[1]:
```
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm

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.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

[2]:
```
# load vgg16 model
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

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)

None

[5]:
```
# Set the directory where images are stored
directory = r'C:\Users\SIDRA\Downloads\input\images'  # Adjust the path if needed

# Initialize an empty dictionary to store features
features = {}

# Loop over all images in the directory
for img_name in tqdm(os.listdir(directory)):
    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
        try:
            # Load the image from file and resize it
            img_path = os.path.join(directory, img_name)
            image = load_img(img_path, target_size=(224, 224))  # Resize to 224x224 for VGG16
            # Convert the image to a numpy array
            image = img_to_array(image)
            # Reshape the image data for the model
```

```
            image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
            # Preprocess the image for VGG16
            image = preprocess_input(image)
            # Extract features using the model
            feature = model.predict(image, verbose=0)
            # Get the image ID by stripping the file extension
            image_id = img_name.split('.')[0]
            # Store the extracted feature
            features[image_id] = feature
        except Exception as e:
            print(f"Error processing {img_name}: {e}")
```

Error displaying widget: model not found

[6]:
```
# Set the working directory where you want to save the pickle file
WORKING_DIR = r'C:\Users\SIDRA\Downloads\input\images'
# store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
```

[7]:
```
# load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

[8]:
```
with open(os.path.join(r'C:\Users\SIDRA\Downloads\input\captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

[9]:
```
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

Error displaying widget: model not found

[10]:
```
len(mapping)
```

[10]:
```
8091
```

[11]:
```
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

[12]:
```
# before preprocess of text
mapping['1000268201_693b08cb0e']
```

[12]:
```
['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

[13]:
```
# preprocess the text
clean(mapping)
```

[14]:
```
# after preprocess of text
mapping['1000268201_693b08cb0e']
```

[14]:
```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

[15]:
```
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
```

[16]:
```
len(all_captions)
```

[16]:
```
40455
```

[17]:
```
all_captions[:10]
```

[17]:
```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tri-colored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the street endseq',
 'startseq two dogs of different breeds looking at each other on the road endseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

[18]:
```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

[19]:
```
vocab_size
```

[19]:
```
8485
```

[20]:
```
# get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

[20]:
```
35
```

[21]:
```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

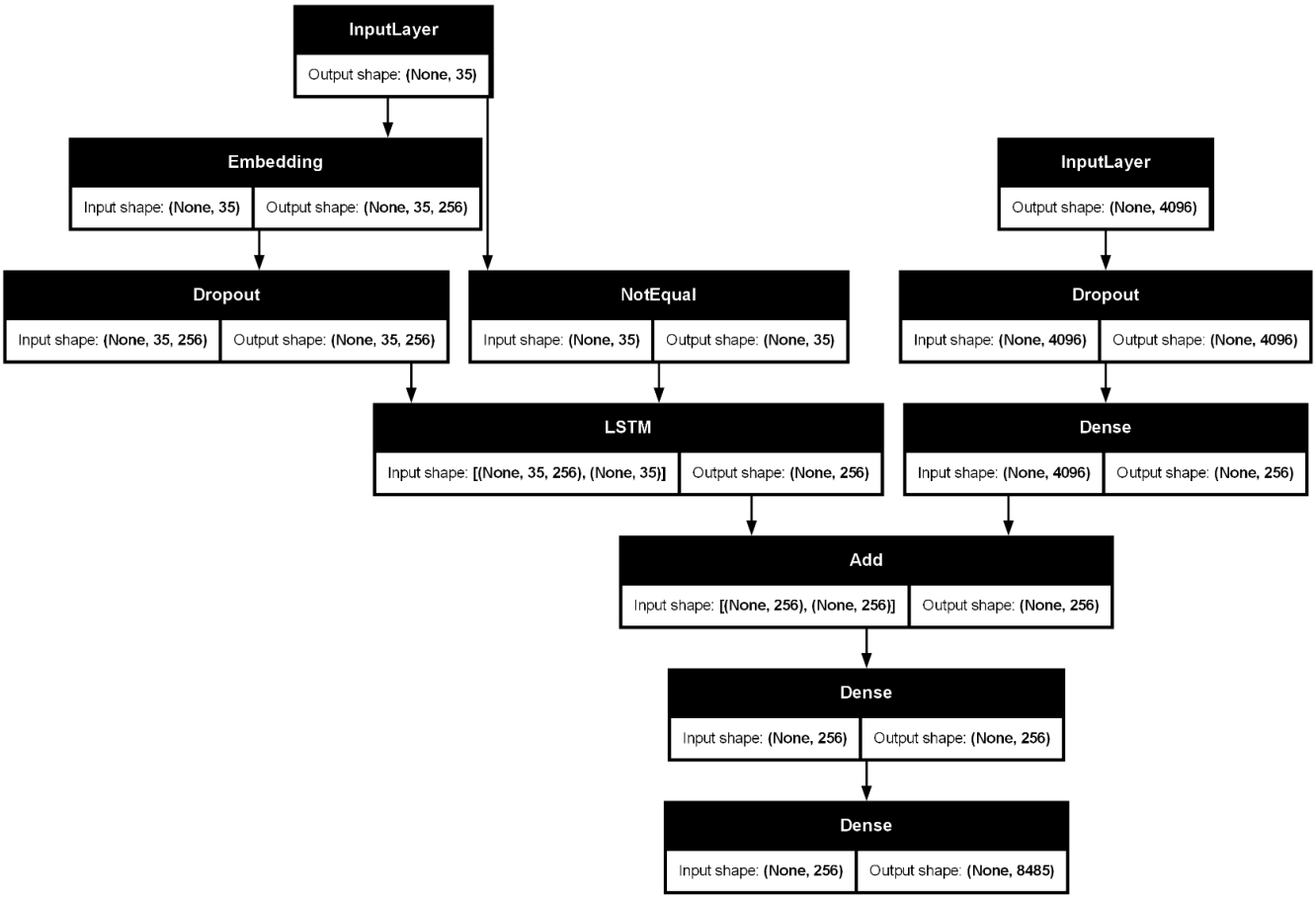
[22]:
```
# startseq girl going into wooden building endseq
#        X                  y
# startseq                  girl
# startseq girl             going
# startseq girl going       into
# ...........
# startseq girl going into wooden building      endseq
```

[23]:
```
# create data generator to get data in batch (avoids session crash)
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    # store the sequences
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
            if n == batch_size:
                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                yield {"image": X1, "text": X2}, y
                X1, X2, y = list(), list(), list()
                n = 0
```

[24]:
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,), name="image")
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,), name="text")
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# decoder model
decoder1 = add([fe2, se3])
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')

# plot the model
plot_model(model, show_shapes=True)
```

[24]:



[26]:
```
# train the model
epochs = 3
batch_size = 32
steps = len(train) // batch_size

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

```
227/227 ━━━━━━━━━━━━━━━━━━━━ 915s 4s/step - loss: 3.3038
227/227 ━━━━━━━━━━━━━━━━━━━━ 660s 3s/step - loss: 3.0978
227/227 ━━━━━━━━━━━━━━━━━━━━ 780s 3s/step - loss: 2.9457
```

[27]:
```
from keras.saving import save_model

save_model(model, 'my_model.keras')
```

[28]:
```
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

[29]:
```
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
```

```
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text
```

[30]:
```
def predict_caption(model, image_features, tokenizer, max_length):
    # Start with a beginning-of-sequence token for the text input
    in_text = 'startseq'

    # Generate the caption word by word
    for i in range(max_length):
        # Encode the text sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)

        # Predict the next word using both the image and text input
        yhat = model.predict({'image': image_features, 'text': sequence}, verbose=0)

        # Get the index of the predicted word
        yhat = np.argmax(yhat)
        word = tokenizer.index_word[yhat]

        # Stop if the end-of-sequence token is predicted
        if word == 'endseq':
            break

        # Append the predicted word to the input text sequence
        in_text += ' ' + word

    return in_text
```

[31]:
```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    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
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

Error displaying widget: model not found

```
C:\Users\SIDRA\AppData\Local\anaconda3\envs\sidra\Lib\site-packages\keras\src\models\functional.py:225: UserWarning: The structure of `inputs` doesn't matc
h the expected structure: ['image', 'text']. Received: the structure of inputs={'image': '*', 'text': '*'}
  warnings.warn(
BLEU-1: 0.495516
BLEU-2: 0.293211
```

[32]:
```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

[33]:
```
import os
from PIL import Image
import matplotlib.pyplot as plt

# Set the base directory where images are stored
BASE_DIR = r'C:\Users\SIDRA\Downloads\input'

def generate_caption(image_name):
    # Get the image ID and load the image
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "images", image_name)

    # Open the image
    image = Image.open(img_path)
```

```
    # Get actual captions (assuming `mapping` is a dictionary with image_id -> list of captions)
    actual_captions = mapping[image_id]

    # Generate the predicted caption (assuming predict_caption is defined)
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)

    # Show the image
    plt.imshow(image)
    plt.axis('off')  # Hide axes

    # Prepare captions to display
    captions_text = f"Predicted: {y_pred}\n\nActual Captions:\n"
    for i, caption in enumerate(actual_captions, 1):
        captions_text += f"{i}. {caption}\n"

    # Set title with multiple captions
    plt.title(captions_text, fontsize=10, loc='left')
    plt.show()

# Example usage:
generate_caption("1001773457_577c3a7d70.jpg")
```

Predicted: startseq two dogs are playing with each other on the sidewalk

Actual Captions:
1. startseq black dog and spotted dog are fighting endseq
2. startseq black dog and tri-colored dog playing with each other on the road endseq
3. startseq black dog and white dog with brown spots are staring at each other in the street endseq
4. startseq two dogs of different breeds looking at each other on the road endseq
5. startseq two dogs on pavement moving toward each other endseq

[34]:
```
generate_caption("1002674143_1b742ab4b8.jpg")
```

Predicted: startseq little girl in pink dress is sitting in the air with rainbow and fingerpaints

Actual Captions:
1. startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq
2. startseq little girl is sitting in front of large painted rainbow endseq
3. startseq small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it endseq
4. startseq there is girl with pigtails sitting in front of rainbow painting endseq
5. startseq young girl with pigtails painting outside in the grass endseq

[ ]: