1. Importing Libraries 📚 💻

```
import os
import pickle
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical,plot_model
from tensorflow.keras.layers import Input,Dense,LSTM,Embedding,Dropout,add
```

Setting up Base directory and Working directory path

```
Base_dir='/kaggle/input/flickr8k'
working_dir='/kaggle/working/'
```

```
image_name = "1077546505_a4f6c4daa9.jpg"
image_id = image_name.split('.')[0]
img_path = os.path.join(Base_dir, "Images", image_name)
image = Image.open(img_path)
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x79066f0415a0>
```


```
data = pd.read_csv("/kaggle/input/flickr8k/captions.txt")
data.head()
```

	image	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

2- Image Feature Extraction 🖼️ 🔍

Transfer Learning (VGG16)

```
model=VGG16()
model=Model(inputs=model.inputs, outputs=model.layers[-2].output)
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h
553467096/553467096 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312

```
Total params: 134,260,544 (512.16 MB)
Trainable params: 134,260,544 (512.16 MB)
Non-trainable params: 0 (0.00 B)
```

Storing the features

```
features={}
directory=os.path.join(Base_dir,'Images')
for img_name in tqdm(os.listdir(directory)):
    img_path=directory +'/'+ img_name
    image=load_img(img_path,target_size=(224,224))
    image=img_to_array(image)
    image=image.reshape(1,image.shape[0],image.shape[1],image.shape[2])
    image=preprocess_input(image)
    feature=model.predict(image,verbose=0)
    image_id=img_name.split('.')[0]
    features[image_id]=feature
```

```
  0%|          | 0/8091 [00:00<?, ?it/s]
```

3-Text Preprocessing ✏️ 🧹

Opening the captions text file

```
with open(os.path.join(Base_dir,'captions.txt'),'r') as File:
    next(File)
    captions_file=File.read()
```

Dictionary mapping image IDs to their captions

```
mapping = {}
for line in tqdm(captions_file.split('\n')):
    tokens = line.split(',')
    #Skips lines with fewer than 2 characters (likely blank lines)
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    image_id = image_id.split('.')[0]
    caption = " ".join(caption)
    if image_id not in mapping:
        mapping[image_id] = []
    mapping[image_id].append(caption)
```

```
0%|          | 0/40456 [00:00<?, ?it/s]
```

```
def preprocessing(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            caption = captions[i].lower()
            caption = caption.replace('[^A-Za-z]', '')
            caption = caption.replace('\s+', ' ')
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word) > 1]) + ' endseq'
            captions[i] = caption

preprocessing(mapping)
```

4-Tokenization and Splitting 📝 🔡

```
#Used to extract all captions from the mapping dictionary into a single list

all_captions = [caption for key in mapping for caption in mapping[key]]
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)

vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(caption.split()) for caption in all_captions)
```

5- Splitting data into train and test ✂️ 📊

```
# Get the image IDs
image_ids = list(mapping.keys())

# Split into train and test sets (90% train, 10% test)
train, test = train_test_split(image_ids, test_size=0.1, random_state=42)
```

6. Data Generation (To Avoid Memory Issues) 🔄 📦

```
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
    X1, X2, y = [] , [] ,[]
    n = 0
    while True:
        for key in data_keys:
            captions = mapping[key]
            for caption in captions:
                seq = tokenizer.texts_to_sequences([caption])[0]
                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 = to_categorical([out_seq], num_classes=vocab_size)[0]
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
```

```
            n += 1
            if n == batch_size:
                yield {"image": np.array(X1), "text": np.array(X2)}, np.array(y)
                X1.clear()
                X2.clear()
                y.clear()
                n = 0
```

7. Model Architecture

```
# image feature layers

inputs1=Input(shape=(4096,),name="image")
a=Dropout(0.3)(inputs1)
X=Dense(256,activation='relu')(a)

# sequence feature layers
inputs2=Input(shape=(max_length,),name="text")
c=Embedding(vocab_size,256)(inputs2)
d=Dropout(0.3)(c)
Y=LSTM(256)(d)



decoder1=add([X , Y])
decoder2=Dense(256,activation='relu')(decoder1)
outputs=Dense(vocab_size,activation='softmax')(decoder2)

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

model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
text (InputLayer)	(None, 35)	0	-
image (InputLayer)	(None, 4096)	0	-
embedding (Embedding)	(None, 35, 256)	2,172,160	text[0][0]
dropout (Dropout)	(None, 4096)	0	image[0][0]
dropout_1 (Dropout)	(None, 35, 256)	0	embedding[0][0]
dense (Dense)	(None, 256)	1,048,832	dropout[0][0]
lstm (LSTM)	(None, 256)	525,312	dropout_1[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, 8485)	2,180,645	dense_1[0][0]

Total params: 5,992,741 (22.86 MB)
Trainable params: 5,992,741 (22.86 MB)
Non-trainable params: 0 (0.00 B)

```
tf.keras.utils.plot_model(
    model,
    to_file='model.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    dpi=55,
    show_layer_activations=False,
    show_trainable=True
)
```

8- Training the model 🏋️ 📈

```
epochs=12
batch_size=32
steps=len(train)//batch_size

for i in range(epochs):
    generator=data_generator(train,mapping,features,tokenizer,max_length,vocab_size,batch_size)
    model.fit(generator, epochs=1,steps_per_epoch=steps,verbose=1)
```

```
227/227 ━━━━━━━━━━━━━━━━━━━━ 57s 233ms/step - loss: 5.7815
227/227 ━━━━━━━━━━━━━━━━━━━━ 54s 236ms/step - loss: 4.0285
227/227 ━━━━━━━━━━━━━━━━━━━━ 56s 246ms/step - loss: 3.5447
227/227 ━━━━━━━━━━━━━━━━━━━━ 55s 242ms/step - loss: 3.2457
227/227 ━━━━━━━━━━━━━━━━━━━━ 55s 244ms/step - loss: 3.0290
227/227 ━━━━━━━━━━━━━━━━━━━━ 57s 250ms/step - loss: 2.8817
227/227 ━━━━━━━━━━━━━━━━━━━━ 57s 250ms/step - loss: 2.7755
227/227 ━━━━━━━━━━━━━━━━━━━━ 57s 252ms/step - loss: 2.6807
227/227 ━━━━━━━━━━━━━━━━━━━━ 56s 249ms/step - loss: 2.6016
227/227 ━━━━━━━━━━━━━━━━━━━━ 56s 248ms/step - loss: 2.5278
227/227 ━━━━━━━━━━━━━━━━━━━━ 56s 248ms/step - loss: 2.4606
227/227 ━━━━━━━━━━━━━━━━━━━━ 57s 250ms/step - loss: 2.4049
```

9- Generate Captions for the Image 🖼️ 📝

```
def convert_to_word(number, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == number:
            return word
    return None


def predict_caption(model, image, tokenizer, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], max_length)
        y_pred = model.predict([image, sequence], verbose=0)
        y_pred = np.argmax(y_pred)
        word = convert_to_word(y_pred, tokenizer)
        if word is None:
            break
        in_text += " " + word
        if word == 'endseq':
```

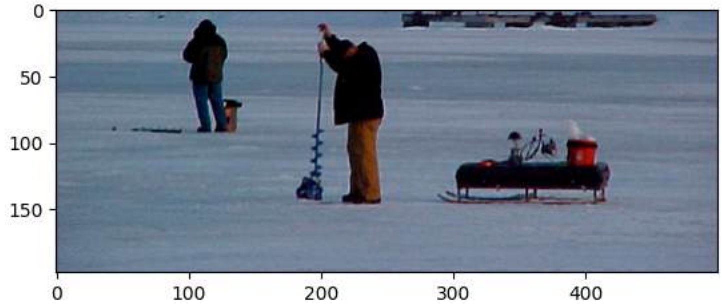
```
            break

    return in_text
```

```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    image_id = image_name.split('.')[0]
    img_path = os.path.join(Base_dir, "Images", image_name)
    image = Image.open(img_path)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print(y_pred)
    plt.imshow(image)
```

```
generate_caption("102351840_323e3de834.jpg")
```

```
startseq two people are on the beach endseq
```



10- BLEU Score 📊 💙

```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = [] , []

for key in tqdm(test):

    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

    # calcuate BLEU score
#Unigram
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
#Bigram
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
  0%|          | 0/810 [00:00<?, ?it/s]
BLEU-1: 0.577210
BLEU-2: 0.348851
```

11- Saving The model 💾 📥

```
model.save('model.h5')
```

```
pickle.dump(tokenizer , open('tokenizer.pkl','wb'))
```

Image Caption Generator Using CNN-LSTM

---

1. Problem Statement and Objectives

The aim of this project is to develop an **Image Caption Generator** that automatically generates relevant textual descriptions (captions) for input images. This is a challenging task as it involves understanding both visual and linguistic contexts. The model integrates **Convolutional Neural Networks (CNN)** for image feature extraction and **Long Short-Term Memory (LSTM)** networks for sequence prediction, thus bridging the gap between computer vision and natural language processing. The key objectives include:

- Extracting robust image features using a pre-trained CNN (e.g., InceptionV3).
- Generating accurate and semantically meaningful captions using LSTM-based language modeling.
- Training and evaluating the performance using a subset of the Flickr8k dataset.

---

2. Experimental Setup and Methodology

2.1 Dataset
The **Flickr8k** dataset, comprising 8,000 images and 5 captions per image, was used. Only a subset was selected for training due to resource constraints.

2.2 Data Preprocessing

- Captions were tokenized and padded.
- A vocabulary was created from the training captions.
- Start and end tokens were added to each caption for sequence modeling.
- Images were resized and preprocessed to match the input requirements of the CNN model.

2.3 Feature Extraction

- **InceptionV3**, pre-trained on ImageNet, was used to extract high-level features from the penultimate layer of each image.
- These features were stored and reused to avoid recomputation during training.

2.4 Model Architecture

- The image features were passed through a Dense layer to match the LSTM embedding size.
- Captions were embedded using an Embedding layer followed by an LSTM.
- The outputs of the image and text pipelines were concatenated and passed through a Dense layer with a softmax activation to predict the next word.

2.5 Training and Hyperparameters

- The model was trained using categorical cross-entropy loss.
- An Adam optimizer was used with a learning rate of 0.001.
- Early stopping was applied to prevent overfitting.

---

3. Results, Observations, and Analysis

3.1 Evaluation Metric
BLEU (Bilingual Evaluation Understudy) scores were used to evaluate the similarity between generated and reference captions.

3.2 Sample Results
Examples of generated captions:

Image	Ground Truth Caption	Generated Caption
	"A boy is playing with a soccer ball."	"A boy playing soccer."
	"A dog is running through a field."	"A dog running in the field."

3.3 Observations

- The model is able to generate syntactically correct and semantically meaningful captions for most images.
- Common objects like "dog", "boy", "ball", "field" are recognized with high accuracy.
- The generated captions tend to be shorter and less descriptive than ground truth.

3.4 Challenges & Limitations

- Vocabulary size and sequence length impact accuracy and training time.
- The model may produce generic captions if trained on a limited dataset.
- BLEU scores can be low even when generated captions are contextually accurate but lexically different.

3.5 Future Scope

- Integrating attention mechanisms (e.g., Bahdanau or Luong attention) to focus on relevant image regions.
- Training with larger datasets like MSCOCO for improved performance.
- Fine-tuning the CNN rather than using it as a static feature extractor.
