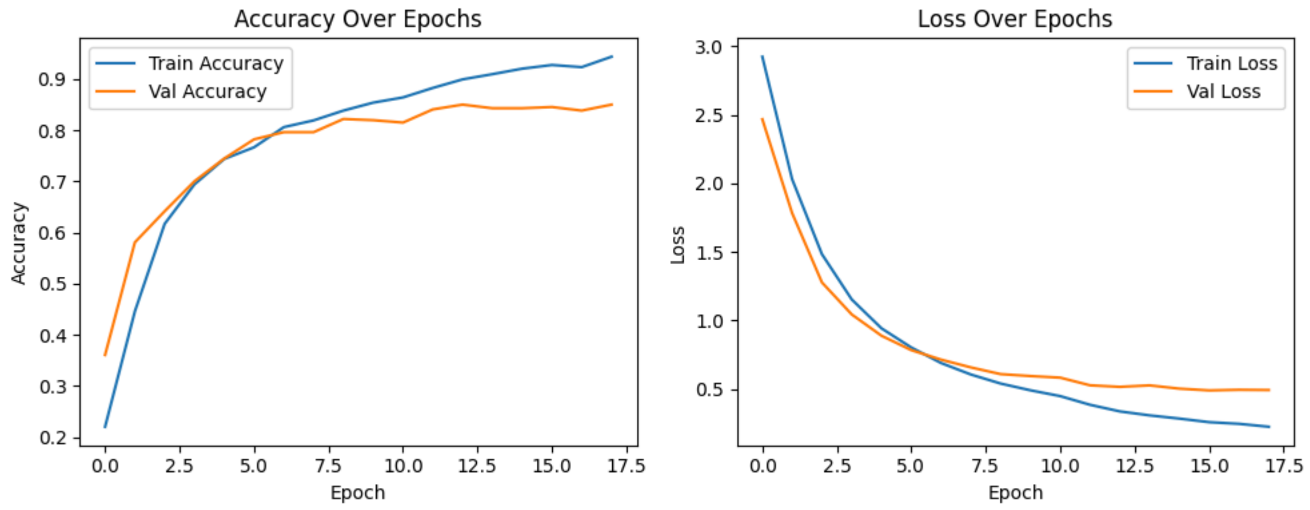
```
# train_model_full_set.py
# ---------------------------------------------------------
# Trains a ResNet50 image classification model on refund data.
# Logs training and validation metrics using MLflow for tracking.
# Saves the final model as refund_classifier_final.pt for the API.
# Designed for reproducible and scalable model development.
# ---------------------------------------------------------

import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
from torchvision.models import resnet50, ResNet50_Weights
from torch.utils.data import DataLoader
from tqdm import tqdm
import mlflow
import mlflow.pytorch
from collections import Counter

# --- CONFIGURATION ---
EPOCHS = 30
LR = 0.001
BATCH_SIZE = 32
MODEL_PATH = "refund_classifier_final.pt"
DATA_DIR = "/content/data/full"

# --- TRANSFORMS ---
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([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# --- DATA LOADERS ---
train_data = datasets.ImageFolder(os.path.join(DATA_DIR, 'train'), transform=train_transform)
val_data = datasets.ImageFolder(os.path.join(DATA_DIR, 'val'), transform=val_transform)
train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_data, batch_size=BATCH_SIZE)

print("Train class distribution:")
train_labels = [label for _, label in train_data.imgs]
print(Counter(train_labels))

print("Val class distribution:")
```

```
val_labels = [label for _, label in val_data.imgs]
print(Counter(val_labels))

# --- MODEL SETUP ---
model = resnet50(weights=ResNet50_Weights.DEFAULT)
model.fc = nn.Sequential(
    nn.Dropout(0.3),
    nn.Linear(model.fc.in_features, len(train_data.classes))
)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LR, momentum=0.9)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.5, verbose=True)

# --- MLflow Tracking ---
mlflow.set_experiment("Refund_Classifier")
with mlflow.start_run():
    mlflow.log_params({
        "epochs": EPOCHS,
        "learning_rate": LR,
        "batch_size": BATCH_SIZE,
        "optimizer": "SGD",
        "model": "resnet50"
    })

    best_val_acc = 0
    patience = 5
    trigger = 0

    for epoch in range(EPOCHS):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for inputs, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{EPOCHS}"):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = correct / total

        # Evaluate
        model.eval()
        val_loss = 0.0
        val_correct = 0
        val_total = 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = outputs.max(1)
                val_total += labels.size(0)
                val_correct += predicted.eq(labels).sum().item()

        val_loss /= len(val_loader)
        val_acc = val_correct / val_total

        print(f"Epoch {epoch+1}/{EPOCHS} - Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # Log to MLflow
        mlflow.log_metric("train_loss", train_loss, step=epoch)
        mlflow.log_metric("train_acc", train_acc, step=epoch)
        mlflow.log_metric("val_loss", val_loss, step=epoch)
        mlflow.log_metric("val_acc", val_acc, step=epoch)

        # Learning rate adjustment
        scheduler.step(val_loss)

        # Early stopping
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            trigger = 0
        else:
            trigger += 1

        if trigger >= patience:
            print("Early stopping triggered.")
            break

# Save model + log artifact
torch.save(model, MODEL_PATH)
mlflow.pytorch.log_model(model, "model")


# Save to Google Drive
from google.colab import drive
drive.mount('/content/drive')
torch.save(model, "/content/drive/MyDrive/refund_classifier_final.pt")

# Download model directly to local PC
from google.colab import files
files.download(MODEL_PATH)
```

```
Train class distribution:
Counter({28: 340, 17: 325, 13: 147, 29: 140, 3: 105, 25: 91, 8: 68, 12: 62, 9: 52, 7: 46, 16: 42, 21: 34, 27: 31, 14: 23, 10: 22, 22: 20, 20: 18, 23: 16, 6: 14, 24: 11, 26: 11, 19: 10, 2: 9, 15:
Val class distribution:
Counter({28: 89, 17: 75, 13: 39, 29: 32, 3: 24, 12: 20, 25: 20, 8: 16, 16: 14, 9: 13, 7: 10, 14: 10, 10: 8, 21: 8, 27: 8, 20: 6, 22: 5, 24: 5, 4: 4, 0: 3, 6: 3, 15: 3, 26: 3, 2: 2, 23: 2, 1: 1, 5
Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to /root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 261MB/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
2025/07/15 00:15:44 INFO mlflow.tracking.fluent: Experiment with name 'Refund_Classifier' does not exist. Creating a new experiment.
Epoch 1/30: 100%|██████████| 53/53 [04:27<00:00,  5.04s/it]
Epoch 1/30 - Train Acc: 0.2202, Val Acc: 0.3607
Epoch 2/30: 100%|██████████| 53/53 [04:22<00:00,  4.95s/it]
Epoch 2/30 - Train Acc: 0.4458, Val Acc: 0.5808
Epoch 3/30: 100%|██████████| 53/53 [04:01<00:00,  4.56s/it]
Epoch 3/30 - Train Acc: 0.6170, Val Acc: 0.6417
Epoch 4/30: 100%|██████████| 53/53 [04:01<00:00,  4.57s/it]
Epoch 4/30 - Train Acc: 0.6942, Val Acc: 0.7002
Epoch 5/30: 100%|██████████| 53/53 [03:58<00:00,  4.50s/it]
Epoch 5/30 - Train Acc: 0.7439, Val Acc: 0.7447
Epoch 6/30: 100%|██████████| 53/53 [04:01<00:00,  4.55s/it]
Epoch 6/30 - Train Acc: 0.7666, Val Acc: 0.7822
Epoch 7/30: 100%|██████████| 53/53 [03:58<00:00,  4.49s/it]
Epoch 7/30 - Train Acc: 0.8061, Val Acc: 0.7963
Epoch 8/30: 100%|██████████| 53/53 [04:02<00:00,  4.58s/it]
Epoch 8/30 - Train Acc: 0.8193, Val Acc: 0.7963
Epoch 9/30: 100%|██████████| 53/53 [04:01<00:00,  4.55s/it]
Epoch 9/30 - Train Acc: 0.8384, Val Acc: 0.8220
Epoch 10/30: 100%|██████████| 53/53 [03:59<00:00,  4.52s/it]
Epoch 10/30 - Train Acc: 0.8540, Val Acc: 0.8197
Epoch 11/30: 100%|██████████| 53/53 [04:00<00:00,  4.54s/it]
Epoch 11/30 - Train Acc: 0.8642, Val Acc: 0.8150
Epoch 12/30: 100%|██████████| 53/53 [04:00<00:00,  4.55s/it]
Epoch 12/30 - Train Acc: 0.8827, Val Acc: 0.8407
Epoch 13/30: 100%|██████████| 53/53 [04:00<00:00,  4.54s/it]
Epoch 13/30 - Train Acc: 0.8995, Val Acc: 0.8501
Epoch 14/30: 100%|██████████| 53/53 [04:01<00:00,  4.56s/it]
Epoch 14/30 - Train Acc: 0.9096, Val Acc: 0.8431
Epoch 15/30: 100%|██████████| 53/53 [04:03<00:00,  4.60s/it]
Epoch 15/30 - Train Acc: 0.9204, Val Acc: 0.8431
Epoch 16/30: 100%|██████████| 53/53 [03:59<00:00,  4.53s/it]
Epoch 16/30 - Train Acc: 0.9276, Val Acc: 0.8454
Epoch 17/30: 100%|██████████| 53/53 [03:57<00:00,  4.48s/it]
Epoch 17/30 - Train Acc: 0.9234, Val Acc: 0.8384
Epoch 18/30: 100%|██████████| 53/53 [04:02<00:00,  4.58s/it]
2025/07/15 01:33:48 WARNING mlflow.models.model: `artifact_path` is deprecated. Please use `name` instead.
Epoch 18/30 - Train Acc: 0.9437, Val Acc: 0.8501
Early stopping triggered.
2025/07/15 01:33:48 WARNING mlflow.utils.requirements_utils: Found torch version (2.6.0+cu124) contains a local version label (+cu124). MLflow logged a pip requirement for this package as 'torch==
2025/07/15 01:33:56 WARNING mlflow.utils.requirements_utils: Found torchvision version (0.21.0+cu124) contains a local version label (+cu124). MLflow logged a pip requirement for this package as
2025/07/15 01:33:56 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

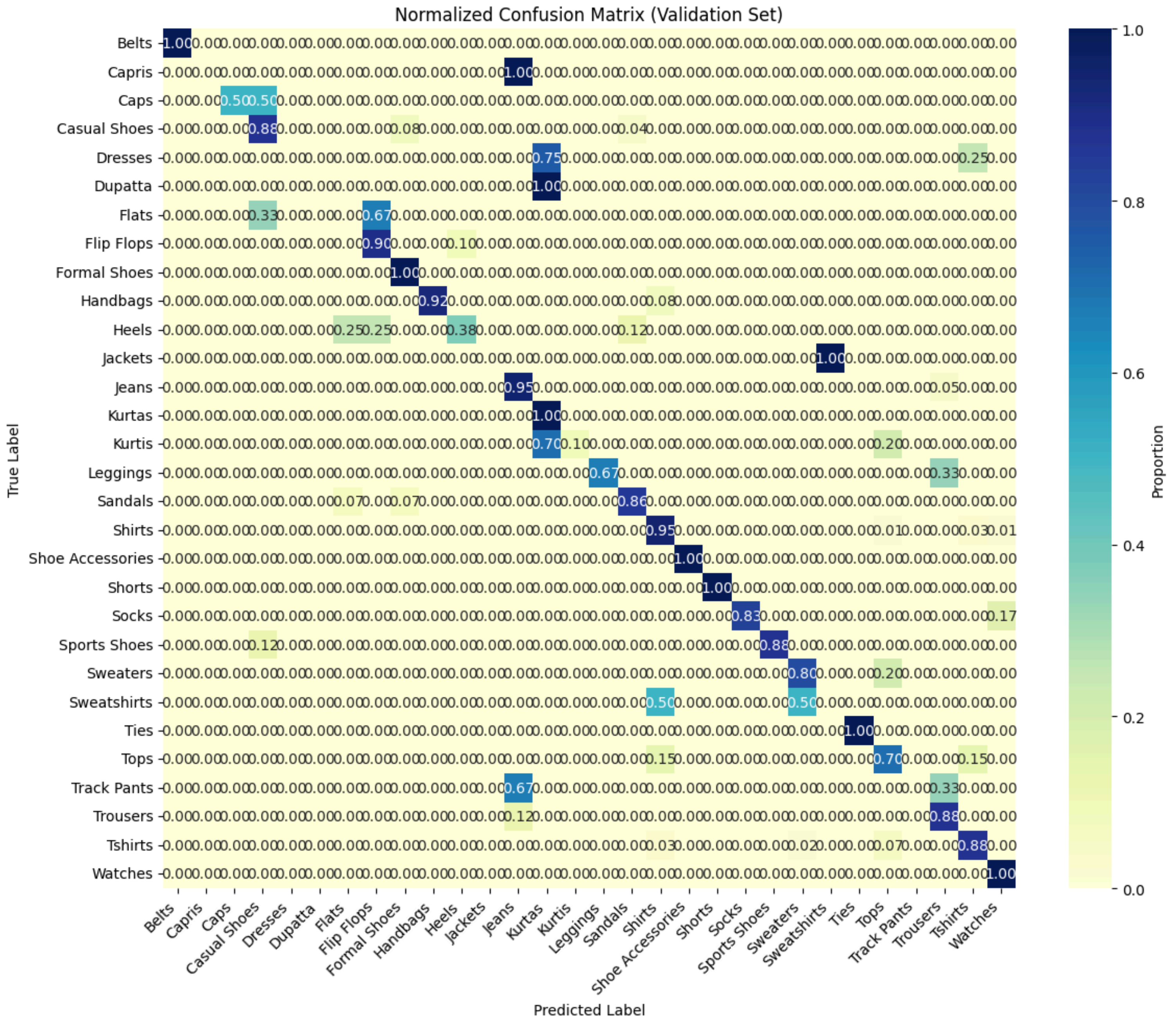
```
# visualize_training_results.py
# --------------------------------------------------
# Script to extract and plot accuracy and loss
# from MLflow logs after training is complete.
# ---------------------------------------------------

import mlflow
import pandas as pd
import matplotlib.pyplot as plt

# Load latest MLflow run (now that mlruns is available)
client = mlflow.tracking.MlflowClient()
experiment = client.get_experiment_by_name("Refund_Classifier")
runs = client.search_runs(experiment.experiment_id)

# Just use the first available run
run = runs[0]
run_id = run.info.run_id

# Retrieve metrics
metrics = client.get_metric_history(run_id, "train_acc")
train_acc = [m.value for m in metrics]
train_steps = [m.step for m in metrics]

metrics = client.get_metric_history(run_id, "val_acc")
val_acc = [m.value for m in metrics]

metrics = client.get_metric_history(run_id, "train_loss")
train_loss = [m.value for m in metrics]

metrics = client.get_metric_history(run_id, "val_loss")
val_loss = [m.value for m in metrics]

# Plot
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(train_steps, train_acc, label="Train Accuracy")
plt.plot(train_steps, val_acc, label="Val Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy Over Epochs")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_steps, train_loss, label="Train Loss")
plt.plot(train_steps, val_loss, label="Val Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Over Epochs")
plt.legend()

plt.tight_layout()
plt.show()
```



```
print("Train acc:", train_acc)
print("Val acc:", val_acc)
```

```
Train acc: [0.22022740873728305, 0.44584081388390184, 0.6169958108916817, 0.6941950927588271, 0.743865948533812, 0.7666068222621185, 0.8061041292639138, 0.8192698982645122, 0.8384201077199281, 0.8
Val acc: [0.36065573770491804, 0.5807962529274004, 0.6416861826697893, 0.7002341920374707, 0.7447306791569087, 0.7822014051522248, 0.7962529274004684, 0.7962529274004684, 0.8220140515222483, 0.819
```

```
# confusion_matrix_heatmap.py
# --------------------------------------------------
# Script to generate a normalized confusion matrix
# for the trained refund classification model using
# the validation set. Highlights true vs. predicted
# class performance with a heatmap.
# --------------------------------------------------

import torch
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from torchvision import transforms

# --- Load final model ---
model = torch.load("refund_classifier_final.pt")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# --- Load validation data ---
val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

val_data = ImageFolder("data/full/val", transform=val_transform)
val_loader = DataLoader(val_data, batch_size=32, shuffle=False)
class_names = val_data.classes

# --- Predict on validation set ---
all_preds = []
all_labels = []

with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# --- Compute normalized confusion matrix ---
cm = confusion_matrix(all_labels, all_preds, normalize="true")

# --- Plot confusion matrix as heatmap ---
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt=".2f", cmap="YlGnBu",
            xticklabels=class_names,
            yticklabels=class_names,
            cbar_kws={"label": "Proportion"})
plt.title("Normalized Confusion Matrix (Validation Set)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```


Normalized Confusion Matrix (Validation Set)
True Label
Predicted Label
Proportion
Belts
Capris
Caps
Casual Shoes
Dresses
Dupatta
Flats
Flip Flops
Formal Shoes
Handbags
Heels
Jackets
Jeans
Kurtas
Kurtis
Leggings
Sandals
Shirts
Shoe Accessories
Shorts
Socks
Sports Shoes
Sweaters
Sweatshirts
Ties
Tops
Track Pants
Trousers
Tshirts
Watches
0.0
0.2
0.4
0.6
0.8
1.0