resnet50todebug

December 3, 2025

[ ]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split, Dataset
import os
```

[ ]:
```
np.random.seed(0)
```

[ ]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
elif torch.cuda.is_available():
    device = torch.device("cpu")
    use_mps = False

print(device)
```

[ ]:
```
from PIL import Image

class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None):
        with open(file_path, 'rb') as f:
            data = pickle.load(f)
        self.images = data['images']
        self.labels = data['labels'].reshape(-1)
        self.transform = transform

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

    def __getitem__(self, idx):
        image = self.images[idx]
```

```
        label = self.labels[idx]

        image = Image.fromarray(image.astype('uint8'))

        if self.transform:
            image = self.transform(image)
        return image, label
```

[ ]:
```
dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
↪pkl')
```

[ ]:
```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[ ]:
```
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

[ ]:
```
from sklearn.model_selection import train_test_split

labels = dataset.labels
idx = np.arange(len(dataset))
train_idx, valid_idx = train_test_split(idx, test_size=0.2, stratify=labels,␣
↪random_state=42)
```

[ ]:
```
class TransformSubset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, idx):
        image, label = self.subset[idx]
        if self.transform:
            image = self.transform(image)
        return image, label
```

```
    def __len__(self):
        return len(self.subset)
```

[ ]:
```
from torch.utils.data import Subset

train_dataset = Subset(dataset, train_idx)
train_data = TransformSubset(train_dataset, train_transform)

val_dataset = Subset(dataset, valid_idx)
val_data = TransformSubset(val_dataset, val_transform)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True,␣
 ↪pin_memory=True)
val_loader = DataLoader(val_data, batch_size=32, shuffle=False, pin_memory=True)
```

[ ]:
```
model = models.resnet50(weights="IMAGENET1K_V2")
```

[ ]:
```
num_classes = 5

model.fc = nn.Linear(model.fc.in_features, num_classes)
```

[ ]:
```
for param in model.parameters():
    param.requires_grad = False

for param in model.fc.parameters():
    param.requires_grad = True

for param in model.layer4.parameters():
    param.requires_grad = True
```

[ ]:
```
classes = np.unique(dataset.labels)
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight('balanced', classes=classes, y=dataset.
 ↪labels)
weights_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)
```

[ ]:
```
criterion = nn.CrossEntropyLoss(weight=weights_tensor)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),␣
 ↪lr=1e-4)
```

[ ]:
```
for epoch in range(20):
    model.train()

    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()

    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

[ ]:
```
model.eval()

all_preds = []
all_labels = []

from sklearn.metrics import accuracy_score, recall_score,␣
 ↪balanced_accuracy_score, classification_report, confusion_matrix

with torch.no_grad():
    for images, labels in val_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())


all_preds = np.array(all_preds)
all_labels = np.array(all_labels)
```

[ ]:
```
bal_acc = balanced_accuracy_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds, average='macro')
acc = accuracy_score(all_labels, all_preds)

print(f"Validation Balanced Accuracy: {bal_acc:.4f}")
print(f"Validation Recall: {recall:.4f}")
print(f"Validation Accuracy: {acc:.4f}")
print(classification_report(all_labels, all_preds, digits=4))
```

[ ]:
```
import seaborn as sns

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

[ ]:
```
torch.save({
    "model_state_dict": model.state_dict(),
    "class_to_idx": train_dataset.dataset.labels,
}, "resnet50_finetuned.pth")
```

[ ]:
```
model = model.ResNet50(weights=None)
num_classes = 5
model.fc = nn.Linear(model.fc.in_features, num_classes)

checkpoint = torch.load("resnet50_finetuned.pth", map_location=device)
model.load_state_dict(checkpoint["model_state_dict"])

model.eval()
```

[ ]:
```
test_dataset = pickle.load(open('ift-3395-6390-kaggle-2-competition-fall-2025/
 ↪test_data.pkl', 'rb'))
test_images = test_dataset['images']

test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

[ ]:
```
class TestPKLDataset(Dataset):
    def __init__(self, images, transform=None):
        self.images = images
        self.transform = transform

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

    def __getitem__(self, idx):
        image = self.images[idx]
        image = Image.fromarray(image.astype('uint8'))
        if self.transform:
            image = self.transform(image)
        return image
```

[ ]:
```
test_ds = TestPKLDataset(test_images, transform=test_transform)
test_loader = DataLoader(test_ds, batch_size=32, shuffle=False, pin_memory=True)
preds = []

with torch.no_grad():
    for images in test_loader:
        outputs = model(images)
```

```
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.cpu().numpy())

df = pd.DataFrame({
    "Id": np.arange(len(preds)),
    "Label": preds
})

df.to_csv("IFT3395_YAPS_MCSV50.csv", index=False)
```