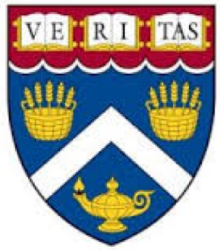
Final Project

Generation of Captions from Images

Menon, Surya



CSCI S-89a Deep Learning, Summer 2019

Harvard University Extension School

Prof. Zoran B. Djordjević

Flickr8k Dataset

- 8000 images (mostly of everyday scenes and objects) taken from Flickr, with each image having 5 associated captions
  - 6000 predefined training images
  - 1000 predefined validation and test images

man is fishing in foggy lake .

man fishing near large tree .

man fishes under large tree .

man fishes by tree in the morning mist .

fisherman fishes at the bank of foggy river .

the dog is wearing purple cape .

brown dog with green and purple cape runs on grass .

small dog wearing purple and green cape .

small brown dog is running across the grass wearing purple and green coat .

miniature dachshund has an apron on its back .

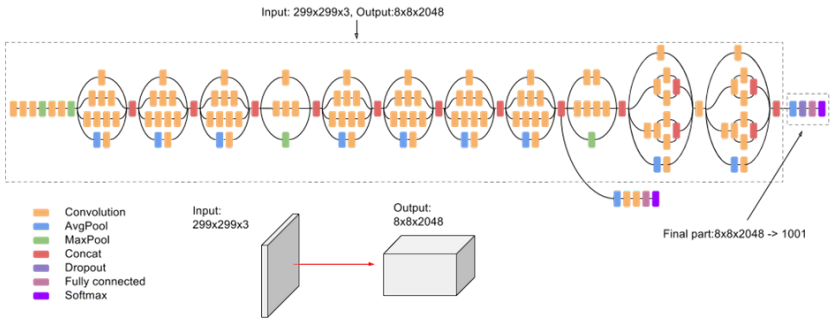
- Can request access here: https://forms.illinois.edu/sec/1713398

Image Feature Extraction

- Use pre-trained convolutional neural network to extract image features

In [7]:
```
# feature dictionary (index by image name)
imgs = dict()

# go through images in folder
for i in os.listdir(img_folder):
    # print image name - just for tracking
    print(i)
    # create path
    path = img_folder + i
    # load image with InceptionV3 default
    img = image.load_img(path, target_size=(299,299))
    # prepare image for CNN model
    img = img_size(img)
    # get features
    feat = feature_mod.predict(img)
    feat = np.reshape(feat, feat.shape[1])
    # pass to dictionary
    imgs[i.split('.')[0]] = feat

# get number of images in folder
print(len(imgs))

# save for later
pickle.dump(imgs, open('img_features.pkl', 'wb'))
```

```
1000268201_693b08cb0e.jpg
1001773457_577c3a7d70.jpg
1002674143_1b742ab4b8.jpg
1003163366_44323f5815.jpg
```



- Create 2048 dimensional vector representations

Text Preparation

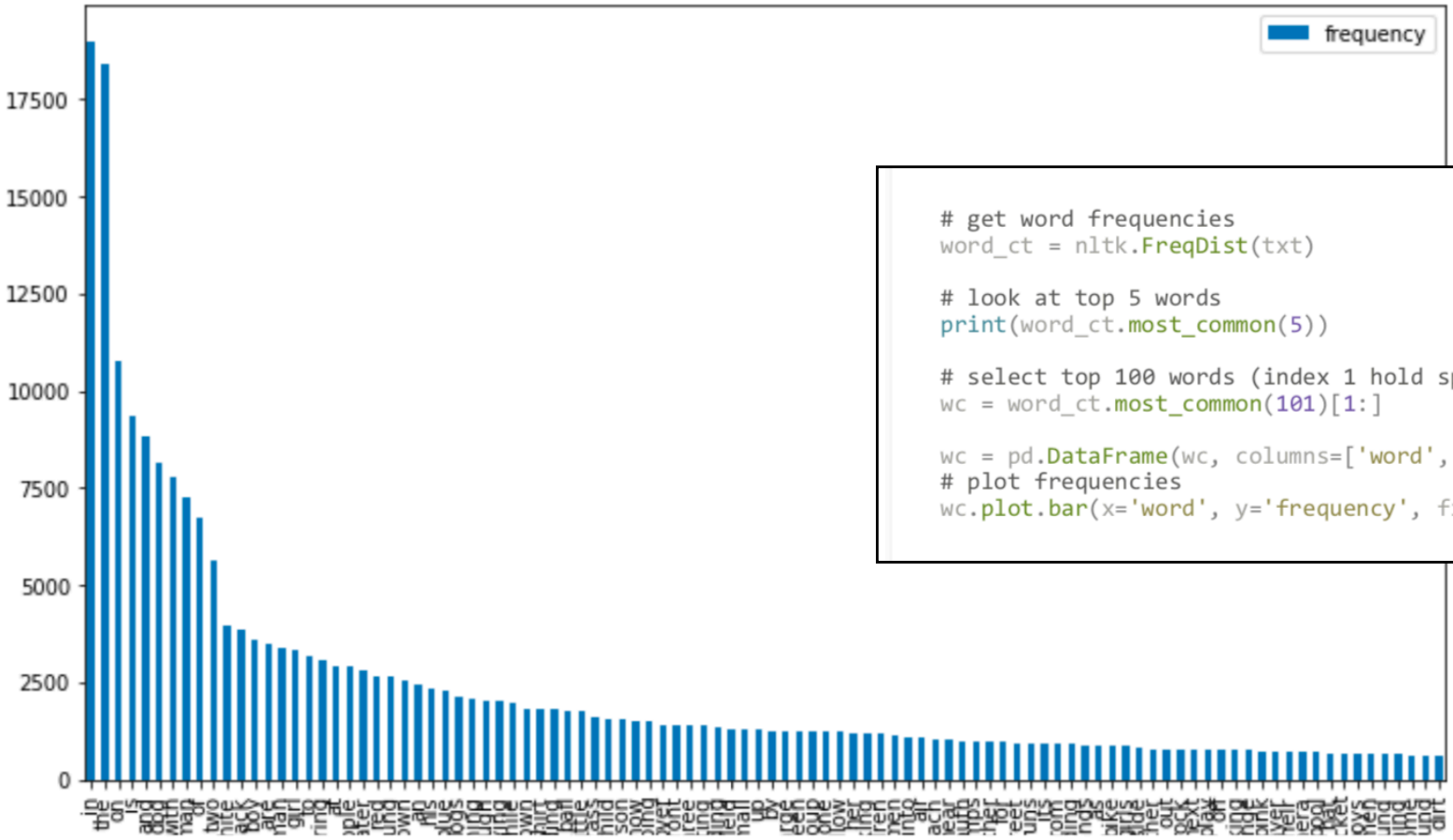
- Clean up captions

```
# make captions clean
def new_caps(s):
    # take out punctuation
    # https://stackoverflow.com/questions/265960/best-way-to-strip-punctuation-from-a-string
    s = s.translate(str.maketrans('', '', string.punctuation))
    # make lower case (prevent duplicates)
    s = s.lower()
    # no words with numbers
    s = ' '.join(w for w in s.split() if not any(d.isdigit() for d in w))
    # no 1 letter words (i.e. s that follows removed apostrophe)
    s = ' '.join(w for w in s.split() if len(w)>1)
    return s

# create dictionary of photo_id and captions
captions = dict()
# each line
for line in doc.split('\n'):
    # skip last empty line
    if len(line) < 1:
        continue
    # split on .jpg to get image name, 1st token is image id
    tokens = line.split('.')
    img_id = tokens[0]
    # check pics with no captions
    if '\t' not in tokens[1]:
        cap = tokens[1]
    # split 2nd token into descriptions, after \t, put in list
    else:
        cap = tokens[1].split('\t')[1]
        # make lower case, remove punctuation
        cap = new_caps(cap)
        # put start and end tokens
        cap = '< ' + cap + ' >'
    # append caption to list of associated image entry
    if img_id not in captions:
        captions[img_id] = list()
    captions[img_id].append(cap)
```

```
['1000268201_693b08cb0e.jpg#0\tA child in a pink dress is climbing up a set of stairs in an entry way .',
 '1000268201_693b08cb0e.jpg#1\tA girl going into a wooden building .',
 '1000268201_693b08cb0e.jpg#2\tA little girl climbing into a wooden playhouse .',
 '1000268201_693b08cb0e.jpg#3\tA little girl climbing the stairs to her playhouse .',
 '1000268201_693b08cb0e.jpg#4\tA little girl in a pink dress going into a wooden cabin .']
```

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

Vocabulary




```
# get word frequencies
word_ct = nltk.FreqDist(txt)

# look at top 5 words
print(word_ct.most_common(5))

# select top 100 words (index 1 hold space so exclude)
wc = word_ct.most_common(101)[1:]

wc = pd.DataFrame(wc, columns=['word', 'frequency'])
# plot frequencies
wc.plot.bar(x='word', y='frequency', figsize=(12,7))
```

- Vocabulary size: **8761**
- Total word count: **413286**

Tokenization and Embeddings

```
# use pretrained word embeddings - follow lecture examples
import io

embeddings_index = {}
# Google Colab file
with io.open('/content/drive/My Drive/nlp_project_data/glove.6B.300d.txt', encoding='utf8') as f:
# file path
# with io.open('glove.6B.300d.txt', encoding='utf8') as f:
  for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:],dtype='float32')
    embeddings_index[word] = coefs

# embedding dimension layer
embedding_dim = 300

# create embedding matrix with weights, of appropriate size
embedding_matrix = np.zeros((vocab_size, embedding_dim))
# go through tokenizer
for w, i in tokens.word_index.items():
  embedding_vector = embeddings_index.get(w)
  # if within vocab size
  if i < vocab_size:
    if embedding_vector is not None:
      # add vector representation
      embedding_matrix[i] = embedding_vector

# ALTERNATIVE - save/load embedding matrix
# save embeddings (can load later)
# np.save('em3500_300d.npy', embedding_matrix)
# load embedding matrix -
# embedding_matrix = np.load('em3500_300d.npy')

# check shape
embedding_matrix.shape
```

```
(3501, 300)
```

- Encode words to integers using tokenizer with vocab size of 3500
- Pre-trained GloVe 300-dim vectors for Embedding layer of model

Model Creation

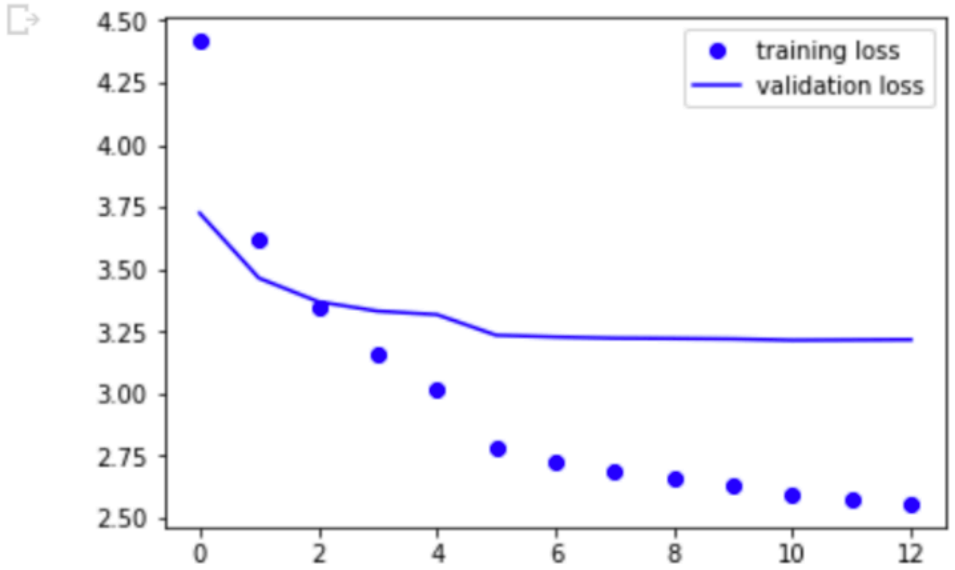
- "Merge" model
  - 2 inputs – image features vector and text vector (start token)
  - Concatenate inputs, pass through recurrent network
  - Output is next word in sequence

```
# create model

# get max_len of sequence for training
max_len = max_length(train_caps)

# take photo features of (2048, ) input
img_inputs = Input(shape=(2048,))
# handle overfitting
feats1 = Dropout(0.5)(img_inputs)
# reduce size
feats2 = Dense(300, activation='relu')(feats1)
# so same size as text input when merge
feats3 = RepeatVector(max_len)(feats2)

# pass text sequences
# expect max_len inputs since padded
txt_inputs = Input(shape=(max_len,))
# put in embedding - size of vocab, 0 is padding
# add embedding matrix, freeze (don't update weights)
seq1 = Embedding(vocab_size, 300, weights=[embedding_matrix], trainable=False,
                 mask_zero=True)(txt_inputs)

# merge inputs
merge1 = add([feats3, seq1])
# handle overfitting
merge2 = Dropout(0.5)(merge1)
# gru model to read sequences, make bidirectional
merge3 = Bidirectional(GRU(500, return_sequences=False))(merge2)
# generate classification/output layer
outputs = Dense(vocab_size, activation='softmax')(merge3)

# pass inputs and outputs into model
model = Model(inputs=[img_inputs, txt_inputs], outputs=outputs)

# look at layers
print(model.summary())

# compile model - set loss, optimizer, learning rate
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(lr=0.0005))
```

Training Results

- Track training and validation losses
  - Observe overfitting

```
def plot_loss(loss, val):
  epochs=range(len(loss))
  plt.figure()
  plt.plot(epochs, loss, 'bo', label='training loss')
  plt.plot(epochs, val, 'b', label='validation loss')
  plt.legend()
  plt.show()

# plot training and validation losses
plot_loss(loss, val_loss)
```



Model Evaluation

- Corpus BLEU scores to evaluate model quality
  - 1.0 = exact match, 0.0 = complete mismatch

We discussed BLEU scores during lecture, and according to a couple of references including this site, we can use the corpus BLEU scores to see how close the predicted and actual captions are to each other. We loosely follow lecture notes to create these scores:

```
# evaluate model on test images using greedy search
def bleu(mod, test_caps, test_imgs, tokens, max_len):
  act, pred = list(), list()
  for i, caps in test_caps.items():
    img = test_imgs[i].reshape(1,2048)
    p = make_cap(mod, tokens, img, max_len)
    act.append([c.split() for c in caps])
    pred.append(p.split())
  print('BLEU-1: %f' % corpus_bleu(act, pred, weights=(1.0, 0, 0, 0)))
  print('BLEU-2: %f' % corpus_bleu(act, pred, weights=(0.5, 0.5, 0, 0)))
  print('BLEU-3: %f' % corpus_bleu(act, pred, weights=(0.3, 0.3, 0.3, 0)))
  print('BLEU-4: %f' % corpus_bleu(act, pred, weights=(0.25, 0.25, 0.25, 0.25)))
```

```
bleu(mod, test_caps, test_imgs, tokens, max_len)
```

```
BLEU-1: 0.583389
BLEU-2: 0.348187
BLEU-3: 0.250425
BLEU-4: 0.128517
```

- BLEU-1 highest
- Effect of limited vocabulary from just training data

Demo

- Predict results on an actual test image
  - Greedy and beam search captions

In [4]:
```
# create greedy captions - from model_create.ipynb
def make_cap(model, tokens, img, max_len):
    # start token
    in_txt='<'
    # go over entire possible sequence (of max length)
    for i in range(max_len):
        # encode input - similar set up to generator
        seq = tokens.texts_to_sequences([in_txt])[0]
        # pad
        seq = pad_sequences([seq], maxlen=max_len)
        # get next word in sequence
        pred = model.predict([np.array(img), seq])
        # get encoded of highest probability
        pred = np.argmax(pred)
        # decode output
        w = idtw(pred, tokens)
        # break if cannot map
        if w is None:
            break
        # append to sequence
        in_txt += ' ' + w
        # stop if reach end token
        if w == '>':
            break
    #txt = in_txt.split()[1:-1]
    #txt = ' '.join(txt)
    return in_txt
```

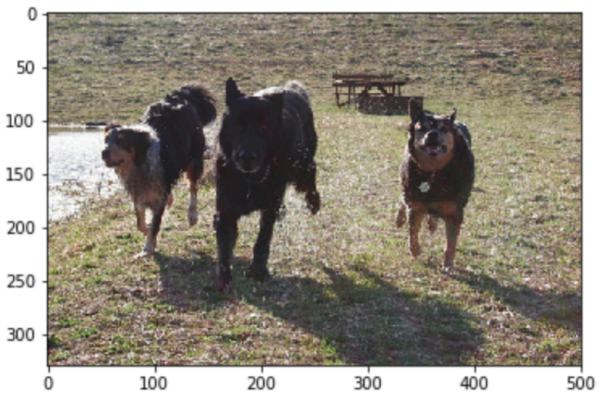
In [5]:
```
# create beam search - from model_create.ipynb
def beam_caps(model, tokens, img, max_len, beam_index=3):
    # start sequence
    in_txt = [[tokens.texts_to_sequences(['<'])[0], 0.0]]

    # while less than max length
    while len(in_txt[0][0]) < max_len:
        temp = []
        # go through sequence
        for i in in_txt:
            # pad input sequences
            seq = pad_sequences([i[0]], maxlen=max_len, padding='post')
            # get predictions for next word
            pred = model.predict([np.array(img), seq])
            # sort and pick top beam_index possibilities
            word_preds = np.argsort(pred[0])[-beam_index:]

            # create new lists from top picks - words and probabilities
            # incorporate with model prediction
            for w in word_preds:
                # get current seq, prob
                next_cap, prob = i[0][:], i[1]
                # add word and associated probability
                next_cap.append(w)
                prob += pred[0][w]
                temp.append([next_cap, prob])

        # hold possibilities
        in_txt = temp
        # sort probabilities - lambda identifies value
        in_txt = sorted(in_txt, key=lambda p: p[1])
        # pick top in full list of options
        in_txt = in_txt[-beam_index:]

    # pick top word in all
    in_txt = in_txt[-1][0]
    # decode
    inter_cap = [idtw(i, tokens) for i in in_txt]

    # construct caption
    final_cap = []
    for i in inter_cap:
        # if not end token
        if i != '>':
            final_cap.append(i)
        else:
            break

    # take out start token, make string
    final_cap = ' '.join(final_cap[1:])
    return final_cap
```

Demo (cont.)

In [70]:
```
# acquire greedy and beam searches and display an image - from model_create.ipynb
def print_img_caps(path, mod, tokens, img, max_len):
    # plot image
    x= plt.imread(path)
    plt.imshow(x)
    # reshape image features
    img = img.reshape(1,2048)
    print('\033[1m', 'Image ID:','\033[0m', path.split('\\')[-1])
    # get greedy caption
    txt = make_cap(mod, tokens, img, max_len)[1:-1]
    # get beam searches
    print('Greedy:', txt)
    print('Beam, k=3:', beam_caps(mod, tokens, img, max_len))
    print('Beam, k=5:', beam_caps(mod, tokens, img, max_len, beam_index=5))
    print('Beam, k=7:', beam_caps(mod, tokens, img, max_len, beam_index=7))
```

In [71]:
```
# test image
i ='3462454965_a481809cea.jpg'
path = os.path.join(img_folder, i)
print_img_caps(path, mod, tokens, test_imgs[i.split('.')[0]], max_len)
```

```
 Image ID:  3462454965_a481809cea.jpg
Greedy:  black dog is running through the grass
Beam, k=3: black dog and brown dog are playing in the grass
Beam, k=5: the black dog is running through the grass
Beam, k=7: the black dog is running through the green grass
```



Next Steps

- Deal with overfitting
  - Increased data set size (MS-COCO)
  - Data augmentation
  - Cross-validation
- Larger variety of training images/contexts
- Alternate CNN model for feature extraction
- Learning word embeddings
- Alternate model structures
- Attention mechanism

YouTube link

- Insert link

References

- **Dataset**
  - https://forms.illinois.edu/sec/1713398
  - Alternative: https://github.com/jbrownlee/Datasets/releases

- **Project file names:**
  - **`project_img-features.ipynb`** – image feature extraction
  - **`model_create.ipynb`** – data processing and model creation/evaluation
  - **`model_demo.ipynb`** – predict captions on test images

- **References:**

  - Karpathy, A., & Fei-Fei, L. (2015). Deep Visual-Semantic Alignments for Generating Image Descriptions. Retrieved 27 July 2019, from https://cs.stanford.edu/people/karpathy/cvpr2015.pdf.
  - Brownlee, J. (2017). How to Develop a Deep Learning Photo Caption Generator from Scratch. Retrieved 27 July 2019, from https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/.
  - Vinyals, O., Toshev, A., Benigo, S., & Erhan, D. (2015). Show and Tell: A Neural Image Caption Generator. Retrieved 27 July 2019, from https://arxiv.org/pdf/1411.4555.pdf.
  - Applications - Keras Documentation. Retrieved 27 July 2019, from https://keras.io/applications/#inceptionv3.

References (cont.)

- ImageNet. Retrieved 27 July 2019, from https://en.wikipedia.org/wiki/ImageNet.
- IOPub data rate exceeded. The notebook server will temporarily stop sending output to the client in order to avoid crashing it. (2019). Retrieved 27 July 2019, from https://www.drjamesfroggatt.com/python-and-neural-networks/iopub-data-rate-exceeded-the-notebook-server-will-temporarily-stop-sending-output-to-the-client-in-order-to-avoid-crashing-it/.
- Koehrsen, W. (2018). Neural Network Embeddings Explained. Retrieved 27 July 2019, from https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526.
- Using word embeddings. Retrieved 27 July 2019, from https://jjallaire.github.io/deep-learning-with-r-notebooks/notebooks/6.1-using-word-embeddings.nb.html.
- Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. Retrieved 27 July 2019, from https://nlp.stanford.edu/projects/glove/.
- Lamba, H. (2018). Image Captioning with Keras. Retrieved 27 July 2019, from https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8.
- Tanti, M., Gatt, A., & Camilleri, K. (2018). Where to put the Image in an Image Caption Generator. Retrieved 27 July 2019, from https://arxiv.org/pdf/1703.09137.pdf.
- Surmenok, P. (2017). Estimating an Optimal Learning Rate For a Deep Neural Network. Retrieved 27 July 2019, from https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0.
- Brownlee, J. (2017). A Gentle Introduction to Calculating the BLEU Score for Text in Python. Retrieved 27 July 2019, from https://machinelearningmastery.com/calculate-bleu-score-for-text-python/.

References (cont.)

- Why is beam search required in sequence to sequence transduction using recurrent neural networks? - Quora. (2017). Retrieved 27 July 2019, from https://www.quora.com/Why-is-beam-search-required-in-sequence-to-sequence-transduction-using-recurrent-neural-networks.
- Coursera. *Beam Search* [Video]. Retrieved from https://www.coursera.org/lecture/nlp-sequence-models/beam-search-4EtHZ.
- Mustafa, F. (2018). Keras implementation of Image Captioning Model. Retrieved 27 July 2019, from https://medium.com/@faizanmustafa75/keras-implementation-of-image-captioning-model-3a7ab68e67d4.
- COCO - Common Objects in Context. Retrieved 27 July 2019, from http://cocodataset.org/#overview.