effnet_rcnn_style

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_effnet",␣
 ↪config={
    "epochs": 20,
    "batch_size": 32,
    "learning_rate": 0.001,
    "model": "EFFNET_RCNN"
})
```

```
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 0x714a24c95720>
```

0.1 Data loading

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

[4]:
```
# get the number of classes
num_classes = 27 # from style_class.txt
```

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

val_images = df_val['image_path'].values
val_labels = df_val['style_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=16,␣
 ↪shuffle=True)
val_dataset = WikiArtDataset(val_images, val_labels)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16,␣
 ↪shuffle=False)

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

```
('Post_Impressionism/paul-gauguin_and-the-gold-of-their-bodies-1901.jpg',
'Rococo/maurice-quentin-de-la-tour_portrait-of-madame-de-pompadour-1752.jpg',
'Fauvism/aldemir-martins_galo-1987.jpg', 'Baroque/adriaen-van-ostade_portrait-
of-a-boy.jpg', 'Baroque/jan-steen_merry-company-on-a-terrace-1675.jpg',
'Baroque/francisco-de-zurbaran_st-romanus-and-st-barulas-of-antioch-1638.jpg',
'Post_Impressionism/pierre-bonnard_girl-with-a-dog-in-the-park-at-grand-lemps-
also-known-as-dauphine-1900.jpg', 'Realism/gustave-courbet_the-white-
sail-1877.jpg', 'Impressionism/walter-sickert_mrs-barrett.jpg', 'Rococo/fyodor-
rokotov_portrait-of-catherine-ii-repeat-version-of-a-portrait-after-1768.jpg',
'Realism/konstantin-somov_lady-in-blue-portrait-of-the-artist-yelizaveta-
martynova-1900.jpg', 'Expressionism/martiros-saryan_in-armenia-1923.jpg',
'Realism/giovanni-boldini_portrait-of-guiseppe-verdi-1813-1901-1886.jpg',
'Impressionism/pierre-auguste-renoir_still-life-with-pomegranates.jpg',
'High_Renaissance/luca-signorelli_dante-and-virgil-entering-purgatory-1502.jpg',
'Pop_Art/claes-oldenburg_free-stamp-at-cleveland-city-hall-collaboration-with-
van-bruggen.jpg')
tensor([[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., 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., 0., 0., 1., 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., 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., 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., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [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., 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., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 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., 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., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 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., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 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.]])
```

[ ]:
```python
# CNN model

import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models

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

        # EfficientNet-B0 backbone (outputs 1280 channels)
        effnet = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.
 ↪IMAGENET1K_V1)
        self.cnn = effnet.features

        self.channel_reducer = nn.Sequential(
            nn.Conv2d(1280, 512, kernel_size=1),
```

```
            nn.BatchNorm2d(512),
            nn.ReLU()
        )

        self.lstm = nn.LSTM(
            input_size=512,
            hidden_size=256,
            num_layers=2,
            bidirectional=True,
            batch_first=True
        )

        self.classifier = nn.Sequential(
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        features = self.cnn(x)

        x = self.channel_reducer(features)
        bs, c, h, w = x.size()
        x = x.permute(0, 2, 3, 1).reshape(bs, h*w, c)

        lstm_out, (h_n, c_n) = self.lstm(x)
        last_hidden = torch.cat((h_n[-2], h_n[-1]), dim=1)

        return self.classifier(last_hidden)

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

# Loss and optimizer
import torch.optim as optim

wandb.watch(model, log="all")
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam([
    {'params': model.cnn.parameters(), 'lr': 1e-5},
    {'params': model.channel_reducer.parameters(), 'lr': 1e-4},
    {'params': model.lstm.parameters(), 'lr': 1e-4},
    {'params': model.classifier.parameters(), 'lr': 1e-4}
])
```

0.3 Training the model

[9]:
```
# Train the model
num_epochs = 20

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}%")
    if(epoch%5==0):
        torch.save(model.state_dict(), f"effnet_rcnn_epoch_{epoch+1}_style.pth")
        torch.save(optimizer.state_dict(),␣
↪f"effnet_rcnn_optimizer_epoch_{epoch+1}_style.pth")
```

```
Epoch 1/20:  48%|        | 1718/3565 [10:12<11:26,  2.69it/s,
loss=2.23]Corrupt JPEG data: bad Huffman code
Epoch 1/20:  83%|      | 2958/3565 [17:25<03:57,  2.55it/s,
loss=2.17]Corrupt JPEG data: premature end of data segment
Epoch 1/20: 100%|      | 3565/3565 [20:57<00:00,  2.83it/s, loss=3.05]
Validation: 100%|      | 1527/1527 [06:48<00:00,  3.74it/s, loss=2.99]

Epoch 1/20 - Train Loss: 2.1877, Val Loss: 1.8370, Val Accuracy: 40.27%

Epoch 2/20:   5%|          | 180/3565 [01:02<17:15,  3.27it/s, loss=1.86]Corrupt
JPEG data: premature end of data segment
Epoch 2/20:  81%|      | 2905/3565 [16:46<04:00,  2.74it/s, loss=1.45]
Corrupt JPEG data: bad Huffman code
Epoch 2/20: 100%|      | 3565/3565 [20:40<00:00,  2.87it/s, loss=1.93]
Validation: 100%|      | 1527/1527 [06:40<00:00,  3.81it/s, loss=2.75]

Epoch 2/20 - Train Loss: 1.8250, Val Loss: 1.6779, Val Accuracy: 45.74%

Epoch 3/20:  95%|      | 3372/3565 [19:11<00:57,  3.34it/s, loss=1.84]
Corrupt JPEG data: bad Huffman code
Epoch 3/20:  98%|      | 3503/3565 [20:00<00:21,  2.92it/s, loss=1.39]
Corrupt JPEG data: premature end of data segment
Epoch 3/20: 100%|      | 3565/3565 [20:23<00:00,  2.91it/s, loss=2.12]
Validation: 100%|      | 1527/1527 [06:45<00:00,  3.76it/s, loss=2.6]

Epoch 3/20 - Train Loss: 1.6678, Val Loss: 1.5599, Val Accuracy: 48.49%

Epoch 4/20:  60%|       | 2137/3565 [12:19<06:57,  3.42it/s, loss=1.59]
Corrupt JPEG data: premature end of data segment
Epoch 4/20:  81%|      | 2894/3565 [17:01<03:16,  3.42it/s, loss=1.69]
Corrupt JPEG data: bad Huffman code
Epoch 4/20: 100%|      | 3565/3565 [21:09<00:00,  2.81it/s, loss=2.71]
Validation: 100%|      | 1527/1527 [06:55<00:00,  3.67it/s, loss=2.29]

Epoch 4/20 - Train Loss: 1.5504, Val Loss: 1.5222, Val Accuracy: 50.18%

Epoch 5/20:  43%|        | 1529/3565 [08:54<10:25,  3.25it/s, loss=1.84]
Corrupt JPEG data: bad Huffman code
```

```
Epoch 5/20:  60%|       | 2141/3565 [12:28<09:31,  2.49it/s, loss=1.32]
Corrupt JPEG data: premature end of data segment
Epoch 5/20: 100%|      | 3565/3565 [20:49<00:00,  2.85it/s, loss=1.22]
Validation: 100%|      | 1527/1527 [06:25<00:00,  3.96it/s, loss=2.3]

Epoch 5/20 - Train Loss: 1.4688, Val Loss: 1.4633, Val Accuracy: 51.37%

Epoch 6/20:  31%|        | 1093/3565 [06:07<14:14,  2.89it/s, loss=1.49]
Corrupt JPEG data: premature end of data segment
Epoch 6/20:  68%|       | 2409/3565 [13:36<05:56,  3.24it/s, loss=2.02]
Corrupt JPEG data: bad Huffman code
Epoch 6/20: 100%|      | 3565/3565 [20:15<00:00,  2.93it/s, loss=3.74]
Validation: 100%|      | 1527/1527 [06:26<00:00,  3.95it/s, loss=2]

Epoch 6/20 - Train Loss: 1.3947, Val Loss: 1.4699, Val Accuracy: 51.94%

Epoch 7/20:  63%|       | 2233/3565 [12:26<08:28,  2.62it/s, loss=1.05]
Corrupt JPEG data: bad Huffman code
Epoch 7/20:  90%|      | 3194/3565 [18:23<02:38,  2.34it/s, loss=1.02]
Corrupt JPEG data: premature end of data segment
Epoch 7/20: 100%|      | 3565/3565 [20:41<00:00,  2.87it/s, loss=6.95]
Validation: 100%|      | 1527/1527 [06:55<00:00,  3.68it/s, loss=2.23]

Epoch 7/20 - Train Loss: 1.3154, Val Loss: 1.4130, Val Accuracy: 54.04%

Epoch 8/20:  62%|       | 2206/3565 [12:48<06:47,  3.33it/s, loss=1.07]
Corrupt JPEG data: premature end of data segment
Epoch 8/20:  90%|      | 3198/3565 [18:46<02:03,  2.97it/s, loss=1.39]
Corrupt JPEG data: bad Huffman code
Epoch 8/20: 100%|      | 3565/3565 [20:56<00:00,  2.84it/s, loss=0.741]
Validation: 100%|      | 1527/1527 [06:59<00:00,  3.64it/s, loss=1.74]

Epoch 8/20 - Train Loss: 1.2510, Val Loss: 1.3922, Val Accuracy: 54.58%

Epoch 9/20:  14%|         | 491/3565 [03:02<17:49,  2.88it/s,
loss=0.577]Corrupt JPEG data: bad Huffman code
Epoch 9/20:  49%|        | 1754/3565 [10:39<11:43,  2.57it/s, loss=1.27]
Corrupt JPEG data: premature end of data segment
Epoch 9/20: 100%|      | 3565/3565 [21:50<00:00,  2.72it/s, loss=4.02]
Validation: 100%|      | 1527/1527 [07:20<00:00,  3.47it/s, loss=1.58]

Epoch 9/20 - Train Loss: 1.1916, Val Loss: 1.3341, Val Accuracy: 56.07%

Epoch 10/20:   6%|         | 224/3565 [01:19<21:12,  2.62it/s,
loss=0.931]Corrupt JPEG data: bad Huffman code
Epoch 10/20:  50%|        | 1800/3565 [11:07<09:32,  3.08it/s,
loss=0.867]Corrupt JPEG data: premature end of data segment
Epoch 10/20: 100%|      | 3565/3565 [21:35<00:00,  2.75it/s, loss=1.08]
Validation: 100%|      | 1527/1527 [06:51<00:00,  3.71it/s, loss=2.02]

Epoch 10/20 - Train Loss: 1.1321, Val Loss: 1.3832, Val Accuracy: 55.46%

Epoch 11/20:   3%|         | 96/3565 [00:34<19:56,  2.90it/s, loss=1.32]
Corrupt JPEG data: premature end of data segment
```

```
Epoch 11/20:  58%|       | 2061/3565 [12:14<11:01,  2.27it/s, loss=1.68]
Corrupt JPEG data: bad Huffman code
Epoch 11/20: 100%|      | 3565/3565 [21:02<00:00,  2.82it/s, loss=2.94]
Validation: 100%|      | 1527/1527 [06:34<00:00,  3.87it/s, loss=1.73]

Epoch 11/20 - Train Loss: 1.0710, Val Loss: 1.3593, Val Accuracy: 56.45%

Epoch 12/20:   7%|          | 254/3565 [01:24<15:14,  3.62it/s,
loss=0.753]Corrupt JPEG data: bad Huffman code
Epoch 12/20:  43%|        | 1532/3565 [08:46<11:35,  2.92it/s, loss=1]
Corrupt JPEG data: premature end of data segment
Epoch 12/20: 100%|      | 3565/3565 [20:30<00:00,  2.90it/s, loss=1.01]
Validation: 100%|      | 1527/1527 [06:36<00:00,  3.85it/s, loss=1.86]

Epoch 12/20 - Train Loss: 1.0099, Val Loss: 1.3726, Val Accuracy: 56.44%

Epoch 13/20:  29%|         | 1048/3565 [05:54<15:43,  2.67it/s,
loss=0.536]Corrupt JPEG data: bad Huffman code
Epoch 13/20:  86%|      | 3059/3565 [17:26<03:39,  2.30it/s,
loss=0.653]Corrupt JPEG data: premature end of data segment
Epoch 13/20: 100%|      | 3565/3565 [20:26<00:00,  2.91it/s, loss=2.25]
Validation: 100%|      | 1527/1527 [06:36<00:00,  3.85it/s, loss=1.95]

Epoch 13/20 - Train Loss: 0.9635, Val Loss: 1.4224, Val Accuracy: 56.26%

Epoch 14/20:  67%|       | 2381/3565 [13:40<06:36,  2.98it/s, loss=1.57]
Corrupt JPEG data: bad Huffman code
Epoch 14/20:  69%|       | 2452/3565 [14:05<05:38,  3.29it/s, loss=1.16]
Corrupt JPEG data: premature end of data segment
Epoch 14/20: 100%|      | 3565/3565 [20:53<00:00,  2.84it/s, loss=4.8]
Validation: 100%|      | 1527/1527 [07:13<00:00,  3.52it/s, loss=2.21]

Epoch 14/20 - Train Loss: 0.9141, Val Loss: 1.3958, Val Accuracy: 56.37%

Epoch 15/20:  46%|        | 1632/3565 [09:59<11:23,  2.83it/s,
loss=0.893]Corrupt JPEG data: bad Huffman code
Epoch 15/20:  97%|      | 3462/3565 [21:02<00:39,  2.60it/s,
loss=0.699]Corrupt JPEG data: premature end of data segment
Epoch 15/20: 100%|      | 3565/3565 [21:40<00:00,  2.74it/s, loss=8.51]
Validation: 100%|      | 1527/1527 [06:51<00:00,  3.71it/s, loss=2.95]

Epoch 15/20 - Train Loss: 0.8691, Val Loss: 1.4523, Val Accuracy: 55.90%

Epoch 16/20:  26%|         | 926/3565 [05:32<15:09,  2.90it/s,
loss=0.872]Corrupt JPEG data: premature end of data segment
Epoch 16/20:  65%|       | 2309/3565 [13:51<07:32,  2.78it/s, loss=0.99]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[9], line 9
      7 train_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")
      8 for image_paths, labels in train_bar:
```

```
----> 9     image_tensors = torch.stack([get_image(image_path) for image_path i
 ↪image_paths])
     10     images = image_tensors.to('cuda')
     11     labels = labels.to('cuda')

Cell In[9], line 9, in <listcomp>(.0)
      7 train_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")
      8 for image_paths, labels in train_bar:
----> 9     image_tensors = torch.stack([get_image(image_path) for image_path i
 ↪image_paths])
     10     images = image_tensors.to('cuda')
     11     labels = labels.to('cuda')

Cell In[7], line 5, in get_image(image_path, image_size)
      3 def get_image(image_path,image_size=224):
      4     try:
----> 5         img = cv2.imread('./wikiart/' + image_path)
      6         if img is None:
      7             raise ValueError(f"Image not loaded: ./wikiart/{image_path} )

KeyboardInterrupt:
```