IMAGE CAPTIONING

Yanuka Deneth

FCC FINAL ASSIGNMENT - MS23001602

Table of Contents

Introduction

In this project, we will be looking at an Image Captioning Model from Scratch. The Model will take an image as input and print out a caption explaining the image.

Data Preparation

The Dataset has a collection of images and a text file that contains the captions for each image.

The text file needs to be cleansed of any unwanted symbols and spaces.

Extracting Features from Images

We'll be using the Neural Network created earlier to extract the features from the dataset and then save them into a variable.

```
# Dictionary for the features
features = {}

# Looping over every image inside the Flicker Extracted Directory
for img_name in tqdm(os.listdir(flickr_images)):
  # Calling the file name
  img_path = os.path.join(flickr_images, img_name)
  # Loading the image with a size of 224 x 224
  image = load_img(img_path, target_size = (224,224))
  # Convert the image into pixel arrays
  image = img_to_array(image)
  # Reshape the array for model
  image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
  # Preprocessing numpy array to convert from RGB to BGR, and zero-center each channel
  image = preprocess_input(image)
  # Extract Features via the model
  feature = model.predict(image, verbose = 0)
  # Get Image ID - Removing the extension
  image_id = img_name.split('.')[0]
  # Store Generated feature in the variable
  features[image_id] = feature
```

Splitting Image ID and Captions

```
[ ] # Dictionary to store the captions for image ID
    mapping = {}

    for line in tqdm(captions_doc.split('\n')):
      # Skipping lines that are bad
      if len(line) < 2:
        continue
      # Split the line by .jpg
      tokens = line.split('.jpg')
      # Saving Image ID and the rest as caption
      image_id, caption = tokens[0], tokens[1:]
      # Breaking into spaces and removing the unnecessary parts of the caption
      caption = " ".join(caption)
      caption = caption[3:]
      # Only adding if a feature exists for that image ID
      if image_id not in features:
        print(f"Not in Features - Image ID : {image_id}, Caption : {caption}")
        continue
      # If Image ID not already saved, so make an array for that image id
      if image_id not in mapping:
        mapping[image_id] = []
      # Add in the caption
      mapping[image_id].append(caption)
```

We'll be cleaning up the caption data for better results

```
[ ] # Before
    mapping['3249062399_0dafe5e4f5']
```

```
['Three Asian women , one elderly , one a child and one an adult crossing a street .',
 'Three women in hats walk down a sidewalk .',
 'Three women in white hats in a line .',
 'Three women , one old , one young and one a child , walk down a street .',
 'Three women walk in a line on the sidewalk .']
```

```
[ ] for key, captions in mapping.items():
      for i in range (len(captions)):
        # Take one caption
        caption = captions[i]
        # Convert to lowercase
        caption = caption.lower()
        # Delete digits and special characters
        caption = caption.replace("[^A-Za-z]", "")
        # Delete additional spaces
        caption = caption.replace("\s+", " ")
        # Add start and end tags
        caption = 'startseq ' + " ".join([word for word in caption.split() if len(word) > 1]) + ' endseq'
        # Set it back
        captions[i] = caption
```

```
[ ] # After
    mapping['3249062399_0dafe5e4f5']
```

```
['startseq three asian women one elderly one child and one an adult crossing street endseq',
 'startseq three women in hats walk down sidewalk endseq',
 'startseq three women in white hats in line endseq',
 'startseq three women one old one young and one child walk down street endseq',
 'startseq three women walk in line on the sidewalk endseq']
```

Model Architecture

The Model Architecture contains a model based on VGG-16. This was made from scratch using Tensorflow.

```
# Create the Model Object
model = Sequential()

# Layer 1
model.add(Conv2D(input_shape = (224, 224, 3), filters = 64, kernel_size = 3, padding = "same", activation = "relu"))

# Layer 2
model.add(Conv2D(filters = 64, kernel_size = 3, padding = "same", activation = "relu"))
model.add(MaxPool2D(pool_size = 2))

# Layer 3
model.add(Conv2D(filters = 128, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 128, kernel_size = 3, padding = "same", activation = "relu"))
model.add(MaxPool2D(pool_size = 2))

# Layer 4
model.add(Conv2D(filters = 256, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 256, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 256, kernel_size = 3, padding = "same", activation = "relu"))
model.add(MaxPool2D(pool_size = 2))

# Layer 5
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(MaxPool2D(pool_size = 2))

# Layer 6
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(Conv2D(filters = 512, kernel_size = 3, padding = "same", activation = "relu"))
model.add(MaxPool2D(pool_size = 2))

# Layer 7
model.add(Flatten())
model.add(Dense(units = 4096, activation = "relu"))
model.add(Dense(units = 4096, activation = "relu"))
```

The above model is used to extract the image features from the image.

The next model is used to train the network inputting an image and several captions.

```
# Image Feature Layers
inputs1 = Input(shape = (4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation = "relu")(fe1)

# Sequence Feature Layers
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)

# Compiling the Model
main_model = Model(inputs = [inputs1, inputs2], outputs = outputs)
main_model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics=["accuracy"])

# Plot the Model
plot_model(main_model, show_shapes = True)
```

Model Training

Since the training environment does not have enough specs to train the model enough times.

I used a loop to prevent the RAM from flooding which cleared out the list every loop.

```
# Training stats
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
  # Iteration Setup
  log_dir = f'logs/epochs_{epochs}'
  tensorboard_callback = TensorBoard(log_dir=log_dir)
  # Creating Data Generator
  generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
  # Fitting for one Epoch
  main_model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1, callbacks=[tensorboard_callback])
  # Printing Epoch
  print(f"Epoch {i+1}")
```

Furthermore, I saved and loaded up the weights everytime to make sure the network trains on-top of the previous weights.

```
[ ] # Loading weights
    main_model.load_weights(f"{EXPORT_MODEL_DIR}/main_model_weights.hdf5")
```

```
[ ] # Saving Weights
    main_model.save_weights(f"{EXPORT_MODEL_DIR}/main_model_weights.hdf5")
```

Results

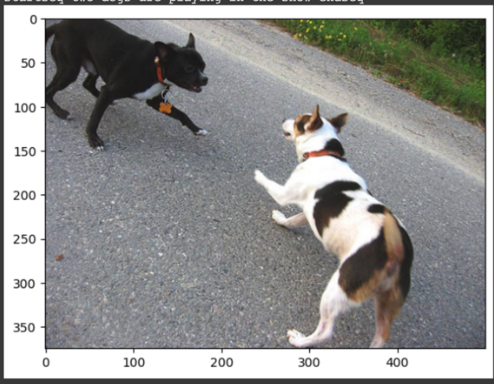
The final trained model gave out the following :

```
loss: 1.7236 - accuracy: 0.5415
```

Here is an image example :

```
generate_caption("1001773457_577c3a7d70.jpg")
```

```
--------------------Actual--------------------
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
-------------------Predicted-------------------
startseq two dogs are playing in the snow endseq
```



Conclusion

The network can be improved more by tweaking around the epochs and how the model is trained in batches.

Targeting to improve the BLEU value will help in the overall model.

```python
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(main_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)))
```

```
100% 1619/1619 [18:32<00:00, 1.44it/s]
BLEU-1: 0.399939
BLEU-2: 0.169213
```