Assignment_3_1

June 12, 2020

1 Assignment 3.1. Sequence Classification

2 Task: Aspect-level Sentiment Classification(10pt)

Group 82 * Student 1 : Joris Willems + 0908753 * Student 2 : Lars Schilders + 0908729

Reading material: - [1] R. He, WS. Lee & D. Dahlmeier. Exploiting document knowledge for aspect-level sentiment classification. 2018. https://arxiv.org/abs/1806.04346.

Build an attention-based aspect-level sentiment classification model with biLSTM. Your model shall include:

- BiLSTM network that learns sentence representation from input sequences.
- Attention network that assigns attention score over a sequence of biLSTM hidden states based on aspect terms representation.
- Fully connected network that predicts sentiment label, given the representation weighted by the attention score.

Requirements:

- You shall train your model bsaed on transferring learning. That is, you need first train your model on documnet-level examples. Then the learned weights will be used to initialize aspect-level model and fine tune it on aspect-level examples.
- You shall use the alignment score function in attention network as following expression:

$$f_{score}(h, t) = tanh(h^T W_a t)$$

- You shall evaluate the trained model on the provided test set and show the accuracy on test set.

[1]:
```
from google.colab import drive
drive.mount('/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
```

```
..........
Mounted at /drive
```

[1]:
```
import os
import sys
import codecs
import operator
import numpy as np
import re
from time import time
```

[2]:
```
import _pickle as cPickle
```

3 Load Data

[3]:
```
def read_pickle(data_path, file_name):

    f = open(os.path.join(data_path, file_name), 'rb')
    read_file = cPickle.load(f)
    f.close()

    return read_file

def save_pickle(data_path, file_name, data):

    f = open(os.path.join(data_path, file_name), 'wb')
    cPickle.dump(data, f)
    print(" file saved to: %s"%(os.path.join(data_path, file_name)))
    f.close()
```

[4]:
```
aspect_path = '/drive/My Drive/Colab Notebooks/2IMM10 - Deep Learning/
↪Assignment 3/aspect-level' #for lars
aspect_path = '/drive/My Drive/deeplearning/data/aspect-level/' #for joris
aspect_path = '../courseFiles/Practicals/Practical 5/data/aspect-level/' #for␣
↪local jupyter notebook

vocab = read_pickle(aspect_path, 'all_vocab.pkl')

train_x = read_pickle(aspect_path, 'train_x.pkl')
train_y = read_pickle(aspect_path, 'train_y.pkl')
dev_x = read_pickle(aspect_path, 'dev_x.pkl')
dev_y = read_pickle(aspect_path, 'dev_y.pkl')
test_x = read_pickle(aspect_path, 'test_x.pkl')
test_y = read_pickle(aspect_path, 'test_y.pkl')

train_aspect = read_pickle(aspect_path, 'train_aspect.pkl')
```

```
dev_aspect = read_pickle(aspect_path, 'dev_aspect.pkl')
test_aspect = read_pickle(aspect_path, 'test_aspect.pkl')


pretrain_data = read_pickle(aspect_path, 'pretrain_data.pkl')
pretrain_label = read_pickle(aspect_path, 'pretrain_label.pkl')
```

[5]:
```
def DecodeSequence(idx):
    review = str([list(vocab.keys())[i] for i in train_x[idx] if i!=0])
    aspectTerm = [list(vocab.keys())[i] for i in train_aspect[idx]]
    target = ["Positive", "Negative", "Neutral"][int(np.
 ↪nonzero(train_y[idx])[0])]
    print("Full review: {}".format(review))
    print("Aspect terms: {}".format(aspectTerm))
    print("Target: {}".format(target))
```

[6]:
```
class Dataiterator_doc():
    '''
      1) Iteration over minibatches using next(); call reset() between epochs␣
 ↪to randomly shuffle the data
      2) Access to the entire dataset using all()
    '''

    def __init__(self, X, y, seq_length=32, decoder_dim=300, batch_size=32):
        self.X = X
        self.y = y
        self.num_data = len(X) # total number of examples
        self.batch_size = batch_size # batch size
        self.reset() # initial: shuffling examples and set index to 0

    def __iter__(self): # iterates data
        return self


    def reset(self): # initials
        self.idx = 0
        self.order = np.random.permutation(self.num_data) # shuffling examples␣
 ↪by providing randomized ids

    def __next__(self): # return model inputs - outputs per batch
        X_ids = [] # hold ids per batch
        while len(X_ids) < self.batch_size:
            X_id = self.order[self.idx] # copy random id from initial shuffling
            X_ids.append(X_id)
            self.idx += 1 #
            if self.idx >= self.num_data: # exception if all examples of data␣
 ↪have been seen (iterated)
```

```
                self.reset()
                raise StopIteration()

        batch_X = self.X[np.array(X_ids)] # X values (encoder input) per batch
        batch_y = self.y[np.array(X_ids)] # y_in values (decoder input) per␣
↪batch
        return batch_X, batch_y


    def all(self): # return all data examples
        return self.X, self.y
class Dataiterator_aspect():
    '''
      1) Iteration over minibatches using next(); call reset() between epochs␣
↪to randomly shuffle the data
      2) Access to the entire dataset using all()
    '''

    def __init__(self, aspect_data, seq_length=32, decoder_dim=300,␣
↪batch_size=32):

        len_aspect_data = len(aspect_data[0])
        #self.len_doc_data = len(doc_data[0])

        self.X_aspect = aspect_data[0]
        self.y_aspect = aspect_data[1]
        self.aspect_terms = aspect_data[2]
        self.num_data = len_aspect_data
        self.batch_size = batch_size # batch size
        self.reset() # initial: shuffling examples and set index to 0

    def __iter__(self): # iterates data
        return self


    def reset(self): # initials
        self.idx = 0
        self.order = np.random.permutation(self.num_data) # shuffling examples␣
↪by providing randomized ids

    def __next__(self): # return model inputs - outputs per batch

        X_ids = [] # hold ids per batch
        while len(X_ids) < self.batch_size:
            X_id = self.order[self.idx] # copy random id from initial shuffling
            X_ids.append(X_id)
            self.idx += 1 #
```

```
            if self.idx >= self.num_data: # exception if all examples of data␣
↪have been seen (iterated)
                self.reset()
                raise StopIteration()

        batch_X_aspect = self.X_aspect[np.array(X_ids)] # X values (encoder␣
↪input) per batch
        batch_y_aspect = self.y_aspect[np.array(X_ids)] # y_in values (decoder␣
↪input) per batch
        batch_aspect_terms = self.aspect_terms[np.array(X_ids)]

        return batch_X_aspect, batch_y_aspect, batch_aspect_terms


    def all(self): # return all data examples
        return self.X_aspect, self.y_aspect, self.aspect_terms
```

[7]:
```
from tensorflow import keras
from keras.models import Model
from keras.layers import Input, Embedding, Dense, Lambda, Dropout,␣
↪LSTM,Bidirectional
from keras.layers import Reshape, Activation, RepeatVector, concatenate,␣
↪Concatenate, Dot, Multiply, Add
import keras.backend as K
from keras.engine.topology import Layer
from keras import initializers
from keras import regularizers
from keras import constraints
```

```
Using TensorFlow backend.
```

[8]:
```
overal_maxlen = 82
overal_maxlen_aspect = 7
```

4 Define Attention Network Layer

- Define class for Attention Layer
- You need to finish the code for calculating the attention weights

[9]:
```
class Attention(Layer):
    def __init__(self,  **kwargs):
        """
        Keras Layer that implements an Content Attention mechanism.
        Supports Masking.
        """
```

```
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert type(input_shape) == list

        self.steps = input_shape[0][1]

        self.W = self.add_weight(shape=(input_shape[0][-1], input_shape[1][-1]),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),)

        self.built = True

    def compute_mask(self, input_tensor, mask=None):
        assert type(input_tensor) == list
        assert type(mask) == list
        return None

    def call(self, input_tensor, mask=None):

        # output of BiLSTM for sentence (h in paper)
        x = input_tensor[0]  #(None, 82, 600)

        # output of word embedding layer of aspect terms (t in paper)
        aspect = input_tensor[1] #(None, 300)

        # used to remove influence of padded value
        #mask = mask[0]

        #(None, 600)
        aspect = K.transpose(K.dot(self.W, K.transpose(aspect)))

        #(None, 1, 600)
        aspect = K.expand_dims(aspect, axis=-2)

        #(None, 82, 600)
        aspect = K.repeat_elements(aspect, self.steps, axis=1)

        #(None, 82)
        beta_vec = K.tanh(K.sum(x*aspect, axis=-1))

        #(None, 82)
        alpha = K.exp(beta_vec)
```

```python
        alpha /= K.cast(K.sum(alpha, axis=1, keepdims=True) + K.epsilon(), K.
↪floatx())

        return alpha


    def compute_output_shape(self, input_shape):
        print((input_shape[0][0], input_shape[0][1]))
        return (input_shape[0][0], input_shape[0][1]) #(None, 82)


    ##### van paper ######
    def call2(self, input_tensor, mask=None):
        x = input_tensor[0]
        aspect = input_tensor[1]
        mask = mask[0]

        aspect = K.transpose(K.dot(self.W, K.transpose(aspect)))
        aspect = K.expand_dims(aspect, axis=-2)
        aspect = K.repeat_elements(aspect, self.steps, axis=1)
        eij = K.sum(x*aspect, axis=-1)

        if self.bias:
            b = K.repeat_elements(self.b, self.steps, axis=0)
            eij += b

        eij = K.tanh(eij)

        a = K.exp(eij)

        if mask is not None:
            a *= K.cast(mask, K.floatx())

        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())

        return a
```

[10]:
```python
class Average(Layer):

    def __init__(self, mask_zero=True, **kwargs):
        self.mask_zero = mask_zero
        self.supports_masking = True
        super(Average, self).__init__(**kwargs)

    def call(self, x,mask=None):
        if self.mask_zero:
            mask = K.cast(mask, K.floatx())
```

```
            mask = K.expand_dims(mask)
            x = x * mask
            return K.sum(x, axis=1) / (K.sum(mask, axis=1) + K.epsilon())
        else:
            return K.mean(x, axis=1)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[-1])

    def compute_mask(self, x, mask):
        return None
```

5 Establish computation Grah for model

- Input tensors
- Shared WordEmbedding layer
- Attention network layer

- Shared BiLSTM layer
- Shared fully connected layer(prediction layer)

[11]:
```
dropout = 0.5
recurrent_dropout = 0.1
vocab_size = len(vocab)
num_outputs = 3 # labels
```

5.1 Input tensors

[14]:
```
#YOUR CODE HERE ##### Inputs #####
sentence_input = Input(shape=(overal_maxlen,), dtype='int32',␣
 ↪name='sentence_input')
aspect_input = Input(shape=(overal_maxlen_aspect,), dtype='int32',␣
 ↪name='aspect_input')
pretrain_input = Input(shape=(None,), dtype='int32', name='pretrain_input')
```

5.2 Shared WordEmbedding layer

[15]:
```
#YOUR CODE HERE### represent aspect as averaged word embedding ###
word_emb = Embedding(vocab_size, 300, mask_zero=True, name='word_emb')
aspect_term_embs = word_emb(aspect_input)
aspect_embs = Average(mask_zero=True, name='aspect_emb')(aspect_term_embs)
```

[16]:
```
#YOUR CODE HERE ### sentence representation from embedding ###
sentence_embs = word_emb(sentence_input) # from aspect-level domain
pretrain_embs = word_emb(pretrain_input) # from document-level domain
```

5.3 Shared BiLSTM layer

[17]:
```
#YOUR CODE HERE ### sentence representation from embedding ###
bilstm = Bidirectional(LSTM(300, return_sequences=True, dropout=dropout,␣
 ↪recurrent_dropout=recurrent_dropout), name='BiLSTM')
pretrain_lstm = bilstm(pretrain_embs) #pretrain the bilstm (no attention␣
 ↪involved) MUST BE AVERAGED
sentence_lstm = bilstm(sentence_embs) #finetune bilstm with attention mechanism␣
 ↪to obtain aspect-level info
```

5.4 Attention Layer

[18]:
```
##YOUR CODE HERE
attention = Attention()([sentence_lstm, aspect_embs])
```

```
(None, 82)
```

5.5 Prediction Layer

[19]:
```
z = Dot(axes=1)([sentence_lstm, attention])

predictionLayer = Dense(3, activation='softmax')

aspect_probs = predictionLayer(z)

pretrain_avg = Average(mask_zero=True)(pretrain_lstm)
pretrain_probs = predictionLayer(pretrain_avg)
```

6 Build Models for document-level and aspect-level data

- The two models shared the embedding, BiLSTM, Prediction Layer

[22]:
```
### YOUR CODE HERE
model1 = Model(inputs=[pretrain_input], outputs=[pretrain_probs])

model2 = Model(inputs=[sentence_input, aspect_input], outputs=[aspect_probs])

model2.summary()
```

```
Model: "model_2"
____________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================
==================
sentence_input (InputLayer)     (None, 82)           0
____________________________________________________________________________
```

```
------------------
aspect_input (InputLayer)       (None, 7)            0

--------------------------------------------------------------------------------
------------------
word_emb (Embedding)            multiple             3000900
aspect_input[0][0]
sentence_input[0][0]

--------------------------------------------------------------------------------
------------------
BiLSTM (Bidirectional)          multiple             1442400     word_emb[1][0]

--------------------------------------------------------------------------------
------------------
aspect_emb (Average)            (None, 300)          0           word_emb[0][0]

--------------------------------------------------------------------------------
------------------
attention_1 (Attention)         (None, 82)           180000      BiLSTM[1][0]
aspect_emb[0][0]

--------------------------------------------------------------------------------
------------------
dot_1 (Dot)                     (None, 600)          0           BiLSTM[1][0]
attention_1[0][0]

--------------------------------------------------------------------------------
------------------
dense_1 (Dense)                 (None, 3)            1803        dot_1[0][0]
================================================================================
==================
Total params: 4,625,103
Trainable params: 4,625,103
Non-trainable params: 0

--------------------------------------------------------------------------------
------------------
```

7 Train Model

- First Train model on document-level data.
- Then Train model on aspect-level data

7.1 Train on document-level data

[ ]:
```
import keras.optimizers as opt
from sklearn.model_selection import train_test_split

# split the training dataset to be able to stop training when overfitting␣
↪starts to kick in
pretrain_data_train, pretrain_data_test, pretrain_label_train,␣
↪pretrain_label_test = train_test_split(pretrain_data,
```

```
                                                                                    ␣
↪                       pretrain_label,
                                                                                    ␣
↪                       test_size=0.05,
                                                                                    ␣
↪                       stratify=pretrain_label)

optimizer=opt.RMSprop(lr=0.0005, rho=0.9, epsilon=1e-06, clipnorm=10,␣
↪clipvalue=0)
model1.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['categorical_accuracy'])
batch_size = 256

train_steps_epoch = len(pretrain_data_train)/batch_size
batch_train_iter_doc = Dataiterator_doc(pretrain_data_train,␣
↪pretrain_label_train, batch_size=batch_size)

test_steps_epoch = len(pretrain_data_test)/batch_size
batch_test_iter_doc = Dataiterator_doc(pretrain_data_test, pretrain_label_test,␣
↪batch_size=batch_size)
```

[24]:

```
###YOUR CODE HERE###

from keras.callbacks import EarlyStopping, ModelCheckpoint

def train_gen_doc():
    while True:
        for batch in batch_train_iter_doc:
            yield batch

def test_gen_doc():
    while True:
        for batch in batch_test_iter_doc:
            yield batch

history1 = model1.fit_generator(train_gen_doc(),
                                steps_per_epoch=train_steps_epoch,
                                validation_data=test_gen_doc(),
                                validation_steps=test_steps_epoch,
                                epochs=20,
                                callbacks = [EarlyStopping(monitor='val_loss',␣
↪patience=2)]
                                )
# save model
model1.save(aspect_path + "../model1.h5")
```

/usr/local/lib/python3.6/dist-

```
packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may
consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Epoch 1/20
112/111 [==============================] - 262s 2s/step - loss: 0.9200 -
categorical_accuracy: 0.5683 - val_loss: 0.7362 - val_categorical_accuracy:
0.6628
Epoch 2/20
112/111 [==============================] - 263s 2s/step - loss: 0.7161 -
categorical_accuracy: 0.6853 - val_loss: 0.6919 - val_categorical_accuracy:
0.6914
Epoch 3/20
112/111 [==============================] - 264s 2s/step - loss: 0.6462 -
categorical_accuracy: 0.7210 - val_loss: 0.7817 - val_categorical_accuracy:
0.6934
Epoch 4/20
112/111 [==============================] - 263s 2s/step - loss: 0.6059 -
categorical_accuracy: 0.7405 - val_loss: 0.7327 - val_categorical_accuracy:
0.7031
```

7.2 Train on aspect-level data

[ ]:
```
batch_size = 128

train_steps_epoch = len(train_x)/batch_size
batch_train_iter_aspect = Dataiterator_aspect([train_x, train_y, train_aspect],␣
↪batch_size=batch_size)
val_steps_epoch = len(dev_x)/batch_size
batch_val_iter_aspect = Dataiterator_aspect([dev_x, dev_y, dev_aspect],␣
↪batch_size=batch_size)

import keras.optimizers as opt
optimizer = opt.Adam(lr=0.00005, beta_1=0.9, beta_2=0.999, epsilon=1e-08,␣
↪clipnorm=10, clipvalue=0)
model2.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['categorical_accuracy'])
```

[26]:
```
### YOUR CODE HERE

def train_gen_aspect():
    while True:
        train_batches = [[[X, aspect], [y]] for X, y, aspect in␣
↪batch_train_iter_aspect]

        for batch in train_batches:
```

```
            yield batch

def val_gen_aspect():
    while True:
        val_batches = [[[X, aspect], [y]] for X, y, aspect in␣
 ↪batch_val_iter_aspect]

        for batch in val_batches:
            yield batch

history2 = model2.fit_generator(train_gen_aspect(),
                     steps_per_epoch=train_steps_epoch,
                     epochs=25,
                     validation_data=val_gen_aspect(),
                     validation_steps=val_steps_epoch
                     )
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may
consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Epoch 1/25
15/14 [================================] - 12s 798ms/step - loss: 0.9739 -
categorical_accuracy: 0.5594 - val_loss: 1.0163 - val_categorical_accuracy:
0.5586
Epoch 2/25
15/14 [================================] - 10s 698ms/step - loss: 0.9114 -
categorical_accuracy: 0.5833 - val_loss: 0.8177 - val_categorical_accuracy:
0.6172
Epoch 3/25
15/14 [================================] - 11s 703ms/step - loss: 0.8659 -
categorical_accuracy: 0.6094 - val_loss: 0.8023 - val_categorical_accuracy:
0.5996
Epoch 4/25
15/14 [================================] - 10s 700ms/step - loss: 0.8493 -
categorical_accuracy: 0.6182 - val_loss: 0.9249 - val_categorical_accuracy:
0.6211
Epoch 5/25
15/14 [================================] - 10s 696ms/step - loss: 0.8144 -
categorical_accuracy: 0.6380 - val_loss: 0.8567 - val_categorical_accuracy:
0.6367
Epoch 6/25
15/14 [================================] - 10s 694ms/step - loss: 0.8110 -
categorical_accuracy: 0.6484 - val_loss: 0.9310 - val_categorical_accuracy:
0.6562
Epoch 7/25
```

```
15/14 [==============================] - 10s 694ms/step - loss: 0.7958 -
categorical_accuracy: 0.6505 - val_loss: 0.8238 - val_categorical_accuracy:
0.6406
Epoch 8/25
15/14 [==============================] - 10s 697ms/step - loss: 0.7703 -
categorical_accuracy: 0.6635 - val_loss: 0.8402 - val_categorical_accuracy:
0.6348
Epoch 9/25
15/14 [==============================] - 10s 700ms/step - loss: 0.7614 -
categorical_accuracy: 0.6687 - val_loss: 0.7398 - val_categorical_accuracy:
0.6543
Epoch 10/25
15/14 [==============================] - 10s 687ms/step - loss: 0.7421 -
categorical_accuracy: 0.6885 - val_loss: 0.7482 - val_categorical_accuracy:
0.6699
Epoch 11/25
15/14 [==============================] - 10s 695ms/step - loss: 0.7427 -
categorical_accuracy: 0.6703 - val_loss: 0.8124 - val_categorical_accuracy:
0.6387
Epoch 12/25
15/14 [==============================] - 10s 691ms/step - loss: 0.7210 -
categorical_accuracy: 0.6984 - val_loss: 0.8509 - val_categorical_accuracy:
0.6777
Epoch 13/25
15/14 [==============================] - 11s 720ms/step - loss: 0.7122 -
categorical_accuracy: 0.6938 - val_loss: 0.9618 - val_categorical_accuracy:
0.6582
Epoch 14/25
15/14 [==============================] - 11s 700ms/step - loss: 0.6920 -
categorical_accuracy: 0.7115 - val_loss: 0.8218 - val_categorical_accuracy:
0.6738
Epoch 15/25
15/14 [==============================] - 10s 695ms/step - loss: 0.6884 -
categorical_accuracy: 0.7141 - val_loss: 0.7906 - val_categorical_accuracy:
0.6777
Epoch 16/25
15/14 [==============================] - 10s 699ms/step - loss: 0.6769 -
categorical_accuracy: 0.7271 - val_loss: 0.7560 - val_categorical_accuracy:
0.6738
Epoch 17/25
15/14 [==============================] - 10s 692ms/step - loss: 0.6476 -
categorical_accuracy: 0.7432 - val_loss: 0.7991 - val_categorical_accuracy:
0.6719
Epoch 18/25
15/14 [==============================] - 11s 714ms/step - loss: 0.6561 -
categorical_accuracy: 0.7365 - val_loss: 0.7956 - val_categorical_accuracy:
0.6660
Epoch 19/25
```

```
15/14 [==============================] - 10s 697ms/step - loss: 0.6389 -
categorical_accuracy: 0.7427 - val_loss: 0.8070 - val_categorical_accuracy:
0.6641
Epoch 20/25
15/14 [==============================] - 10s 697ms/step - loss: 0.6126 -
categorical_accuracy: 0.7609 - val_loss: 0.9036 - val_categorical_accuracy:
0.6777
Epoch 21/25
15/14 [==============================] - 11s 705ms/step - loss: 0.6242 -
categorical_accuracy: 0.7484 - val_loss: 0.9776 - val_categorical_accuracy:
0.6602
Epoch 22/25
15/14 [==============================] - 11s 703ms/step - loss: 0.5987 -
categorical_accuracy: 0.7661 - val_loss: 0.8078 - val_categorical_accuracy:
0.6621
Epoch 23/25
15/14 [==============================] - 11s 701ms/step - loss: 0.5807 -
categorical_accuracy: 0.7635 - val_loss: 0.8075 - val_categorical_accuracy:
0.6621
Epoch 24/25
15/14 [==============================] - 10s 694ms/step - loss: 0.5899 -
categorical_accuracy: 0.7708 - val_loss: 0.7806 - val_categorical_accuracy:
0.6562
Epoch 25/25
15/14 [==============================] - 11s 704ms/step - loss: 0.5587 -
categorical_accuracy: 0.7703 - val_loss: 0.8241 - val_categorical_accuracy:
0.6582
```

7.3 Evaluating on test set

- show the accuracy

[ ]:
```
##YOUR CODE HERE
```

[27]:
```
loss, accuracy = model2.evaluate([test_x, test_aspect], test_y)

print("Categorical accuracy: {:.3f}%".format(accuracy*100))
```

```
638/638 [==============================] - 2s 3ms/step
Categorical accuracy: 59.718%
```

[ ]:
```

```