به نام خدا

گزارش پروژه درس یادگیری عمیق

Image Captioning

استاد درس: **دکتر فاطمی زاده**

علی صابری

۹۷۲۰۵۹۴۴

زمستان ۱۳۹۸

مقدمه

هدف از انجام این پروژه، طراحی شبکه ای است که یک تصویر را به عنوان ورودی می گیرد و در خروجی یک caption مناسب از جزئیات عکس تولید می کند.

همانطور که در دستور پروژه گفته شده است، یک ساختار تولید Caption برای تصویر، از دو شبکه تشکیل می شود:

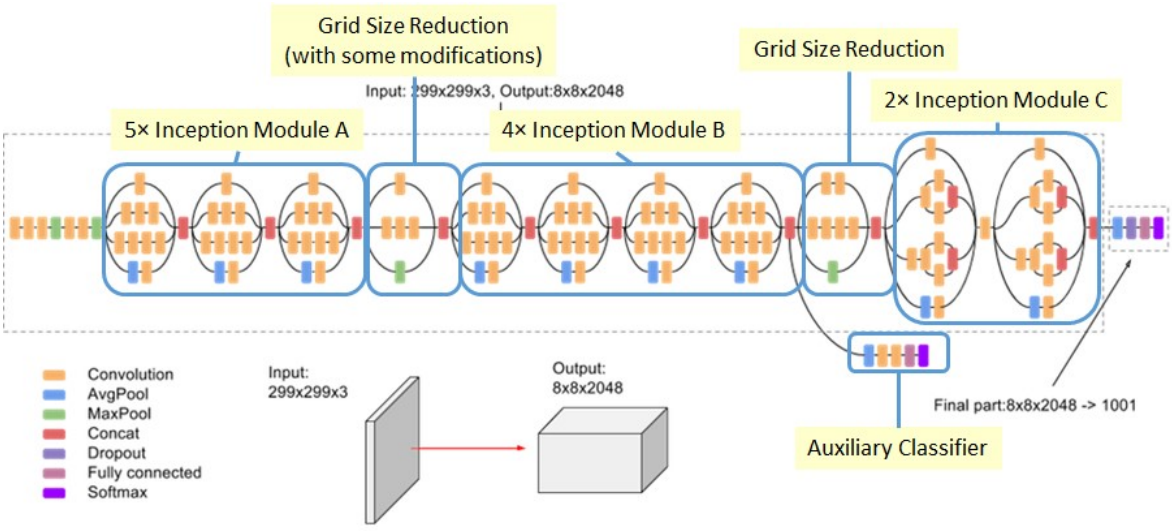
۱- شبکه CNN که اطلاعات فضایی و ویژگی های تصویر را استخراج می کند.

۲- شبکه RNN که دنباله واژگان را با توجه به بردار ویژگی تصویر می سازد.

در ادامه بخش های مختلف پروژه به توضیح داده شده است.

شبکه استخراج ویژگی

این قسمت همان طور که گفته شد برای استخراج اطلاعات فضایی و ویژگی های تصویر استفاده می شود. ساختار این شبکه به صورت کانولوشنی است. پایه شبکه طراحی شده برای این قسمت از شبکه Inception-v3 گرفته شده است. ساختار شبکه Inception-v3 به صورت زیر است:



همان طور که مشاهده می شود این شبکه از ۴۲ لایه مختلف تشکیل شده است که تصاویر ورودی با ابعاد 3*299*299 را دریافت می کند و انتهای قسمت کانولوشنی خود این خروجی با ابعاد 2048*8*8 تولید می کند.

هدف اصلی در این بخش در واقع طراحی یک طبقه بند multi-label است که از لایه های ابتدایی آن به عنوان استخراج‌کننده ویژگی در مدل اصلی استفاده خواهد شد.

کد مربوط به این مدل در فایل classifier.py قرار دارد. در ادامه با قسمت های مختلف کد آشنا می شویم.

```
with open('categories_info.json') as categories_file:
    categories = json.load(categories_file)

with open('labels.json') as labels_file:
    labels = json.load(labels_file)

with open('images_info.json') as images_info_file:
    images_info = json.load(images_info_file)

# image ID --> file_name
image_files = {image['id']: image['file_name'] for image in images_info}

# category ID --> category_name
categories = {category['id']: category['name'] for category in categories}
category_ids = list(categories.keys())

n_classes = len(categories)

# vector of ones and zeros for multi label classification
targets = {image_id: np.zeros((n_classes)) for image_id in image_files.keys()}

for label in labels:
    targets[label['image_id']][category_ids.index(label['category_id'])] = 1.
```

در این قسمت کد، فایل های categories_info.json، labels.json و images_info.json خوانده می شوند و با توجه به اطلاعات موجود در این فایل ها، دیکشنری های Image_files، category_ids و targets ساخته می شوند. کلیدها و مقادیر این سه دیکشنری به صورت زیر است:

image_files: Key = image ID, Value = image file name

category_ids: Key = category ID, Value = category name

targets: Key = image Id, Value = binary array for labels associated with image

در مورد دیکشنری targets، این دیکشنری همان خروجی های مطلوب طبقه بند multi-label را شامل می شود که به ازای هر تصویر یک آرایه باینری با طول ۸۰ (تعداد کلاس ها) وجود دارد که تعلق یا عدم تعلق به هر کلاس را نشان می دهد.

```
image_folder = './val2017/'

# image = load_img('./val2017/000000000139.jpg', target_size=(299, 299))

images = np.array([img_to_array(load_img(image_folder + image_file, target_size=(299, 299))) for image_file in image_files.values()])
targets = np.array([targets[image_id] for image_id in image_files.keys()])

x_train, x_validation, y_train, y_validation = train_test_split(images, targets, test_size=0.2, random_state=100)

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   rotation_range = 30,
                                   zoom_range = 0.3,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   horizontal_flip = True,
                                   fill_mode = "nearest")
train_generator = train_datagen.flow(x_train, y_train, shuffle=True, batch_size=32, seed=10)

val_datagen = ImageDataGenerator(rescale = 1./255)
val_generator = train_datagen.flow(x_validation, y_validation, shuffle=False, batch_size=32, seed=10)
```

در این قسمت کد، پردازش های لازم روی تصاویر ورودی انجام می شود. برای این منظور ابتدا تصاویر با استفاده از دیکشنری image_files به یک آرایه با ابعاد (5000, 299, 299, 3) و دیکشنری target به یک آرایه با ابعاد (5000, 80) تبدیل می شود. در ادامه داده ها با نسبت 0.2 به داده های train و validation تقسیم می شوند. برای افزایش داده های آموزشی از روش data augmentation استفاده شده است. پیش پردازش دیگری که روی داده های train و validation انجام می شود، اسکیل کردن مقادیر پیکسل ها به مقادیر بین 0 تا 1 است.

```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))

# global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# fully-connected layer
x = Dense(1024, activation='relu')(x)

# output layer
predictions = Dense(n_classes, activation='sigmoid')(x)

# classifier model
model = Model(inputs=base_model.input, outputs=predictions)

# freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

model.compile(Adam(lr=.0001), loss='binary_crossentropy', metrics=["accuracy"])

model.fit_generator(train_generator,
                    steps_per_epoch = len(x_train) // 32,
                    validation_data = val_generator,
                    epochs = 10,
                    verbose = 1)

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

در این قسمت مدل شبکه طبقه بند ساخته می شود. در ابتدا، مدل پایه (بدون لایه های fully-connected انتهایی) شبکه Inception-v3 خوانده می شود. در ادامه خروجی این قسمت به یک لایه GlobalAveragePooling داده می شود تا به صورت خطی تبدیل شود. سپس یک لایه fully-connected با 1024 نورون و تابع فعالسازی ReLU (خروجی این لایه بردار ویژگی هر تصویر را استخراج می کند که در قسمت بعدی مورد استفاده قرار می گیرد) و درنهایت لایه خروجی با 80 نورون و تابع فعالسازی sigmoid قرار دارد. علت این که در لایه خروجی از تابع فعالسازی sigmoid به جای softmax استفاده شده است این است که در اینجا با یک مسئله multi-label classification سروکار داریم که در واقع چندین مسئله binary classification را شامل می شود و نتیجه هر نورون خروجی باید به صورت مستقل درنظر گرفته شود.

در هنگام آموزش شبکه، از آموزش قسمت های اضافه شده از شبکه Inception-v3 صرف نظر شده است. البته قسمتی از کد برای fine-tuning لایه های انتهایی به کار برده شده از شبکه Inception-v3 نوشته شده است ولی با توجه به این که دقت بدون درنظرگرفتن fine-tuning نیز قابل ملاحظه است، این قسمت در مدل نهایی طبقه بند لحاظ نشده است.

در نهایت پس از آموزش شبکه، دقت طبقه بندی برای داده های train 0.9753 و برای داده های validation 0.9724 به دست آمد.

تنظیمات مربوط به شبکه و نحوه آموزش شبکه به صورت زیر است:

- batch size = 32
- optimizer = Adam
- learning rate = 0.0001
- loss = binary cross entropy
- epochs = 10

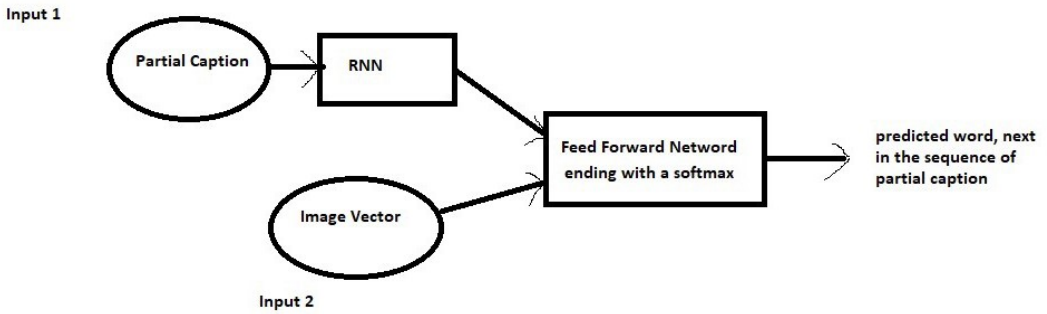
```
test_image_folder = './val2017/'

# image = load_img('./val2017/000000000139.jpg', target_size=(299, 299))

test_images = np.array([img_to_array(load_img(image_folder + image_file, target_size=(299, 299))) for image_file in image_files.values()])
test_targets = np.array([targets[image_id] for image_id in image_files.keys()])

test_images = test_images / 255.

test_pred = clf.predict(test_images)

accuracy = binary_accuracy(test_targets, test_pred)

print("Test accuracy = {}".format(accuracy))

######################

image_path = None

image = np.array([img_to_array(load_img(image_path), target_size=(299, 299))])
image = image / 255.

pred = model.predict(image)

binary_pred = [p > 0.5 for p in pred]

labels = category_ids.values()[binary_pred]
```

تست طبقه بند به دو صورت انجام می شود:

- یک فولدر حاوی تصاویر تست معرفی می شود (test_image_folder)، تصاویر این فولدر پیش پردازش می شوند و تارگت های این تصاویر نیز مشخص می شوند (البته چون فرمت برچسب های تست مشخص نیست، کدی برای استخراج تارگت ها نوشته نشده است). سپس این تصاویر به مدل داده می شوند تا پیش بینی انجام شود. در نهایت دقت مدل محاسبه و گزارش می شود.
- مسیر یک تصویر مشخص می شود (image_path)، پس از پیش پردازش خروجی مدل به دست می آید سپس با روش thresholding لیبل های تصویر مشخص می شود.

شبکه تولید کننده Caption

این قسمت که بخش اصلی پروژه را شامل می شود. در این قسمت قرار است با توجه به بردار ویژگی تصویر و دنباله ای از واژگان، واژه بعدی دنباله پیش بینی شود. ساختار شبکه مورد استفاده به صورت RNN است و فرم کلی زیر را دارد:



کد مربوط به این قسمت در فایل caption_generator.py قرار دارد.

```
# image ID --> file_name
image_files = {image['id']: image['file_name'] for image in images_info}

image_ids = list(image_files.keys()) # list of image IDs

# image ID --> list of captions
descriptions = {image_id: [] for image_id in image_ids}

for caption in captions:
    descriptions[caption['image_id']].append(caption['caption'])

table = str.maketrans('', '', string.punctuation)
for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc = desc_list[i]
        # tokenize
        desc = desc.split()
        # convert to lower case
        desc = [word.lower() for word in desc]
        # remove punctuation from each token
        desc = [w.translate(table) for w in desc]
        # remove hanging 's' and 'a'
        desc = [word for word in desc if len(word)>1]
        # remove tokens with numbers in them
        desc = [word for word in desc if word.isalpha()]
        # store as string
        desc_list[i] =  'startseq ' + ' '.join(desc) + ' endseq'
```

در این قسمت پس از خواندن فایل json، دیکشنری های image_files و descriptions تشکیل می شود. ساختار این دیکشنری ها به صورت زیر است:

image_files: Key = image ID, Value = image file name

descriptions: Key = image ID, Value = a list containing all captions for the image

در ادامه کپشن های موجود در دیکشنری description پیش پردازش می شوند. پیش پردازش هایی که انجام می شود به صورت زیر است:

- تبدیل حروف کلمات به حروف کوچک
- حذف علائم نگارشی
- حذف ‘s’ و a از کپشن ها
- حذف اعداد
- اضافه کردن واژه های ‘startseq’ و ‘endseq’ به ابتدا و انتهای کپشن ها

```
all_captions = []
for key, val in descriptions.items():
    for cap in val:
        all_captions.append(cap)

# Consider only words which occur at least 10 times in all captions
word_count_threshold = 10
word_counts = {}

for sent in all_captions:
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

# Vocabulary of frequent words
vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
vocab_size = len(vocab) + 1
```

در این قسمت تعداد دفعات تکرار کلیه کلمات منحصر به فرد در همه کپشن ها محاسبه می شوند و کلمات با حداقل ۱۰ مرتبه تکرار، در لیستی به عنوان vocab نگه داری می شود. تعداد این کلمات به همراه ‘0’ (که بعدا برای padding استفاده می شود) برابر 1525 است.

```
images = np.array([img_to_array(load_img(image_folder + image_file, target_size=(299, 299))) for image_file in image_files.values()])

images_ = images / 255.
```

در این قسمت تصاویر با استفاده از دیکشنری image_files خوانده می شوند و به بازه 0 تا 1 اسکیل می شوند.

```
# load classifier model
model = load_model('classifier_model.h5')

# create a new model without considering output layer of classifier model
model_new = Model(model.input, model.layers[-2].output)

# extract feature vectors
feature_vectors = model_new.predict(images_)

# image IDs --> feature vector
temp_zip = zip(image_ids, feature_vectors)
features_dict = dict(temp_zip)

# save feature vectors
pickle_out = open("feature_vectors.pickle", "wb")
pickle.dump(features_dict, pickle_out)
pickle_out.close()
```

در این قسمت، مدل آموزش داده شده در قسمت قبل لود می شود. مدل جدیدی با درنظر نگرفتن لایه خروجی مدل قبل ساخته می شود. از این مدل برای استخراج ویژگی های تصاویر استفاده خواهد. در نهایت با توجه به تصاویر خوانده شده، بردارهای ویژگی تصاویر به دست می آید و در دیکشنری با عنوان feeatures_dict ذخیره می شود. همچنین برای استفاده های آتی از این بردارهای ویژگی، این دیکشنری در فایلی با عنوان feature_vectors.pickle ذخیره می شود.

```
# index --> word
ixtoword = {}
# word --> index
wordtoix = {}

ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1

# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):

    all_desc = list()

    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]

    return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# determine the maximum sequence length
max_length = max_length(descriptions)
print('Max Description Length: %d' % max_length)
```

در این قسمت تو دیکشنری ixtoword و wordtoix با استفاده از کلمات موجود در لیست vocab ساخته می شوند. ساختار این دیکشنری ها به صورت زیر است:

wordtoix: Key = word, Value = index

ixtoword: Key = index, Value = word

در ادامه با استفاده از دیکشنری descriptions، حداکثر طول کپشن به دست می آید. این مقدار برابر 47 است.

```
def data_generator(descriptions, features, wordtoix, max_length):
    X1, X2, y = list(), list(), list()

    for key, desc_list in descriptions.items():
        # retrieve the image feature
        image = features[key]
        for desc in desc_list:
            # encode the sequence
            seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
            # split one sequence into multiple X, y pairs
            for i in range(1, len(seq)):
                # split into input and output pair
                in_seq, out_seq = seq[:i], seq[i]
                # pad input sequence
                in_seq = pad_sequences([in_seq], maxlen=max_length, padding='post')[0]
                # encode output sequence
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                # store
                X1.append(image)
                X2.append(in_seq)
                y.append(out_seq)

    return np.array(X1), np.array(X2), np.array(y)

X1, X2, y = data_generator(descriptions, features_dict, wordtoix, max_length)
X1_train, X1_validation, X2_train, X2_validation, y_train, y_validation = train_test_split(X1, X2, y, test_size=0.2, random_state=100)
```

در این قسمت، با استفاده از دیکشنری descriptions که شامل کپشن های هر تصویر است و دیکشنری features_dict که شامل بردار ویژگی 1024تایی هر تصویر است، داده های ورودی و خروجی با استفاده از تابع data_generator تولید می شوند. ورودی X1، بردارهای ویژگی تصاویر، ورودی X2، کپشن های جزئی (که به اندازه بزرگترین طول کپشن که 47 است pad شده اند) و خروجی y کلمه بعدی دنباله (به صورت one-hot براساس به کلمات موجود در vocab) با توجه به بردار ویژگی و کپشن جزئی متناظر را شامل می شود.

در ادامه نیز، داده ها به دو دسته train و validation تقسیم می شوند.

```
# Load Glove vectors
embeddings_index = {}
f = open('glove.6B.200d.txt', encoding="utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

embedding_dim = 200
# Get 200-dim dense vector for each of words in vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in wordtoix.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

این قسمت پیاده سازی word2vec را انجام می دهد. برای این منظور از بردارهای آماده Glove استفاده شده است که به ازای هر کلمه موجود در vocab، یک بردار با طول 200 تولید می کند و در ماتریسی با ابعاد 200*1525 (تعداد کلمات موجود در vocab) ذخیره می کند (embedding_matrix). از این ماتریس بعدا به عنوان وزن های لایه Embedding استفاده می شود.

```
# image feature extractor model
inputs1 = Input(shape=(1024,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

# partial caption sequence model
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)

# decoder (feed forward) model
decoder1 = Add()([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

# merge the two input models
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

model.compile(loss='categorical_crossentropy', optimizer='adam')
```

در این قسمت شبکه موردنظر پیاده سازی می شود. همان طور که مشاهده می شود، شبکه دو شاخه مجزا دارد:

۱- برداری ویژگی 1024تایی را به عنوان ورودی می گیرد و پس از dropout، با استفاده از یک لایه fully-connected 256تایی با تابع فعالسازی ReLU به یک بردار 256تایی مپ می کند.

۲- دنباله با طول ثابت 47 را به عنوان ورودی می گیرد و در لایه Embedding به صورت دنباله ای از بردارهای با طول 200 تبدیل می کند. در ادامه پس از dropout، به یک لایه LSTM با 256 واحد پردازشی می دهد.

خروجی های این دو شاخه با یکدیگر جمع می شوند و به یک لایه fully-connected با 256 نورون داده می شوند. لایه خروجی نیز یک لایه softmax با طولی به اندازه vocab_size است.

تنظیمات مربوط به شبکه و نحوه آموزش شبکه به صورت زیر است:

- batch size = 64
- loss = categorical cross entropy
- optimizer = Adam
- epochs = 5

```python
def greedySearch(image):
    in_text = 'startseq'

    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = np.array(pad_sequences([sequence], maxlen=max_length))
        yhat = model.predict([image, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break

    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

image_path = './val2017/000000000785.jpg'

image = img_to_array(load_img(image_path, target_size=(299, 299)))
image = image / 255.

image = np.array([features_dict[581781]])
caption = greedySearch(image)

print("Caption = {}".format(caption))
```

برای تست شبکه، مسیر تصویر موردنظر (image_path) گرفته می شود و پس از پیش پردازش، با استفاده الگوریتم greedySearch کپشن تصویر به دست می آید.

در این الگوریتم، در ابتدا تصویر موردنظر و دنباله ای شامل واژه ’startseq‘ (این دنباله با استفاده از دیکشنری Wordtoix به دنباله ایندکس ها تبدیل می شود و در نهایت zero pad می شود) به شبکه داده می شود و کلمه بعدی کپشن پیش بینی می شود. سپس کلمه پیش بینی شده به دنباله قبلی اضافه می شود و پیش بینی دوباره انجام می شود. این عملیات به همین منوال ادامه پیدا می کند. دو شرط برای توقف عملیات وجود دارد:

۱- تعداد کلمات دنباله به طول حداکثر کپشن ها (47) برسد.

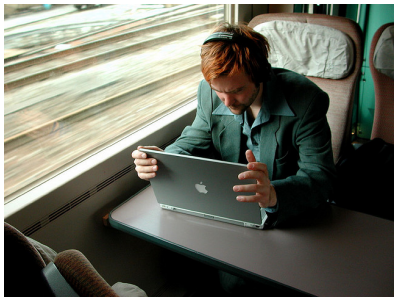
۲- کلمه پیش بینی شده، کلمه ’endseq‘ باشد.

در نهایت پس از حذف کلمات ’startseq‘ و ’endseq‘ کپشن خروجی آماده می شود.

بررسی عملکرد

در قسمت تعدادی تصویر به عنوان ورودی به الگوریتم greedySearch داده شده است و کپشن متناظر با آن تولید شده است:

Caption = man on skis on snowy surface



Caption = man in suit and tie sitting on top of laptop

Caption = an airplane is flying through the sky

Caption = man in white shirt playing tennis