modified_model

April 25, 2025

1 Doing Normalization and Setting up The batch size.

[1]:
```
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader

# Define normalization transformations
val_transform = transforms.Compose([
    transforms.ToTensor(),  # Convert image to tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225]),  # Normalize
])

# Load datasets with normalization (no augmentation)
train_dataset = ImageFolder("processed_data/train", transform=val_transform)
val_dataset = ImageFolder("processed_data/val", transform=val_transform)
test_dataset = ImageFolder("processed_data/test", transform=val_transform)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# Verify dataset sizes
print(f"Training dataset size: {len(train_dataset)} images")
print(f"Validation dataset size: {len(val_dataset)} images")
print(f"Test dataset size: {len(test_dataset)} images")
```

```
Training dataset size: 4800 images
Validation dataset size: 600 images
Test dataset size: 600 images
```

2 Define the Modified EfficientNet-B0 with Seperabale Convolution model

[2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import timm
import os
```

[10]:
```
import torch
import torch.nn as nn
import timm

class EfficientNetWithSepConv(nn.Module):
    def __init__(self, num_classes=4):
        super().__init__()
        # Load the base model
        self.base = timm.create_model("efficientnet_b0", pretrained=True,␣
↪features_only=False)

        # Extract the features before the classifier
        self.base.classifier = nn.Identity()
        self.base.global_pool = nn.Identity()

        # Define input features for the separable convolution
        in_features = 1280

        self.sep_conv = nn.Sequential(
            nn.Conv2d(in_features, in_features, kernel_size=3,␣
↪groups=in_features, padding=1),
            nn.Conv2d(in_features, 256, kernel_size=1),
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.AdaptiveAvgPool2d(1),
        )
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        # Use the forward method without accessing .features
        x = self.base(x)
        # At this point, x should be the output from the EfficientNet's feature␣
↪extractor
        x = self.sep_conv(x)
        x = x.view(x.size(0), -1)
```

```
        x = self.fc(x)
        return x

print("Done Making the model")
```

```
Done Making the model
```

[11]:
```
# Training setup model 1
import torch.optim as optim
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = EfficientNetWithSepConv(num_classes=4).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

print("DONE")
```

```
DONE
```

[12]:
```
#model 1
print("Starting")
num_epochs = 20
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")

    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f"Validation Accuracy: {accuracy:.2f}%")
```

```
Starting
```

```
Epoch 1, Loss: 0.1569676517229527
Validation Accuracy: 98.67%
Epoch 2, Loss: 0.029325566313927992
Validation Accuracy: 99.33%
Epoch 3, Loss: 0.030279114949322925
Validation Accuracy: 99.67%
Epoch 4, Loss: 0.04014143187591496
Validation Accuracy: 99.83%
Epoch 5, Loss: 0.022460392838305174
Validation Accuracy: 100.00%
Epoch 6, Loss: 0.06556297233522249
Validation Accuracy: 99.83%
Epoch 7, Loss: 0.021629659482471954
Validation Accuracy: 100.00%
Epoch 8, Loss: 0.017802004612070352
Validation Accuracy: 99.00%
Epoch 9, Loss: 0.011953749310535689
Validation Accuracy: 100.00%
Epoch 10, Loss: 0.01038636947053116
Validation Accuracy: 99.83%
Epoch 11, Loss: 0.013526884691988622
Validation Accuracy: 99.83%
Epoch 12, Loss: 0.006001068068095871
Validation Accuracy: 100.00%
Epoch 13, Loss: 0.00175331908321823
Validation Accuracy: 100.00%
Epoch 14, Loss: 0.017002984463360918
Validation Accuracy: 99.83%
Epoch 15, Loss: 0.01136660850905173
Validation Accuracy: 100.00%
Epoch 16, Loss: 0.0038979652630829757
Validation Accuracy: 100.00%
Epoch 17, Loss: 0.009245189986019493
Validation Accuracy: 99.50%
Epoch 18, Loss: 0.024024466557420965
Validation Accuracy: 99.83%
Epoch 19, Loss: 0.013796991609512284
Validation Accuracy: 100.00%
Epoch 20, Loss: 0.011391211668766724
Validation Accuracy: 100.00%
```

[13]:
```python
# Save Bare EfficientNet-B0 model
torch.save(model.state_dict(), 'efficientnet_b0main_modified.pth')
print("Model saved as 'efficientnet_b0main_modified.pth'")
```

```
Model saved as 'efficientnet_b0main_modified.pth'
```

[21]:
```python
import torch

# Assuming `model` is your trained or defined model
num_params = sum(p.numel() for p in model.parameters())
print(f"Total number of parameters: {num_params}")
```

```
Total number of parameters: 4349824
```

2.1 2nd Version of Modified Model with Less Parameters

[19]:
```python
# Define the EfficientNetWithMinimalHead model class
class EfficientNetWithMinimalHead(nn.Module):
    def __init__(self, num_classes=4):
        super().__init__()
        self.base = timm.create_model("efficientnet_b0", pretrained=False,␣
 ↪features_only=False)
        self.base.classifier = nn.Identity()
        self.base.global_pool = nn.Identity()
        in_features = 1280
        self.head = nn.Sequential(
            nn.Conv2d(in_features, 4, kernel_size=1),
            nn.ReLU(),
            nn.BatchNorm2d(4),
            nn.AdaptiveAvgPool2d(1),
            nn.Dropout(0.3),
        )
        self.fc = nn.Linear(4, num_classes)

    def forward(self, x):
        x = self.base(x)
        x = self.head(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

[17]:
```python
# Training setup model 2
import torch.optim as optim
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_2 = EfficientNetWithMinimalHead(num_classes=4).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_2.parameters(), lr=0.001)

print("DONE")

# model 2
print("Starting")
num_epochs = 20
for epoch in range(num_epochs):
```

```
    model_2.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model_2(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")

    # Validation
    model_2.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model_2(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f"Validation Accuracy: {accuracy:.2f}%")
```

```
Starting
Epoch 1, Loss: 0.11815237798417608
Validation Accuracy: 99.50%
Epoch 2, Loss: 0.0385976266716898
Validation Accuracy: 99.83%
Epoch 3, Loss: 0.005807919433002553
Validation Accuracy: 99.83%
Epoch 4, Loss: 0.25389040205440416
Validation Accuracy: 97.67%
Epoch 5, Loss: 0.07790098130547753
Validation Accuracy: 99.50%
Epoch 6, Loss: 0.10045307031056533
Validation Accuracy: 99.50%
Epoch 7, Loss: 0.266709964571055
Validation Accuracy: 99.17%
Epoch 8, Loss: 0.14438435910269617
Validation Accuracy: 99.33%
Epoch 9, Loss: 0.04119399467715994
Validation Accuracy: 99.33%
Epoch 10, Loss: 0.023472650517942385
Validation Accuracy: 99.33%
Epoch 11, Loss: 0.02186845612168933
```

```
Validation Accuracy: 99.67%
Epoch 12, Loss: 0.02108318036000128
Validation Accuracy: 99.67%
Epoch 13, Loss: 0.01568641162807277
Validation Accuracy: 99.67%
Epoch 14, Loss: 0.014936921587407899
Validation Accuracy: 99.67%
Epoch 15, Loss: 0.015281060402727841
Validation Accuracy: 99.00%
Epoch 16, Loss: 0.03191355090355501
Validation Accuracy: 99.50%
Epoch 17, Loss: 0.02544845105653318
Validation Accuracy: 99.17%
Epoch 18, Loss: 0.04532601369020995
Validation Accuracy: 99.67%
Epoch 19, Loss: 0.024061461928649806
Validation Accuracy: 99.67%
Epoch 20, Loss: 0.6398472436846351
Validation Accuracy: 25.67%
```

[ ]:
```python
# Save Bare EfficientNet-B0 model
torch.save(model_2.state_dict(), 'efficientnet_b0_2_modified.pth')
print("Model saved as 'efficientnet_b0_2_modified.pth'")
```

[ ]:
```python

```