In [ ]:
```
# ==========================================================
# Image Captioning AI (ResNet50 + LSTM) - Jupyter Version
# ==========================================================

import numpy as np
import string
import os
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Embedding, Dense, Dropout, Add, Input

# ----------------------------------------------------------
# 0. SET PATHS (LOCAL WINDOWS PC)
# ----------------------------------------------------------
captions_file = r"C:\Users\LENOVO\archive\captions.txt"
images_path   = r"C:\Users\LENOVO\archive\Images"
features_file = r"C:\Users\LENOVO\archive\features.npy"

# ----------------------------------------------------------
# 1. LOAD CAPTIONS
# ----------------------------------------------------------
def load_captions(filename):
    captions = {}
    if not os.path.exists(filename):
        raise FileNotFoundError(f"❌ Captions file not found: {filename}")
    with open(filename, 'r', encoding='utf-8') as file:
        text = file.read()
    for line in text.split("\n"):
        if len(line.strip()) < 2:
            continue
        tokens = line.split(' ')
        img = tokens[0].split('#')[0]
        caption = ' '.join(tokens[1:])
        caption = caption.lower().translate(str.maketrans('', '', string.punctuation))
        captions.setdefault(img, []).append("startseq " + caption + " endseq")
    return captions

captions = load_captions(captions_file)
print("✅ Loaded Captions:", len(captions))

# ----------------------------------------------------------
# 2. EXTRACT IMAGE FEATURES (ResNet50) WITH CACHING
# ----------------------------------------------------------
resnet = ResNet50(weights="imagenet", include_top=False, pooling="avg")

def extract_features(directory):
    features = {}
    img_files = [f for f in os.listdir(directory) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
    for idx, img_name in enumerate(img_files, 1):
        filename = os.path.join(directory, img_name)
        try:
            img = image.load_img(filename, target_size=(224, 224))
            x = image.img_to_array(img)
            x = np.expand_dims(x, axis=0)
            x = preprocess_input(x)
            feature = resnet.predict(x, verbose=0)
            features[img_name] = feature
            if idx % 100 == 0:
                print(f"📸 Processed {idx}/{len(img_files)} images")
        except Exception as e:
            print(f"❌ Skipped {img_name} (Error: {e})")
    return features

# Load cached features if available
if os.path.exists(features_file):
    print("✅ Loading cached features...")
    features = np.load(features_file, allow_pickle=True).item()
else:
    print("⚠ No cached features found, extracting now...")
    features = extract_features(images_path)
    np.save(features_file, features)
    print("✅ Features cached for future runs.")

print("✅ Extracted Features for Images:", len(features))

# ----------------------------------------------------------
# 3. BUILD VOCABULARY
# ----------------------------------------------------------
all_captions = [cap for caps in captions.values() for cap in caps]
words = [w for cap in all_captions for w in cap.split()]
unique = sorted(list(set(words)))
vocab_size = len(unique)
print("✅ Vocabulary Size:", vocab_size)

word_to_index = {w: i+1 for i, w in enumerate(unique)}
index_to_word = {i+1: w for i, w in enumerate(unique)}
max_length = max(len(c.split()) for c in all_captions)
print("✅ Max Caption Length:", max_length)

# ----------------------------------------------------------
# 4. DATA GENERATOR
# ----------------------------------------------------------
def data_generator(captions, features, word_to_index, max_length, vocab_size, batch_size=32):
    X1, X2, y = [], [], []
    n = 0
    while True:
        for key, caps in captions.items():
            if key not in features:
                continue
            pic = features[key][0]
            for cap in caps:
                seq = [word_to_index[w] for w in cap.split() if w in word_to_index]
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = tf.keras.utils.to_categorical([out_seq], num_classes=vocab_size+1)[0]
                    X1.append(pic)
                    X2.append(in_seq)
                    y.append(out_seq)
            n += 1
            if n == batch_size:
                yield ([np.array(X1), np.array(X2)], np.array(y))
                X1, X2, y = [], [], []
                n = 0

# ----------------------------------------------------------
# 5. DEFINE MODEL
# ----------------------------------------------------------
embedding_dim = 256

# Image branch
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

# Text branch
inputs2 = Input(shape=(max_length,))
se1 = Embedding(input_dim=vocab_size+1, output_dim=embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# Decoder
decoder1 = Add()([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size+1, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()

# ----------------------------------------------------------
# 6. TRAIN MODEL
# ----------------------------------------------------------
batch_size = 32
steps_per_epoch = max(1, len(features) // batch_size)
epochs = 20  # adjust as needed

generator = data_generator(captions, features, word_to_index, max_length, vocab_size, batch_size=batch_size)
model.fit(generator, epochs=epochs, steps_per_epoch=steps_per_epoch, verbose=1)

# ----------------------------------------------------------
# 7. GENERATE CAPTION
# ----------------------------------------------------------
def generate_caption(model, photo, word_to_index, index_to_word, max_length):
    in_text = "startseq"
    for i in range(max_length):
        seq = [word_to_index[w] for w in in_text.split() if w in word_to_index]
        seq = pad_sequences([seq], maxlen=max_length)
        yhat = model.predict([photo, seq], verbose=0)
        yhat = np.argmax(yhat)
        word = index_to_word.get(yhat, None)
        if word is None:
            break
        in_text += " " + word
        if word == "endseq":
            break
    return in_text

# Test one image
if len(features) > 0:
    test_img = list(features.keys())[0]
    caption = generate_caption(model, features[test_img], word_to_index, index_to_word, max_length)
    print("🖼 Image:", test_img)
    print("📝 Generated Caption:", caption)
else:
    print("⚠ No features extracted. Check image folder path.")
```

```
✅ Loaded Captions: 21672
✅ Loading cached features...
✅ Extracted Features for Images: 8091
✅ Vocabulary Size: 8747
✅ Max Caption Length: 37
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 37)	0	-
input_layer_1 (InputLayer)	(None, 2048)	0	-
embedding (Embedding)	(None, 37, 256)	2,239,488	input_layer_2[0][0]
dropout (Dropout)	(None, 2048)	0	input_layer_1[0][0]
dropout_1 (Dropout)	(None, 37, 256)	0	embedding[0][0]
not_equal (NotEqual)	(None, 37)	0	input_layer_2[0][0]
dense (Dense)	(None, 256)	524,544	dropout[0][0]
lstm (LSTM)	(None, 256)	525,312	dropout_1[0][0], not_equal[0][0]
add (Add)	(None, 256)	0	dense[0][0], lstm[0][0]
dense_1 (Dense)	(None, 256)	65,792	add[0][0]
dense_2 (Dense)	(None, 8748)	2,248,236	dense_1[0][0]

Total params: 5,603,372 (21.38 MB)
Trainable params: 5,603,372 (21.38 MB)
Non-trainable params: 0 (0.00 B)

In [ ]:
```
from itertools import islice

print("Total features:", len(features))
print("Total vocab size:", len(index_to_word))

# Pick 5 keys super fast
test_images = list(islice(features.keys(), 5))
print("✅ Picked test images:", test_images)

# Precompute vocab
vocab_list = list(index_to_word.values())

def mock_caption(vocab_list, max_length=5):
    return "startseq " + " ".join(vocab_list[:max_length]) + " endseq"

for img in test_images:
    print("🖼", img)
    print("📝", mock_caption(vocab_list))
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# ----------------------------------------------------------
# 8. GENERATE AND PRINT 5 REAL CAPTIONS
# ----------------------------------------------------------
num_test_images = 5
test_images = list(features.keys())[:num_test_images]

print("\n🖼 Generated Captions for 5 Images:")
for img_name in test_images:
    photo = features[img_name]
    caption = generate_caption(model, photo, word_to_index, index_to_word, max_length)
```

```
    print(f"Image: {img_name}")
    print(f"Caption: {caption}\n")
```

In [ ]: