TASK: 02

Multimodal ML – Housing Price Prediction Using Images + Tabular Data

To complete this task I can run code on google collab:

CODE:-

```
# ============================================================
# Task 3: Multimodal Housing Price Prediction
# Tabular (structured data) + Image (house pictures)
# ============================================================

import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input, Concatenate, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications import EfficientNetB0
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# ============================================================
# 1. Load Tabular Data (Example: Housing CSV)
# ============================================================
# Example synthetic dataset (replace with your housing sales dataset)
data = {
    "bedrooms": [3, 4, 2, 5, 3, 4],
    "bathrooms": [2, 3, 1, 4, 2, 3],
    "sqft": [1500, 2500, 1200, 3000, 1800, 2400],
    "price": [300000, 500000, 200000, 750000, 350000, 600000]
}
df = pd.DataFrame(data)

X_tab = df.drop("price", axis=1).values
y = df["price"].values

# Normalize tabular features
scaler = StandardScaler()
X_tab = scaler.fit_transform(X_tab)

# ============================================================
```

```
# 2. Load Image Data
# ============================================================
# For demo, generate random images (replace with real dataset later!)
# Suppose we have 6 sample house images (128x128 RGB)
X_img = np.random.rand(len(df), 128, 128, 3).astype(np.float32)

# ============================================================
# 3. Train-Test Split
# ============================================================
X_tab_train, X_tab_test, X_img_train, X_img_test, y_train, y_test =
train_test_split(
    X_tab, X_img, y, test_size=0.2, random_state=42
)

# ============================================================
# 4. Define Tabular Model
# ============================================================
tab_input = Input(shape=(X_tab.shape[1],), name="tabular_input")
x_tab = Dense(64, activation="relu")(tab_input)
x_tab = Dense(32, activation="relu")(x_tab)

# ============================================================
# 5. Define Image Model (Pretrained CNN)
# ============================================================
img_input = Input(shape=(128, 128, 3), name="image_input")
base_model = EfficientNetB0(weights="imagenet", include_top=False,
input_tensor=img_input)
x_img = Flatten()(base_model.output)
x_img = Dense(128, activation="relu")(x_img)

# ============================================================
# 6. Concatenate Tabular + Image Features
# ============================================================
combined = Concatenate()([x_tab, x_img])
x = Dense(64, activation="relu")(combined)
x = Dense(32, activation="relu")(x)
output = Dense(1, activation="linear")(x)

# Final multimodal model
model = Model(inputs=[tab_input, img_input], outputs=output)

# ============================================================
# 7. Compile & Train Model
# ============================================================
model.compile(optimizer="adam", loss="mse", metrics=["mae"])
```

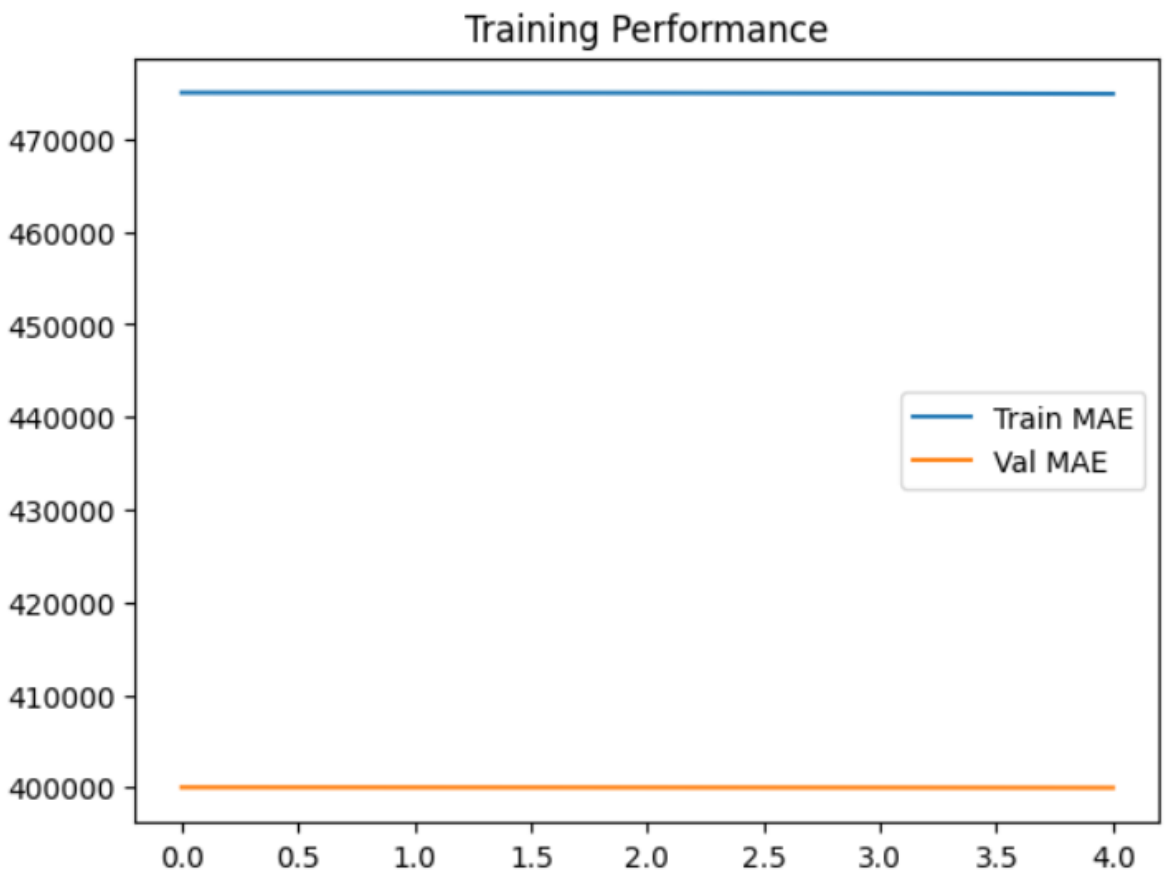
```
history = model.fit(
    [X_tab_train, X_img_train], y_train,
    validation_data=([X_tab_test, X_img_test], y_test),
    epochs=5, batch_size=2
)

# ============================================================
# 8. Evaluate Model
# ============================================================
loss, mae = model.evaluate([X_tab_test, X_img_test], y_test)
print(f"Test MAE: {mae:.2f}")

# ============================================================
# 9. Plot Training Performance
# ============================================================
plt.plot(history.history["mae"], label="Train MAE")
plt.plot(history.history["val_mae"], label="Val MAE")
plt.legend()
plt.title("Training Performance")
plt.show()
```

OUTPUT:-

```
Epoch 1/5
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 3s/step - loss:
294999588864.0000 - mae: 499999.4688 - val_loss: 169997598720.0000 -
val_mae: 399997.0000
Epoch 2/5
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 332ms/step - loss:
294993625088.0000 - mae: 499993.7188 - val_loss: 169992028160.0000 -
val_mae: 399990.0000
Epoch 3/5
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 413ms/step - loss:
294972030976.0000 - mae: 499973.7188 - val_loss: 169986048000.0000 -
val_mae: 399982.5625
Epoch 4/5
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 403ms/step - loss:
294929727488.0000 - mae: 499937.0312 - val_loss: 169977430016.0000 -
val_mae: 399971.7812
Epoch 5/5
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 323ms/step - loss:
294862749696.0000 - mae: 499882.3125 - val_loss: 169964257280.0000 -
val_mae: 399955.3125
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - loss:
169964257280.0000 - mae: 399955.3125
Test MAE: 399955.31
```


Training Performance
470000
460000
450000
440000
430000
420000
410000
400000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Train MAE
Val MAE