Architecture

SUBMITTED BY

RASHID ALI

(IMAGE CAPTIONING PROJECT)

INEURON INTERNSHIP

Domain: Security and Safety, Surveillance

IMAGE - CAPTIONING

Document Version Control:

Date	Version	Author	Comments
20/06/2022	V1.0	Prateek Magdum	First Draft

Approval Status:

Version	Review Data	Reviewed By	Approved By	Comments
V1.0				

Contents

1 Introduction

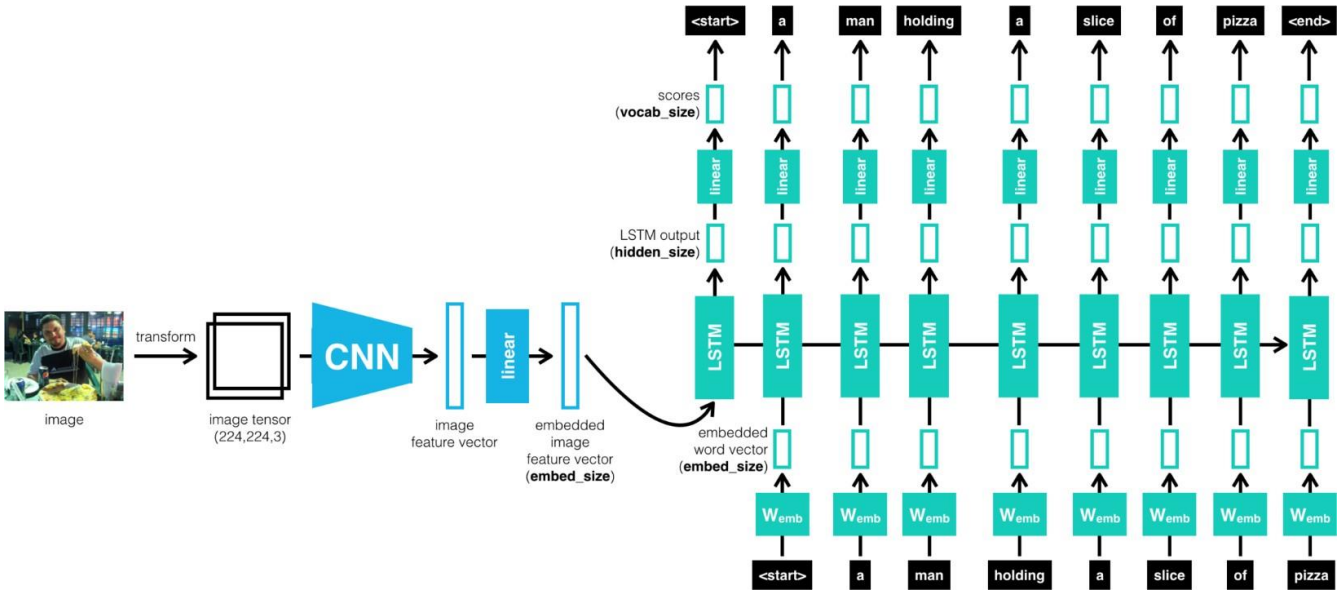
1.1 Why this Architecture design document?

The purpose of this document is to provide a detailed architecture design of the ,image captioning could revolutionize the way we look at images. Project by focusing on each of the attributes of our architecture.
This document will address the background of this project, and the architecturally significant function requirements. The intention of this document is to help the development team to determine how the system will be structured at the highest level.

1.2 Scope

Architecture Design Document (ADD) is an architectural design process that follows a step-by-step refinement process. The process can be used for designing data structures, required software architecture, source code, and ultimately, performance algorithms. Overall, the design principles maybe defined during requirement analysis and then refined during architectural design work.

2 Architecture



2.1 Architecture Description –

2.1.1 Data Description –

We are planning on using the Flickr 8K dataset for our project. Considering that we have two diverse sets of features, images and captions (text), we would be using a combination ofarchitectures that can encompass the features present in the data

2.1.2 Define the Use Cases –

Image Captioning has been widely used in different forms ranging from image-indexing for visually-impaired individuals to group images on platforms like Google and Facebook.The idea can be extended to video captioning, audio-video annotation, and the popular Visual- Question Answering (VQA) challenge which becomes more user-interactive as we ask questions regarding the input image and receives answers in various formats

2.1.3 Load the CNN Model –

In [3]:
```
# load vgg16 model
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
```

2.1.4 Clean the Text –

By removing the extra space and special character

In [10]:
```
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
```

In [11]:
```
# preprocess the text
clean(map)
```

In [12]:
```
#taking all caption in on list
allcaptions = []
for i in map:
    for j in map[i]:
        allcaptions.append(j)
```

In [13]:
```
len(allcaptions)
```

Out[13]:
```
40455
```

In [14]:
```
allcaptions[:10]
```

Out[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',
 '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']
```

In [15]:
```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(allcaptions)
vocab_size = len(tokenizer.word_index) + 1
```

2.1.5 Language Model –

For image captioning, we are creating an LSTM-based model that is used to predict the sequence of words, called the caption, from the feature vectors obtained from the VGG16 network.

:
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

inputs2 = Input(shape=(max_length,))
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)
```

:
```
# train the model
epochs = 20
batch_size = 64
steps = len(train) // batch_size
#steps used for back propagation


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

2.1.6 Generate caption for an image –

```
# 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
```

2.1.7 Validate with test data –

Validate the By BLEU stands for Bilingual Evaluation Understudy. It is an algorithm, which has been used for evaluating the quality of the machine-translated text. We can use BLEU to check the quality of our generated caption. BLEU is language-independent, Easy to understand, it is easy to compute, It lies between [0,1]. Higher the score better the quality of the caption.

```
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)

# calculate 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)))
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
BLEU-1: 0.545905
BLEU-2: 0.330282
```

2.1.8 Visualize the Results–

```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(c):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = c.split('.')[0]
    img_path = directory + "/" + c
    image_show = Image.open(img_path)
    captions = map[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_show)
```

2.1.9 Improvement –

1. The performance of the model can be improved by training it on a larger dataset and hyperparameter tuning.

2. From above we can observe that unigram bleu scores favor flavor predictions.

3. Prediction is good if all the BLEU scores are high.

2.1.10 Conclusion –

A CNN-LSTM architecture has wide-ranging applications as it stands at the helm of Computer Vision and Natural Language Processing. It allows us to use state-of-the-art neural models for NLP tasks such as the transformer for sequential image and video data. At the same time, extremely powerful CNN networks can be used for sequential data such as the natural language. Hence, it allows us to leverage the useful aspects of powerful models in tasks they have never been used for before.