```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error
import joblib
import tensorflow as tf
from tensorflow.keras import layers, models, Input, Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array



# 1. Mount Google Drive
# --------------------------------
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
csv_path = "/content/drive/MyDrive/socal2_cleaned_mod.csv"




# 2. Load CSV
# --------------------------------
df = pd.read_csv(csv_path)

# Target column
y = df["price"].values

# Drop target + image reference
X_tab = df.drop(columns=["price", "Image_id"], errors="ignore")

# Encode categorical columns
for col in X_tab.select_dtypes(include=['object']).columns:
    X_tab[col] = LabelEncoder().fit_transform(X_tab[col].astype(str))

# Keep only numeric features
X_tab = X_tab.select_dtypes(include=['int64','float64'])

# Scale numeric features
scaler = StandardScaler()
X_tab = scaler.fit_transform(X_tab)
joblib.dump(scaler, "scaler.pkl")
```

```
['scaler.pkl']
```

```
# --------------------------------
# 3. Load Image Data
# --------------------------------
from tensorflow.keras.utils import image_dataset_from_directory

train_ds = image_dataset_from_directory(
    "/content/drive/MyDrive/socal_pics/train",
    labels=None,
    image_size=(128, 128),
    batch_size=32,
    shuffle=False
)

val_ds = image_dataset_from_directory(
    "/content/drive/MyDrive/socal_pics/val",
    labels=None,
    image_size=(128, 128),
    batch_size=32,
    shuffle=False
)

# Normalize
train_ds = train_ds.map(lambda x: x/255.0)
val_ds = val_ds.map(lambda x: x/255.0)
```

```
# Convert to numpy
X_train_img = np.concatenate([x.numpy() for x in train_ds], axis=0)
X_val_img   = np.concatenate([x.numpy() for x in val_ds], axis=0)

print("Image train shape:", X_train_img.shape)
print("Image val shape:", X_val_img.shape)
```

```
Found 2000 files.
Found 1000 files.
Image train shape: (2000, 128, 128, 3)
Image val shape: (1000, 128, 128, 3)
```

```
# 4. Train-Validation Split (tabular & target)
# ------------------------------

# Train split size must match image split size
X_train_tab = X_tab[:len(X_train_img)]
y_train = y[:len(X_train_img)]

X_val_tab = X_tab[len(X_train_img): len(X_train_img) + len(X_val_img)]
y_val = y[len(X_train_img): len(X_train_img) + len(X_val_img)]

print("Tabular train shape:", X_train_tab.shape)
print("Tabular val shape:", X_val_tab.shape)
print("Target train shape:", y_train.shape)
print("Target val shape:", y_val.shape)
```

```
Tabular train shape: (2000, 5)
Tabular val shape: (1000, 5)
Target train shape: (2000,)
Target val shape: (1000,)
```

```
# 5. Build Model (CNN + MLP Fusion)
# ------------------------------
# CNN branch for images
img_input = Input(shape=(128,128,3))
x = layers.Conv2D(32, (3,3), activation="relu")(img_input)
x = layers.MaxPooling2D((2,2))(x)
x = layers.Conv2D(64, (3,3), activation="relu")(x)
x = layers.MaxPooling2D((2,2))(x)
x = layers.Flatten()(x)
x = layers.Dense(64, activation="relu")(x)
img_branch = Model(inputs=img_input, outputs=x)

# MLP branch for tabular data
tab_input = Input(shape=(X_train_tab.shape[1],))
y_tab = layers.Dense(64, activation="relu")(tab_input)
y_tab = layers.Dense(32, activation="relu")(y_tab)
tab_branch = Model(inputs=tab_input, outputs=y_tab)

# Fusion
combined = layers.concatenate([img_branch.output, tab_branch.output])
z = layers.Dense(64, activation="relu")(combined)
z = layers.Dense(1)(z)  # Regression output

# Final multimodal model
model = Model(inputs=[img_branch.input, tab_branch.input], outputs=z)

model.compile(optimizer="adam", loss="mse", metrics=["mae"])
model.summary()
```

**Model: "functional_7"**

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer_4 (InputLayer) | (None, 128, 128, 3) | 0 | - |
| conv2d_6 (Conv2D) | (None, 126, 126, 32) | 896 | input_layer_4[0]… |
| max_pooling2d_6 (MaxPooling2D) | (None, 63, 63, 32) | 0 | conv2d_6[0][0] |
| conv2d_7 (Conv2D) | (None, 61, 61, 64) | 18,496 | max_pooling2d_6[… |
| max_pooling2d_7 (MaxPooling2D) | (None, 30, 30, 64) | 0 | conv2d_7[0][0] |
| input_layer_5 (InputLayer) | (None, 5) | 0 | - |
| flatten_3 (Flatten) | (None, 57600) | 0 | max_pooling2d_7[… |
| dense_8 (Dense) | (None, 64) | 384 | input_layer_5[0]… |
| dense_7 (Dense) | (None, 64) | 3,686,464 | flatten_3[0][0] |
| dense_9 (Dense) | (None, 32) | 2,080 | dense_8[0][0] |
| concatenate_1 (Concatenate) | (None, 96) | 0 | dense_7[0][0], dense_9[0][0] |
| dense_10 (Dense) | (None, 64) | 6,208 | concatenate_1[0]… |
| dense_11 (Dense) | (None, 1) | 65 | dense_10[0][0] |

**Total params:** 3,714,593 (14.17 MB)

```
# 6. Train
# ------------------------------
history = model.fit(
    [X_train_img, X_train_tab], y_train,
    validation_data=([X_val_img, X_val_tab], y_val),
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 112ms/step - loss: 425288269824.0000 - mae: 529298.8750 - val_loss: 241055350784.0000 - val_mae: 353962.3
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - loss: 158053138432.0000 - mae: 268269.2188 - val_loss: 286324457472.0000 - val_mae: 407476.96
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 162452979712.0000 - mae: 270113.8438 - val_loss: 307811647488.0000 - val_mae: 432207.281
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 168504934400.0000 - mae: 276483.4062 - val_loss: 323742957568.0000 - val_mae: 449608.125
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 145546952704.0000 - mae: 250540.5312 - val_loss: 278961389568.0000 - val_mae: 399525.781
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 134707011584.0000 - mae: 246883.2344 - val_loss: 346880802816.0000 - val_mae: 473776.968
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 167061618688.0000 - mae: 269569.3438 - val_loss: 355822993408.0000 - val_mae: 482891.687
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 149905932288.0000 - mae: 249402.2500 - val_loss: 293457723392.0000 - val_mae: 418274.781
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 152758943744.0000 - mae: 258992.3594 - val_loss: 272324788224.0000 - val_mae: 394070.031
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 151139958784.0000 - mae: 259256.5156 - val_loss: 297518596096.0000 - val_mae: 424711.687
```

```
# 7. Evaluate
# ------------------------------
preds = model.predict([X_val_img, X_val_tab])
mae = mean_absolute_error(y_val, preds)
rmse = np.sqrt(mean_squared_error(y_val, preds))

print("MAE:", mae)
print("RMSE:", rmse)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step
MAE: 424711.625
RMSE: 545452.6825160914
```

```
# 8. Save Model
# ------------------------------
model.save("multimodal_house_price.h5")
print("✅ Model saved as multimodal_house_price.h5")
```

```
mmend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```