return avg_loss, accuracy
183
184
185 # Start training
186 train_fc_layer() # Step 1: Train FC Layer only
187 fine_tune() # Step 2: Fine-tune the entire ResNet-LSTM model
188
189 print("\ud83c\udf89 Training complete! The best model is saved as 'best_resnet_lstm_model.pth'.")
190

```

Starting Step 1: Training FC Layer Only...

Epoch [1/5]: 100%|██████████| 144/144 [01:22<00:00, 1.75it/s, acc=21.1, loss=3.1]
 Epoch [1/5], Loss: 3.1024, Accuracy: 21.09%
 Epoch [2/5]: 100%|██████████| 144/144 [01:18<00:00, 1.85it/s, acc=40.6, loss=3.03]
 Epoch [2/5], Loss: 3.0299, Accuracy: 40.61%
 Epoch [3/5]: 100%|██████████| 144/144 [01:16<00:00, 1.88it/s, acc=44.4, loss=2.96]
 Epoch [3/5], Loss: 2.9612, Accuracy: 44.39%
 Epoch [4/5]: 100%|██████████| 144/144 [01:17<00:00, 1.85it/s, acc=45.5, loss=2.9]
 Epoch [4/5], Loss: 2.8968, Accuracy: 45.52%
 Epoch [5/5]: 100%|██████████| 144/144 [01:16<00:00, 1.88it/s, acc=46.5, loss=2.84]

ERROR:tornado.general:Uncaught exception in ZMQStream callback

Traceback (most recent call last):

- File "/usr/local/lib/python3.11/dist-packages/zmq/eventloop/zmqstream.py", line 557, in _run_callback
 callback(*args, **kwargs)
- File "/usr/local/lib/python3.11/dist-packages/ipykernel/iostream.py", line 120, in _handle_event
 event_f()
- File "/usr/local/lib/python3.11/dist-packages/ipykernel/iostream.py", line 518, in _flush
 self.session.send()
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 742, in send
 to_send = self.serialize(msg, ident)
 ^^^^^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 630, in serialize
 content = self.pack(content)
 ^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 82, in <lambda>
 json_packer = lambda obj: jsonapi.dumps(obj, default=date_default,
 ^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/zmq/utils/jsonapi.py", line 24, in dumps
 return json.dumps(o, **kwargs).encode("utf8")
 ^^^^^^

UnicodeEncodeError: 'utf-8' codec can't encode characters in position 73-74: surrogates not allowed

ERROR:tornado.general:Uncaught exception in zmqstream callback

Traceback (most recent call last):

- File "/usr/local/lib/python3.11/dist-packages/zmq/eventloop/zmqstream.py", line 578, in _handle_events
 self._handle_recv()
- File "/usr/local/lib/python3.11/dist-packages/zmq/eventloop/zmqstream.py", line 607, in _handle_recv
 self._run_callback(callback, msg)
- File "/usr/local/lib/python3.11/dist-packages/zmq/eventloop/zmqstream.py", line 557, in _run_callback
 callback(*args, **kwargs)
- File "/usr/local/lib/python3.11/dist-packages/ipykernel/iostream.py", line 120, in _handle_event
 event_f()
- File "/usr/local/lib/python3.11/dist-packages/ipykernel/iostream.py", line 518, in _flush
 self.session.send()
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 742, in send
 to_send = self.serialize(msg, ident)
 ^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 630, in serialize
 content = self.pack(content)
 ^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/jupyter_client/session.py", line 82, in <lambda>
 json_packer = lambda obj: jsonapi.dumps(obj, default=date_default,
 ^^^^^^
- File "/usr/local/lib/python3.11/dist-packages/zmq/utils/jsonapi.py", line 24, in dumps
 return json.dumps(o, **kwargs).encode("utf8")
 ^^^^^^

UnicodeEncodeError: 'utf-8' codec can't encode characters in position 73-74: surrogates not allowed

ERROR:asyncio:Exception in callback BaseAsyncIOLoop._handle_events(27, 1)

handle: <Handle BaseAsyncIOLoop._handle_events(27, 1)>

```

1 # Initialize ResNet-50 model (because the saved checkpoint is from ResNet-50)
2 model1 = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
3 model1.fc = nn.Linear(model1.fc.in_features, num_classes).to(device) # Replace the FC layer
4 model1 = model1.to(device)
5
6 # Load the correct saved state dictionary (ResNet-50 checkpoint)
7 model1.load_state_dict(torch.load("/content/drive/MyDrive/art_model/best_resnet50_model.pth"))
8
9 # Print model

```

→ <All keys matched successfully>

```

1 import torch
2 import matplotlib.pyplot as plt
3 import random
4
5
6 # Function to visualize random predictions from test dataset
7 def visualize_random_predictions(model, dataset, class_mapping, num_samples=8):
8     model.eval() # Set model to evaluation mode
9     fig, axes = plt.subplots(1, num_samples, figsize=(20, 8))
10
11    with torch.no_grad(): # Disable gradient calculation for inference
12        for i in range(num_samples):
13            # Pick a random index from the dataset
14            idx = random.randint(0, len(dataset) - 1)
15            image, true_label = dataset[idx]
16
17            # Add batch dimension and move image to the appropriate device
18            image_batch = image.unsqueeze(0).to(device)
19
20            # Make a prediction
21            output = model(image_batch)
22            _, predicted_label = torch.max(output, 1)
23
24            # Convert image tensor to numpy and unnormalize
25            image = image.permute(1, 2, 0).cpu().numpy() # Convert (C, H, W) -> (H, W, C)
26            image = (image * 0.5) + 0.5 # Unnormalize image
27
28            # Display the image and predictions
29            axes[i].imshow(image)
30            axes[i].set_title(
31                f"True: {class_mapping['artist_name'][true_label]} \nPredicted: {class_mapping['artist_name'][predicted_label.item()]}", color="green" if true_label == predicted_label.item() else "red"), fontsize=12,
32            )
33            axes[i].axis("off")
34
35    plt.show()
36
37
38
39
40
41 # Visualize random predictions on test dataset (Adjust `num_samples` as needed)
42 visualize_random_predictions(model1, artist_test_balanced_dataset, artist_class, num_samples=8)
43

```



```

1 num_classes = len(artist_class) # Number of output classes
2 model2 = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
3 model2.fc = nn.Linear(model2.fc.in_features, num_classes).to(device) # Replace final FC layer
4 model2 = model2.to(device)
5 # Load the correct saved state dictionary (ResNet-50 checkpoint)
6 model2.load_state_dict(torch.load("/content/drive/MyDrive/art_model/best_resnet18_model_200.pth"))

```

→ <All keys matched successfully>

```
1 import torch
2 import matplotlib.pyplot as plt
3 import random
4
5
6 # Function to visualize random predictions from test dataset on the same images for both models
7 def visualize_same_random_predictions(model1, model2, dataset, class_mapping, num_samples=8):
8     model1.eval() # Set models to evaluation mode
9     model2.eval()
10
11    fig, axes = plt.subplots(2, num_samples, figsize=(20, 12)) # Two rows: one for each model
12
13    # Generate fixed random indices to ensure both models predict on the same images
14    fixed_indices = [random.randint(0, len(dataset) - 1) for _ in range(num_samples)]
15
16    with torch.no_grad(): # Disable gradient calculation for inference
17        for i, idx in enumerate(fixed_indices):
18            image, true_label = dataset[idx]
19
20            # Add batch dimension and move image to the appropriate device
21            image_batch = image.unsqueeze(0).to(device)
22
23            # Make predictions with both models
24            output1 = model1(image_batch)
25            output2 = model2(image_batch)
26            _, predicted_label1 = torch.max(output1, 1)
27            _, predicted_label2 = torch.max(output2, 1)
28
29            # Convert image tensor to numpy and unnormalize
30            image = image.permute(1, 2, 0).cpu().numpy() # Convert (C, H, W) -> (H, W, C)
31            image = (image * 0.5) + 0.5 # Unnormalize image
32
33            # Display the image and predictions for model1 (top row)
34            axes[0, i].imshow(image)
35            axes[0, i].set_title(
36                f"Model 1 - True: {class_mapping['artist_name'][true_label]}\nPredicted: {class_mapping['artist_name'][predicted_label1]}"
37                color=("green" if true_label == predicted_label1.item() else "red"),
38                fontsize=12,
39            )
40            axes[0, i].axis("off")
41
42            # Display the image and predictions for model2 (bottom row)
43            axes[1, i].imshow(image)
44            axes[1, i].set_title(
45                f"Model 2 - True: {class_mapping['artist_name'][true_label]}\nPredicted: {class_mapping['artist_name'][predicted_label2]}"
46                color=("green" if true_label == predicted_label2.item() else "red"),
47                fontsize=12,
48            )
49            axes[1, i].axis("off")
50
51    plt.show()
52
53
54 # Visualize random predictions on test dataset for both models on the same set of images
55 visualize_same_random_predictions(model1, model2, artist_test_balanced_dataset, artist_class, num_samples=8)
56
```



```

1 # Import necessary libraries
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 from torch.optim.lr_scheduler import ReduceLROnPlateau
6 from torchvision import datasets, transforms, models
7 from torch.utils.data import DataLoader
8 from tqdm import tqdm
9 import time
10
11 # Hyperparameters
12 num_epochs_fc = 5           # Train FC layer only for 5 epochs
13 num_epochs_finetune = 6     # Fine-tune entire model for 6 more epochs
14 initial_lr = 0.001          # Learning rate for training FC layer
15 finetune_lr = 1e-4          # Lower learning rate for fine-tuning
16 batch_size = 32
17 weight_decay = 1e-4         # L2 regularization
18
19 # Define the number of output classes
20 num_classes = len(artist_class) # Replace with your actual class count
21
22 # Load the ResNet-50 model pre-trained on ImageNet
23 model = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
24 model.fc = nn.Linear(model.fc.in_features, num_classes).to(device) # Replace the FC layer to match your task
25
26 model = model.to(device) # Move model to the appropriate device (CPU/GPU)
27
28 # Define loss function and optimizer for training the FC layer
29 criterion = nn.CrossEntropyLoss()
30 fc_optimizer = optim.Adam(model.fc.parameters(), lr=initial_lr, weight_decay=weight_decay)
31 fc_scheduler = ReduceLROnPlateau(fc_optimizer, mode='min', factor=0.1, patience=2, verbose=True) # FC stage LR schedule
32
33 # Function to train the fully connected (FC) layer only
34 def train_fc_layer():
35     print("Starting Step 1: Training FC Layer Only...")
36
37     # Freeze all layers except the FC layer
38     for param in model.parameters():
39         param.requires_grad = False
40
41     # Ensure the new FC layer is trainable
42     for param in model.fc.parameters():

```

```
43     param.requires_grad = True
44
45     model.train() # Set the model to training mode
46
47     for epoch in range(num_epochs_fc):
48         running_loss, correct, total = 0.0, 0, 0
49         loop = tqdm(train_loader, leave=True, desc=f"Epoch [{epoch+1}/{num_epochs_fc}]")
50
51         for images, labels in loop:
52             images, labels = images.to(device), labels.to(device)
53
54             # Zero the gradients
55             fc_optimizer.zero_grad()
56
57             # Forward pass
58             outputs = model(images)
59
60             # Compute loss
61             loss = criterion(outputs, labels)
62             loss.backward() # Backpropagation
63             fc_optimizer.step() # Update FC layer weights
64
65             running_loss += loss.item()
66             _, predicted = torch.max(outputs, 1)
67             correct += (predicted == labels).sum().item()
68             total += labels.size(0)
69
70             # Update the progress bar
71             loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
72
73     # Evaluate on validation set and update the scheduler
74     val_loss, val_acc = evaluate(val_loader)
75     fc_scheduler.step(val_loss)
76
77     print(f"Epoch [{epoch+1}/{num_epochs_fc}], Loss: {running_loss / len(train_loader):.4f}, "
78           f"Accuracy: {100 * correct / total:.2f}%, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
79
80     print("🔴 Step 1 Complete: FC Layer Training Finished!")
81
82
83 # Function to fine-tune the entire ResNet-50 network
84 def fine_tune():
85     print("Starting Step 2: Fine-Tuning the Entire Network...")
86
87     # Unfreeze all layers
88     for param in model.parameters():
89         param.requires_grad = True
90
91     # Define optimizer and learning rate scheduler for fine-tuning
92     finetune_optimizer = optim.Adam(model.parameters(), lr=finetune_lr, weight_decay=weight_decay)
93     finetune_scheduler = ReduceLROnPlateau(finetune_optimizer, mode='min', factor=0.1, patience=3, verbose=True)
94
95     best_val_loss = float('inf')
96
97     for epoch in range(num_epochs_finetune):
98         model.train()
99         running_loss, correct, total = 0.0, 0, 0
100        loop = tqdm(train_loader, leave=True, desc=f"Epoch [{epoch+1}/{num_epochs_finetune}]")
101
102        for images, labels in loop:
103            images, labels = images.to(device), labels.to(device)
104
105            finetune_optimizer.zero_grad()
106
107            # Forward pass
108            outputs = model(images)
109
110            # Compute loss
111            loss = criterion(outputs, labels)
112            loss.backward()
113            finetune_optimizer.step()
114
115            running_loss += loss.item()
116            _, predicted = torch.max(outputs, 1)
117            correct += (predicted == labels).sum().item()
118            total += labels.size(0)
119
```

```

120         # Update the progress bar
121         loop.set_postfix(loss=running_loss / len(train_loader), acc=100 * correct / total)
122
123     # Validation phase and scheduler step
124     val_loss, val_acc = evaluate(val_loader)
125     finetune_scheduler.step(val_loss)
126
127     print(f"Epoch [{epoch+1}/{num_epochs_finetune}], "
128           f"Train Loss: {running_loss / len(train_loader):.4f}, Train Acc: {100 * correct / total:.2f}%, "
129           f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
130
131     # Save the best model based on validation loss
132     if val_loss < best_val_loss:
133         best_val_loss = val_loss
134         torch.save(model.state_dict(), "/content/drive/MyDrive/art_model/best_resnet50_model_ReduceLROnPlateau.pth")
135         print("✅ Best model saved!")
136
137     print("⌚ Step 2 Complete: Fine-Tuning Finished!")
138
139
140 # Function to evaluate the model on validation data
141 def evaluate(loader):
142     model.eval()
143     val_loss, correct, total = 0.0, 0, 0
144
145     with torch.no_grad():
146         for images, labels in loader:
147             images, labels = images.to(device), labels.to(device)
148
149             outputs = model(images)
150             loss = criterion(outputs, labels)
151             val_loss += loss.item()
152             _, predicted = torch.max(outputs, 1)
153             correct += (predicted == labels).sum().item()
154             total += labels.size(0)
155
156     avg_loss = val_loss / len(loader)
157     accuracy = 100 * correct / total
158     return avg_loss, accuracy
159
160
161 # Start training
162 train_fc_layer() # Step 1: Train FC Layer only
163 fine_tune()       # Step 2: Fine-tune the entire ResNet-50 model
164
165 print("🎉 Training complete! The best model is saved as 'best_resnet50_model.pth'.")
```

→ Downloading: "<https://download.pytorch.org/models/resnet50-11ad3fa6.pth>" to /root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth
100% [██████████] | 97.8M/97.8M [00:00<00:00, 189MB/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_warnings.warn()
Starting Step 1: Training FC Layer Only...
Epoch [1/5]: 100% [██████████] | 144/144 [08:04<00:00, 3.36s/it, acc=43.3, loss=2.16]
Epoch [1/5], Loss: 2.1635, Accuracy: 43.28%, Val Loss: 1.6792, Val Acc: 55.65%
Epoch [2/5]: 100% [██████████] | 144/144 [01:20<00:00, 1.79it/s, acc=66.8, loss=1.36]
Epoch [2/5], Loss: 1.3646, Accuracy: 66.83%, Val Loss: 1.3785, Val Acc: 62.26%
Epoch [3/5]: 100% [██████████] | 144/144 [01:19<00:00, 1.81it/s, acc=75.1, loss=1.05]
Epoch [3/5], Loss: 1.0484, Accuracy: 75.11%, Val Loss: 1.2572, Val Acc: 64.17%
Epoch [4/5]: 100% [██████████] | 144/144 [01:19<00:00, 1.82it/s, acc=80.8, loss=0.854]
Epoch [4/5], Loss: 0.8541, Accuracy: 80.78%, Val Loss: 1.1750, Val Acc: 67.04%
Epoch [5/5]: 100% [██████████] | 144/144 [01:19<00:00, 1.80it/s, acc=84.5, loss=0.718]
Epoch [5/5], Loss: 0.7182, Accuracy: 84.48%, Val Loss: 1.1262, Val Acc: 67.48%
⌚ Step 1 Complete: FC Layer Training Finished!
Starting Step 2: Fine-Tuning the Entire Network...
Epoch [1/6]: 100% [██████████] | 144/144 [01:26<00:00, 1.66it/s, acc=82.8, loss=0.617]
Epoch [1/6], Train Loss: 0.6166, Train Acc: 82.85%, Val Loss: 0.8257, Val Acc: 76.26%
✅ Best model saved!
Epoch [2/6]: 100% [██████████] | 144/144 [01:26<00:00, 1.66it/s, acc=97.8, loss=0.127]
Epoch [2/6], Train Loss: 0.1271, Train Acc: 97.80%, Val Loss: 0.8138, Val Acc: 76.78%
✅ Best model saved!
Epoch [3/6]: 100% [██████████] | 144/144 [01:26<00:00, 1.67it/s, acc=99.9, loss=0.0339]
Epoch [3/6], Train Loss: 0.0339, Train Acc: 99.89%, Val Loss: 0.8254, Val Acc: 78.43%
Epoch [4/6]: 100% [██████████] | 144/144 [01:26<00:00, 1.67it/s, acc=99.9, loss=0.0161]
Epoch [4/6], Train Loss: 0.0161, Train Acc: 99.93%, Val Loss: 0.8334, Val Acc: 78.09%
Epoch [5/6]: 100% [██████████] | 144/144 [01:27<00:00, 1.65it/s, acc=100, loss=0.0185]
Epoch [5/6], Train Loss: 0.0105, Train Acc: 99.98%, Val Loss: 0.8419, Val Acc: 78.96%
Epoch [6/6]: 100% [██████████] | 144/144 [01:26<00:00, 1.67it/s, acc=100, loss=0.00694]
Epoch [6/6], Train Loss: 0.0069, Train Acc: 99.98%, Val Loss: 0.8669, Val Acc: 77.74%