ml4sci-24-task2-2-resnet

March 26, 2024

[1]:

```
import torch
import numpy as np
import pandas as pd
import pyarrow.parquet as pq
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import timm
import torch
import torch.nn as nn
from torch.utils.data import Dataset,DataLoader, random_split
import torch.nn.functional as F
from torchvision import models

import torch.optim as optim
from tqdm import tqdm

from sklearn.metrics import roc_auc_score, confusion_matrix ,roc_curve
import seaborn as sns
```

[2]:

```
chunk_size = 15

# List of Parquet file paths
file_paths = [
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.
  ↪snappy.parquet',
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.
  ↪snappy.parquet',
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.
  ↪snappy.parquet'
]

# Initialize an empty list to store dataframes
dfs = []

# Loop through each file path
for file_path in file_paths:
    # Create a Parquet file reader object
```

```
    parquet_file = pq.ParquetFile(file_path)

    # Determine the total number of rows in the file
    total_rows = parquet_file.metadata.num_rows

    # Calculate the number of chunks
    num_chunks = total_rows // chunk_size + (1 if total_rows % chunk_size else␣
 ↪0)

    # Loop over the file in chunks
    for chunk_index in range(num_chunks):
        # Read a chunk of rows from the file
        chunk = parquet_file.read_row_group(chunk_index, columns=None)
        df = chunk.to_pandas()

        # Append the DataFrame to the list
        dfs.append(df)

# Concatenate all the DataFrames into a single DataFrame
data = pd.concat(dfs, ignore_index=True)
```

[3]:

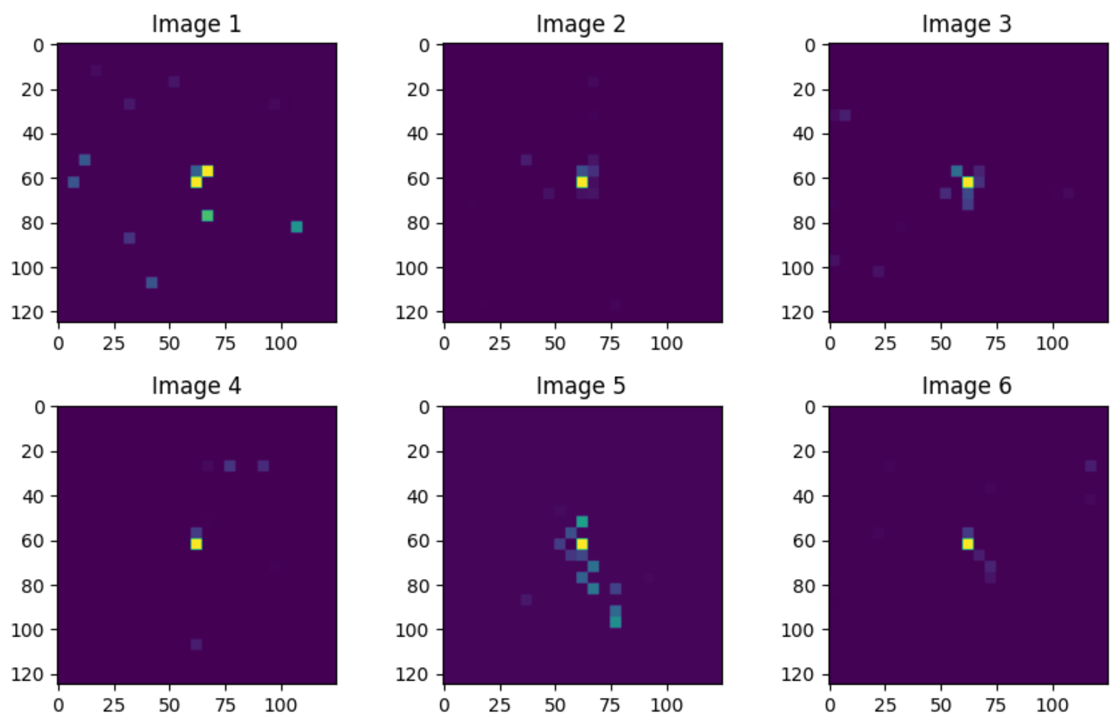
```
def to_3d(arr):
    vishak=[]
    for i in range (0,3):
        vis=np.stack(np.stack(arr)[i],axis=-1)
        vishak.append(vis)
    vishak=np.array(vishak)
    return vishak
```

[4]:

```
data["X_jets"]  = data["X_jets"].apply(to_3d)
```

[5]:

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(10, 6))

# Loop over the axes and image ids, and plot each image on a separate subplot
for i, ax in enumerate(axes.flatten()):
    image = data['X_jets'][i][2,:,:]
    ax.imshow(image)
    ax.set_title(f'Image {i+1}')

# Adjust spacing between subplots
plt.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9, wspace=0.3,␣
 ↪hspace=0.3)

# Show the plot
plt.show()
```



[6]:
```
data.columns
```

[6]: Index(['X_jets', 'pt', 'm0', 'y'], dtype='object')

[7]:
```
# data['y']
```

[8]:
```
class task2Dataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        # Assuming 'X_jets' column contains paths to images or actual image data
        X = self.dataframe.iloc[idx]['X_jets']
        mean = X.mean(axis=(0, 1, 2), keepdims=True)
        std = X.std(axis=(0, 1, 2), keepdims=True)

        # Normalize each channel separately
        X = (X - mean) / std
        y = self.dataframe.iloc[idx]['y']
```

```
        if self.transform:
            X = self.transform(X)

        # Convert X and y to PyTorch tensors
        X_tensor = torch.tensor(X, dtype=torch.float)
        y_tensor = torch.tensor(y, dtype=torch.long)

        return X_tensor, y_tensor
```

[9]:
```
jet_dataset = task2Dataset(dataframe=data)


train_dataset, val_dataset = train_test_split(jet_dataset, test_size=0.2,␣
↪random_state=42)


train_loader = DataLoader(dataset=train_dataset, batch_size=256, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)
```

[10]:
```
next(iter(train_loader))[0].shape
```

[10]:
```
torch.Size([256, 3, 125, 125])
```

[11]:
```
class CustomResNet(nn.Module):
    def __init__(self, num_classes=2, pretrained=True):
        super(CustomResNet, self).__init__()

        self.model = timm.create_model('resnet50', pretrained=pretrained,␣
↪num_classes=num_classes)


    def forward(self, x):
        return self.model(x)

# Initialize your model
model = CustomResNet(num_classes=2, pretrained=True)

# Print your model architecture
print(model)
```

```
model.safetensors:   0%|          | 0.00/102M [00:00<?, ?B/s]
```

```
CustomResNet(
  (model): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (act1): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
```

```
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
    )
    (layer2): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
```

```
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (3): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
    )
    (layer3): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
```

```
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
```

```
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (3): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (4): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (5): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
    )
    (layer4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

```
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act1): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (drop_block): Identity()
        (act2): ReLU(inplace=True)
        (aa): Identity()
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (act3): ReLU(inplace=True)
      )
    )
    (global_pool): SelectAdaptivePool2d(pool_type=avg,
flatten=Flatten(start_dim=1, end_dim=-1))
    (fc): Linear(in_features=2048, out_features=2, bias=True)
  )
)
```

[12]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")



model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
```

[13]:

```
num_epochs = 20
train_losses, val_losses, val_accuracies = [], [], []
best_loss = 100000
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    train_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Train]␣
↪Loss: 0.0000", leave=False)
    for inputs, labels in train_bar:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        train_bar.set_description(f"Epoch {epoch+1}/{num_epochs} [Train] Loss:␣
↪{loss.item():.4f}")

    #scheduler.step()

    epoch_loss = running_loss / len(train_loader.dataset)
    train_losses.append(epoch_loss)

    # Validation phase
    model.eval()
    val_running_loss = 0.0
    correct_predictions = 0
    total_predictions = 0
    val_bar = tqdm(val_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Val] Loss:␣
↪0.0000, Acc: 0.0000", leave=True)

    with torch.no_grad():
        for inputs, labels in val_bar:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item() * inputs.size(0)

            _, predicted = torch.max(outputs, 1)
            correct_predictions += (predicted == labels).sum().item()
            total_predictions += labels.size(0)
```

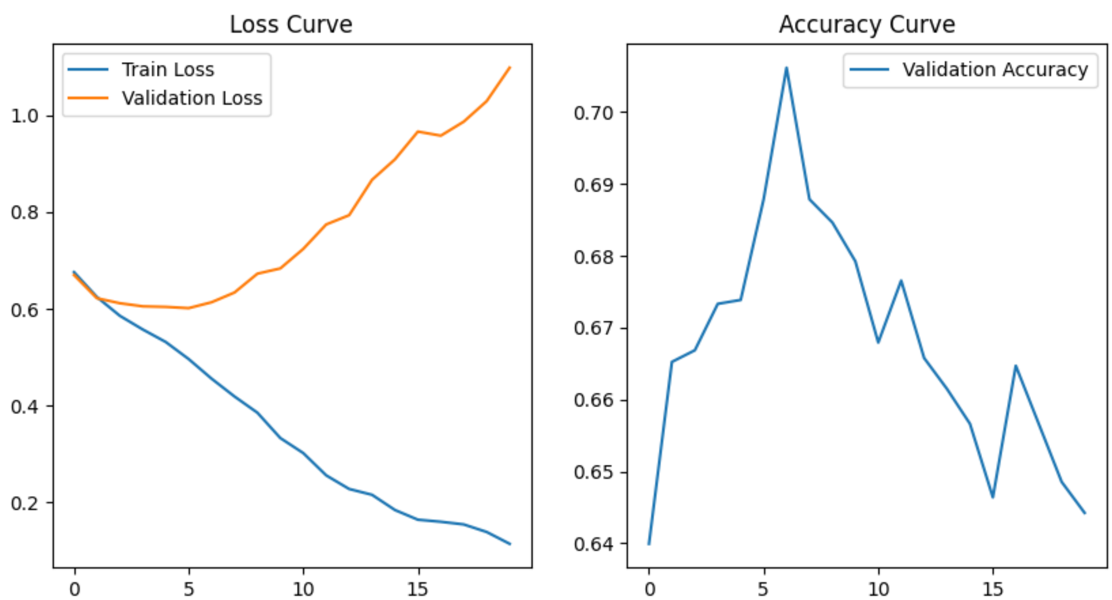
```
            val_bar.set_description(f"Epoch {epoch+1}/{num_epochs} [Val] Loss:␣
↪{loss.item():.4f}, Acc: {correct_predictions/total_predictions:.4f}")

    epoch_val_loss = val_running_loss / len(val_loader.dataset)
    val_losses.append(epoch_val_loss)

    epoch_val_accuracy = correct_predictions / total_predictions
    best_loss = min(epoch_val_loss , best_loss)
    val_accuracies.append(epoch_val_accuracy)

    if(epoch_val_loss== best_loss):

            model_path = f"model_weights_{epoch}.pth"
            torch.save(model.state_dict(), model_path)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_loss:.4f}, Val␣
↪Loss: {epoch_val_loss:.4f}, Val Accuracy: {epoch_val_accuracy:.4f}")
```

```
Epoch 1/20 [Val] Loss: 0.6346, Acc: 0.6399: 100%|     | 59/59 [00:01<00:00,
40.07it/s]

Epoch 1/20, Train Loss: 0.6760, Val Loss: 0.6700, Val Accuracy: 0.6399

Epoch 2/20 [Val] Loss: 0.4195, Acc: 0.6652: 100%|     | 59/59 [00:01<00:00,
45.04it/s]

Epoch 2/20, Train Loss: 0.6245, Val Loss: 0.6220, Val Accuracy: 0.6652

Epoch 3/20 [Val] Loss: 0.3609, Acc: 0.6668: 100%|     | 59/59 [00:01<00:00,
45.13it/s]

Epoch 3/20, Train Loss: 0.5857, Val Loss: 0.6117, Val Accuracy: 0.6668

Epoch 4/20 [Val] Loss: 0.2964, Acc: 0.6733: 100%|     | 59/59 [00:01<00:00,
44.97it/s]

Epoch 4/20, Train Loss: 0.5575, Val Loss: 0.6053, Val Accuracy: 0.6733

Epoch 5/20 [Val] Loss: 0.2632, Acc: 0.6738: 100%|     | 59/59 [00:01<00:00,
44.99it/s]

Epoch 5/20, Train Loss: 0.5317, Val Loss: 0.6042, Val Accuracy: 0.6738

Epoch 6/20 [Val] Loss: 0.2085, Acc: 0.6878: 100%|     | 59/59 [00:01<00:00,
44.90it/s]

Epoch 6/20, Train Loss: 0.4964, Val Loss: 0.6016, Val Accuracy: 0.6878

Epoch 7/20 [Val] Loss: 0.1266, Acc: 0.7061: 100%|     | 59/59 [00:01<00:00,
44.86it/s]

Epoch 7/20, Train Loss: 0.4559, Val Loss: 0.6137, Val Accuracy: 0.7061
```

```
Epoch 8/20 [Val] Loss: 0.2911, Acc: 0.6878: 100%|     | 59/59 [00:01<00:00,
44.96it/s]

Epoch 8/20, Train Loss: 0.4190, Val Loss: 0.6339, Val Accuracy: 0.6878

Epoch 9/20 [Val] Loss: 0.2875, Acc: 0.6846: 100%|     | 59/59 [00:01<00:00,
44.86it/s]

Epoch 9/20, Train Loss: 0.3855, Val Loss: 0.6728, Val Accuracy: 0.6846

Epoch 10/20 [Val] Loss: 0.2499, Acc: 0.6792: 100%|     | 59/59
[00:01<00:00, 44.75it/s]

Epoch 10/20, Train Loss: 0.3331, Val Loss: 0.6837, Val Accuracy: 0.6792

Epoch 11/20 [Val] Loss: 0.2271, Acc: 0.6679: 100%|     | 59/59
[00:01<00:00, 44.91it/s]

Epoch 11/20, Train Loss: 0.3021, Val Loss: 0.7240, Val Accuracy: 0.6679

Epoch 12/20 [Val] Loss: 0.2057, Acc: 0.6765: 100%|     | 59/59
[00:01<00:00, 44.90it/s]

Epoch 12/20, Train Loss: 0.2560, Val Loss: 0.7745, Val Accuracy: 0.6765

Epoch 13/20 [Val] Loss: 0.0503, Acc: 0.6658: 100%|     | 59/59
[00:01<00:00, 44.91it/s]

Epoch 13/20, Train Loss: 0.2279, Val Loss: 0.7936, Val Accuracy: 0.6658

Epoch 14/20 [Val] Loss: 0.0907, Acc: 0.6615: 100%|     | 59/59
[00:01<00:00, 44.32it/s]

Epoch 14/20, Train Loss: 0.2159, Val Loss: 0.8668, Val Accuracy: 0.6615

Epoch 15/20 [Val] Loss: 0.2938, Acc: 0.6566: 100%|     | 59/59
[00:01<00:00, 44.91it/s]

Epoch 15/20, Train Loss: 0.1844, Val Loss: 0.9091, Val Accuracy: 0.6566

Epoch 16/20 [Val] Loss: 0.2356, Acc: 0.6464: 100%|     | 59/59
[00:01<00:00, 44.75it/s]

Epoch 16/20, Train Loss: 0.1644, Val Loss: 0.9664, Val Accuracy: 0.6464

Epoch 17/20 [Val] Loss: 0.2341, Acc: 0.6647: 100%|     | 59/59
[00:01<00:00, 44.98it/s]

Epoch 17/20, Train Loss: 0.1602, Val Loss: 0.9580, Val Accuracy: 0.6647

Epoch 18/20 [Val] Loss: 0.0874, Acc: 0.6566: 100%|     | 59/59
[00:01<00:00, 44.73it/s]

Epoch 18/20, Train Loss: 0.1547, Val Loss: 0.9869, Val Accuracy: 0.6566

Epoch 19/20 [Val] Loss: 0.0967, Acc: 0.6485: 100%|     | 59/59
[00:01<00:00, 44.80it/s]

Epoch 19/20, Train Loss: 0.1390, Val Loss: 1.0294, Val Accuracy: 0.6485
```

```
Epoch 20/20 [Val] Loss: 0.5077, Acc: 0.6442: 100%|          | 59/59
[00:01<00:00, 44.25it/s]

Epoch 20/20, Train Loss: 0.1146, Val Loss: 1.0984, Val Accuracy: 0.6442
```

[14]:
```python
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.legend()
plt.title('Loss Curve')

plt.subplot(1, 2, 2)
plt.plot(val_accuracies, label='Validation Accuracy')
plt.legend()
plt.title('Accuracy Curve')
plt.show()
```



[ ]:

[ ]: