effnet_rcnn_genre

March 25, 2025

[2]:
```
import torch
import pandas as pd
import numpy as np
from tqdm import tqdm
import wandb
```

[3]:
```
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>
```

[3]:
```
<wandb.sdk.wandb_run.Run at 0x7e8191cc1390>
```

0.1 Data loading

[4]:
```
df_train = pd.read_csv('wikiart_csv/genre_train.csv',header=None,␣
 ↪names=["image_path", "genre_id"])
df_val = pd.read_csv('wikiart_csv/genre_val.csv',header=None,␣
 ↪names=["image_path", "genre_id"])
```

[5]:
```
# get the number of classes
num_classes = 10 # from artist_class.txt
```

[6]:
```
# Gather input data
train_images = df_train['image_path'].values
train_labels = df_train['genre_id'].values

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

[7]:
```
from torchvision import transforms
import cv2
```

0.2 Preprocess data and create test and train dataset

[ ]:
```
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)
```

```python
    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=32,␣
 ↪shuffle=True)
val_dataset = WikiArtDataset(val_images, val_labels)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32,␣
 ↪shuffle=False)

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

```
('Impressionism/claude-monet_small-boat-on-the-small-branch-of-the-seine-at-
argenteuil.jpg', 'Art_Nouveau_Modern/raphael-kirchner_lillian-lorraine.jpg',
'Abstract_Expressionism/conrad-marca-relli_untitled-1958.jpg',
'Naive_Art_Primitivism/marc-chagall_lovers.jpg', 'Rococo/bernardo-bellotto_the-
kreuzkirche-in-dresden.jpg', 'High_Renaissance/michelangelo_the-prophet-
daniel-1511.jpg', 'High_Renaissance/pietro-perugino_pala-di-monteripido.jpg',
'Symbolism/nicholas-roerich_and-we-are-opening-the-gates-1922.jpg',
'Impressionism/claude-monet_vetheuil-afternoon.jpg', 'Romanticism/francisco-
goya_the-repentant-saint-peter.jpg', 'Realism/edgar-degas_portrait-of-rene-de-
gas-1855.jpg', 'Realism/camille-corot_dieppe-end-of-a-pier-and-the-
sea-1822.jpg', 'Romanticism/orest-kiprensky_peasant-boy-1814.jpg',
'Expressionism/richard-gerstl_small-street-nu-dorferstra-e-1908.jpg',
'Abstract_Expressionism/rafa-nasiri_untitled-068-2002.jpg',
'Romanticism/gheorghe-tattarescu_laz-r-kalinderu.jpg', 'Realism/ivan-
shishkin_countess-mordvinov-s-forest-1891.jpg', 'Baroque/jan-steen_twelfth-
night-1668.jpg', 'Realism/salvador-dali_cathedral-unfinished.jpg',
'Northern_Renaissance/pieter-bruegel-the-elder_the-fair-at-hoboken-1559.jpg',
'Impressionism/claude-monet_palazzo-contarini-2.jpg',
'Abstract_Expressionism/brice-marden_dragons-2004.jpg',
'Post_Impressionism/henri-de-toulouse-lautrec_girl-in-a-fur-mademoiselle-jeanne-
fontaine-1891.jpg', 'Color_Field_Painting/john-hoyland_22-8-74-1974.jpg',
```

```
'Expressionism/oskar-kokoschka_not_detected_235843.jpg', 'Impressionism/pierre-
auguste-renoir_the-first-outing-1876.jpg', 'Expressionism/nicolae-
tonitza_morning-at-balcic.jpg', 'Baroque/jan-steen_prayer-before-meal-1660.jpg',
'Realism/ivan-shishkin_among-the-open-valley-1883.jpg', 'Baroque/anthony-van-
dyck_margareta-snyders.jpg', 'Ukiyo_e/kitagawa-utamaro_a-woman-watches-two-
children.jpg', 'New_Realism/edward-hopper_untitled.jpg')
tensor([[0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 1., 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., 1., 0., 0.],
        [0., 1., 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., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])
```

[ ]:
```python
# CNN model

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

class EffNetLSTM(nn.Module):
```

```python
    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

[ ]:
```
# 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}_genre.pth")
        torch.save(optimizer.state_dict(),␣
↪f"effnet_rcnn_optimizer_epoch_{epoch+1}_genre.pth")
```

```
Epoch 1/20:  58%|        | 824/1422 [08:56<07:01,  1.42it/s, loss=1.18]
Corrupt JPEG data: premature end of data segment
Epoch 1/20:  82%|       | 1164/1422 [12:35<02:51,  1.51it/s, loss=1.35]
Corrupt JPEG data: bad Huffman code
Epoch 1/20: 100%|      | 1422/1422 [15:20<00:00,  1.54it/s, loss=1.03]
Validation: 100%|      | 610/610 [04:51<00:00,  2.09it/s, loss=0.6]

Epoch 1/20 - Train Loss: 1.1836, Val Loss: 0.8882, Val Accuracy: 70.15%

Epoch 2/20:  56%|        | 791/1422 [08:29<08:09,  1.29it/s, loss=1.07]
Corrupt JPEG data: bad Huffman code
Epoch 2/20:  97%|      | 1386/1422 [14:49<00:21,  1.69it/s,
loss=0.583]Corrupt JPEG data: premature end of data segment
Epoch 2/20: 100%|      | 1422/1422 [15:11<00:00,  1.56it/s, loss=0.774]
Validation: 100%|      | 610/610 [04:49<00:00,  2.11it/s, loss=0.204]

Epoch 2/20 - Train Loss: 0.8713, Val Loss: 0.7987, Val Accuracy: 72.35%

Epoch 3/20:  10%|          | 148/1422 [01:31<13:01,  1.63it/s,
loss=0.905]Corrupt JPEG data: bad Huffman code
Epoch 3/20:  30%|         | 424/1422 [04:28<10:34,  1.57it/s,
loss=0.983]Corrupt JPEG data: premature end of data segment
Epoch 3/20: 100%|      | 1422/1422 [15:05<00:00,  1.57it/s, loss=0.523]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.157]

Epoch 3/20 - Train Loss: 0.7778, Val Loss: 0.7656, Val Accuracy: 73.76%

Epoch 4/20:  38%|        | 537/1422 [05:42<10:12,  1.45it/s, loss=0.44]
```

```
Corrupt JPEG data: premature end of data segment
Epoch 4/20:  69%|       | 978/1422 [10:22<04:53,  1.51it/s, loss=1.04]
Corrupt JPEG data: bad Huffman code
Epoch 4/20: 100%|      | 1422/1422 [15:07<00:00,  1.57it/s, loss=0.917]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.035]

Epoch 4/20 - Train Loss: 0.7073, Val Loss: 0.7555, Val Accuracy: 74.07%

Epoch 5/20:  57%|       | 812/1422 [08:41<07:30,  1.35it/s, loss=1.04]
Corrupt JPEG data: premature end of data segment
Epoch 5/20:  79%|      | 1121/1422 [11:55<03:11,  1.57it/s,
loss=0.732]Corrupt JPEG data: bad Huffman code
Epoch 5/20: 100%|      | 1422/1422 [15:08<00:00,  1.56it/s, loss=0.52]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.079]

Epoch 5/20 - Train Loss: 0.6479, Val Loss: 0.7431, Val Accuracy: 74.32%

Epoch 6/20:   6%|          | 80/1422 [00:52<17:43,  1.26it/s, loss=0.683]Corrupt
JPEG data: bad Huffman code
Epoch 6/20:  18%|         | 251/1422 [02:39<11:19,  1.72it/s,
loss=0.924]Corrupt JPEG data: premature end of data segment
Epoch 6/20: 100%|      | 1422/1422 [15:08<00:00,  1.57it/s, loss=0.751]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.0644]

Epoch 6/20 - Train Loss: 0.5945, Val Loss: 0.7447, Val Accuracy: 74.87%

Epoch 7/20:   6%|          | 84/1422 [00:53<15:58,  1.40it/s, loss=0.664]Corrupt
JPEG data: premature end of data segment
Epoch 7/20:  54%|       | 768/1422 [08:06<07:09,  1.52it/s,
loss=0.575]Corrupt JPEG data: bad Huffman code
Epoch 7/20: 100%|      | 1422/1422 [15:08<00:00,  1.57it/s, loss=0.173]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.0354]

Epoch 7/20 - Train Loss: 0.5459, Val Loss: 0.7553, Val Accuracy: 74.84%

Epoch 8/20:  34%|        | 484/1422 [05:00<09:12,  1.70it/s, loss=0.48]
Corrupt JPEG data: bad Huffman code
Epoch 8/20:  77%|      | 1094/1422 [11:34<03:16,  1.67it/s,
loss=0.685]Corrupt JPEG data: premature end of data segment
Epoch 8/20: 100%|      | 1422/1422 [15:06<00:00,  1.57it/s, loss=0.312]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.195]

Epoch 8/20 - Train Loss: 0.4907, Val Loss: 0.7706, Val Accuracy: 74.84%

Epoch 9/20:  15%|         | 211/1422 [02:12<12:17,  1.64it/s,
loss=0.241]Corrupt JPEG data: bad Huffman code
Epoch 9/20:  24%|         | 335/1422 [03:30<11:42,  1.55it/s, loss=0.69]
Corrupt JPEG data: premature end of data segment
Epoch 9/20: 100%|      | 1422/1422 [15:04<00:00,  1.57it/s, loss=0.492]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.1]

Epoch 9/20 - Train Loss: 0.4510, Val Loss: 0.7911, Val Accuracy: 75.19%
```

```
Epoch 10/20:   2%|          | 24/1422 [00:14<14:36,  1.59it/s,
loss=0.443]Corrupt JPEG data: premature end of data segment
Epoch 10/20:  82%|      | 1171/1422 [12:23<02:49,  1.48it/s,
loss=0.485]Corrupt JPEG data: bad Huffman code
Epoch 10/20: 100%|      | 1422/1422 [15:05<00:00,  1.57it/s, loss=0.174]
Validation: 100%|      | 610/610 [04:49<00:00,  2.11it/s, loss=0.0766]

Epoch 10/20 - Train Loss: 0.4041, Val Loss: 0.8214, Val Accuracy: 74.76%

Epoch 11/20:   3%|          | 48/1422 [00:30<16:54,  1.35it/s,
loss=0.299]Corrupt JPEG data: premature end of data segment
Epoch 11/20:  72%|      | 1019/1422 [10:47<04:36,  1.46it/s, loss=0.76]
Corrupt JPEG data: bad Huffman code
Epoch 11/20: 100%|      | 1422/1422 [15:06<00:00,  1.57it/s, loss=0.26]
Validation: 100%|      | 610/610 [04:50<00:00,  2.10it/s, loss=0.0493]

Epoch 11/20 - Train Loss: 0.3686, Val Loss: 0.8205, Val Accuracy: 74.91%

Epoch 12/20:  56%|       | 796/1422 [08:31<06:17,  1.66it/s, loss=0.481]
Corrupt JPEG data: premature end of data segment
Epoch 12/20:  86%|      | 1228/1422 [13:06<02:02,  1.58it/s, loss=0.218]
Corrupt JPEG data: bad Huffman code
Epoch 12/20: 100%|      | 1422/1422 [15:08<00:00,  1.57it/s, loss=0.485]
Validation: 100%|      | 610/610 [04:51<00:00,  2.09it/s, loss=0.00808]

Epoch 12/20 - Train Loss: 0.3261, Val Loss: 0.8793, Val Accuracy: 74.82%

Epoch 13/20:  13%|         | 179/1422 [01:53<14:04,  1.47it/s, loss=0.106]
Corrupt JPEG data: premature end of data segment
Epoch 13/20:  29%|         | 419/1422 [04:24<11:09,  1.50it/s, loss=0.545]
Corrupt JPEG data: bad Huffman code
Epoch 13/20: 100%|      | 1422/1422 [15:05<00:00,  1.57it/s, loss=0.205]
Validation: 100%|      | 610/610 [04:48<00:00,  2.11it/s, loss=0.024]

Epoch 13/20 - Train Loss: 0.2988, Val Loss: 0.8529, Val Accuracy: 75.08%

Epoch 14/20:  23%|         | 328/1422 [03:25<11:18,  1.61it/s, loss=0.1]
Corrupt JPEG data: bad Huffman code
Epoch 14/20:  35%|        | 500/1422 [05:15<11:12,  1.37it/s, loss=0.261]
Corrupt JPEG data: premature end of data segment
Epoch 14/20: 100%|      | 1422/1422 [15:06<00:00,  1.57it/s, loss=0.174]
Validation: 100%|      | 610/610 [04:49<00:00,  2.10it/s, loss=0.00185]

Epoch 14/20 - Train Loss: 0.2672, Val Loss: 0.9576, Val Accuracy: 74.77%

Epoch 15/20:  55%|       | 781/1422 [08:12<06:12,  1.72it/s, loss=0.558]
Corrupt JPEG data: bad Huffman code
Epoch 15/20:  68%|       | 966/1422 [10:12<05:05,  1.49it/s, loss=0.226]
Corrupt JPEG data: premature end of data segment
Epoch 15/20: 100%|      | 1422/1422 [15:07<00:00,  1.57it/s, loss=0.364]
Validation: 100%|      | 610/610 [04:49<00:00,  2.11it/s, loss=0.00409]

Epoch 15/20 - Train Loss: 0.2376, Val Loss: 0.9952, Val Accuracy: 74.33%
```

```
Epoch 16/20:  49%|        | 694/1422 [07:24<08:46,  1.38it/s, loss=0.245]
Corrupt JPEG data: premature end of data segment
Epoch 16/20:  81%|       | 1147/1422 [12:10<03:21,  1.37it/s, loss=0.103]
Corrupt JPEG data: bad Huffman code
Epoch 16/20: 100%|      | 1422/1422 [15:04<00:00,  1.57it/s, loss=0.234]
Validation:  33%|        | 204/610 [01:51<03:41,  1.83it/s, loss=1.56]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[10], line 36
     34 val_bar = tqdm(val_loader, desc="Validation")
     35 for image_paths, labels in val_bar:
---> 36     image_tensors = torch.stack([get_image(image_path) for image_path i
 ↪image_paths])
     37     image_tensors = image_tensors.to('cuda')
     38     labels = labels.to('cuda')

Cell In[10], line 36, in <listcomp>(.0)
     34 val_bar = tqdm(val_loader, desc="Validation")
     35 for image_paths, labels in val_bar:
---> 36     image_tensors = torch.stack([get_image(image_path) for image_path i
 ↪image_paths])
     37     image_tensors = image_tensors.to('cuda')
     38     labels = labels.to('cuda')

Cell In[8], 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:
```

```
Error in callback <bound method _WandbInit._pause_backend of
<wandb.sdk.wandb_init._WandbInit object at 0x7e8191d8c460>> (for post_run_cell):
```

```
---------------------------------------------------------------------------
BrokenPipeError                           Traceback (most recent call last)
File ~/.local/lib/python3.10/site-packages/wandb/sdk/wandb_init.py:565, in
 ↪_WandbInit._pause_backend(self, *args, **kwargs)
    563 if self.backend.interface is not None:
    564     self._logger.info("pausing backend")  # type: ignore
--> 565     self.backend.interface.publish_pause()

File ~/.local/lib/python3.10/site-packages/wandb/sdk/interface/interface.py:769
 ↪in InterfaceBase.publish_pause(self)
```

```
    767 def publish_pause(self) -> None:
    768     pause = pb.PauseRequest()
--> 769     self._publish_pause(pause)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/interface/interface_shared
 ↪py:289, in InterfaceShared._publish_pause(self, pause)
    287 def _publish_pause(self, pause: pb.PauseRequest) -> None:
    288     rec = self._make_request(pause=pause)
--> 289     self._publish(rec)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/interface/interface_sock.p
 ↪39, in InterfaceSock._publish(self, record, local)
     37 def _publish(self, record: "pb.Record", local: Optional[bool] = None) -
 ↪None:
     38     self._assign(record)
---> 39     self._sock_client.send_record_publish(record)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/lib/sock_client.py:174, in
 ↪SockClient.send_record_publish(self, record)
    172 server_req.request_id = record.control.mailbox_slot
    173 server_req.record_publish.CopyFrom(record)
--> 174 self.send_server_request(server_req)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/lib/sock_client.py:154, in
 ↪SockClient.send_server_request(self, msg)
    153 def send_server_request(self, msg: spb.ServerRequest) -> None:
--> 154     self._send_message(msg)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/lib/sock_client.py:151, in
 ↪SockClient._send_message(self, msg)
    149 header = struct.pack("<BI", ord("W"), raw_size)
    150 with self._lock:
--> 151     self._sendall_with_error_handle(header + data)

File ~/.local/lib/python3.10/site-packages/wandb/sdk/lib/sock_client.py:130, in
 ↪SockClient._sendall_with_error_handle(self, data)
    128 start_time = time.monotonic()
    129 try:
--> 130     sent = self._sock.send(data)
    131     # sent equal to 0 indicates a closed socket
    132     if sent == 0:

BrokenPipeError: [Errno 32] Broken pipe
```

```
The Kernel crashed while executing code in the current cell or a previous cell.
```

```
Please review the code in the cell(s) to identify a possible cause of the␣
↪failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```