Multimodal Housing Price Prediction Using Tabular + Image Data

```
import os
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import models, transforms
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# 1. Example Dataset
data = pd.DataFrame({
    "image_path": ["house1.jpg", "house2.jpg", "house3.jpg"],
    "area": [1200, 1600, 2400],
    "bedrooms": [2, 3, 4],
    "price": [200000, 300000, 450000]
})

# 2. Dataset Class
class HousingDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform
        self.tabular = df.drop(["price", "image_path"], axis=1).values.astype(np.float32)
        self.labels = df["price"].values.astype(np.float32)

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

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        image = Image.open(row["image_path"]).convert("RGB")
        if self.transform:
            image = self.transform(image)
        tabular_data = self.tabular[idx]
        label = self.labels[idx]
        return image, torch.tensor(tabular_data), torch.tensor(label)

# 3. Transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

train_df, test_df = train_test_split(data, test_size=0.2, random_state=42)
train_dataset = HousingDataset(train_df, transform=transform)
test_dataset = HousingDataset(test_df, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=2, shuffle=False)

# 4. Multimodal Model
class MultiModalModel(nn.Module):
    def __init__(self, tabular_input_dim):
        super(MultiModalModel, self).__init__()
        self.cnn = models.resnet50(pretrained=True)
        for param in self.cnn.parameters():
            param.requires_grad = False
        self.cnn.fc = nn.Identity()

        self.tabular_net = nn.Sequential(
            nn.Linear(tabular_input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU()
        )

        self.fc = nn.Sequential(
            nn.Linear(2048 + 32, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, image, tabular):
        img_features = self.cnn(image)
        tab_features = self.tabular_net(tabular)
        combined = torch.cat((img_features, tab_features), dim=1)
        output = self.fc(combined)
        return output.squeeze()

tabular_input_dim = train_df.drop(["price", "image_path"], axis=1).shape[1]
model = MultiModalModel(tabular_input_dim)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# 5. Training
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0
    for images, tabular, labels in train_loader:
        images, tabular, labels = images.to(device), tabular.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images, tabular)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

# 6. Evaluation
model.eval()
preds, trues = [], []
with torch.no_grad():
    for images, tabular, labels in test_loader:
        images, tabular = images.to(device), tabular.to(device)
        outputs = model(images, tabular)
        preds.extend(outputs.cpu().numpy())
        trues.extend(labels.numpy())

mae = mean_absolute_error(trues, preds)
rmse = np.sqrt(mean_squared_error(trues, preds))
print(f"MAE: {mae:.2f}, RMSE: {rmse:.2f}")
```