Sample Code

1. **Demo\_e.py**

**This demo illustrates one of my recent projects using CV and related machine learning method to extract the information from a video clip of a LOL game.**

import argparse

import cfg

from network import East

from predict import predict

from moviepy.editor import VideoFileClip

from tqdm import tqdm

import torch.utils.data

from torch.autograd import Variable

import os

from crnn import util

from crnn import dataset

from crnn.models import crnn

from crnn import keys

import cv2

import json

import pandas as pd

# This demo shows a recent project using CV to extract information from video clips of LOL and trained with pre-defined model that runs on cuda.

def parse\_args():

"""define the working dir"""

parser = argparse.ArgumentParser()

parser.add\_argument('--path', '-p',

default='',

help='image path')

parser.add\_argument('--threshold', '-t',

default=cfg.pixel\_threshold,

help='pixel activation threshold')

return parser.parse\_args()

def work(vnum):

zh = []

ocr = []

for fla in tqdm(range(vnum)):

try:

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, fla)

zh.append(fla)

ret, frame = cap.read()

image0 = predict(east\_detect, frame, threshold)

ocrstr = []

n = 0

for l, img in image0:

n += 1

image = img.convert('L')

scale = image.size[1] \* 1.0 / 32

w = image.size[0] / scale

w = int(w)

# print(w)

transformer = dataset.resizeNormalize((w, 32))

image = transformer(image).cuda()

image = image.view(1, \*image.size())

image = Variable(image)

model.eval()

preds = model(image)

\_, preds = preds.max(2)

preds = preds.squeeze(-2)

preds = preds.transpose(1, 0).contiguous().view(-1)

preds\_size = Variable(torch.IntTensor([preds.size(0)]))

raw\_pred = converter.decode(preds.data, preds\_size.data, raw=True)

sim\_pred = converter.decode(preds.data, preds\_size.data, raw=False)

if len(image0) == 0:

ocr.append([fla,0])

elif len(image0) == 1:

ocr.append([fla, all[l, sim\_pred]])

else:

ocrstr.append([l, sim\_pred])

ocr.append([fla, ocrstr])

# print(dict(zip(zh, ocr)))

if len(zh) == int(cap.get(7)):

break

# focr = dict(zip(zh, ocr))

except:

pass

continue

return [ocr]

if \_\_name\_\_ == '\_\_main\_\_':

"""use pretrained weights and RNN model to run CV on cuda environement """

args = parse\_args()

img\_path = args.path

threshold = float(args.threshold)

east = East()

east\_detect = east.east\_network()

east\_detect.load\_weights(cfg.saved\_model\_weights\_file\_path)

alphabet = keys.alphabet

converter = util.strLabelConverter(alphabet)

model = crnn.CRNN(32, 1, len(alphabet) + 1, 256, 1).cuda()

path = './crnn/samples/netCRNN63.pth'

model.load\_state\_dict(torch.load(path))

videopath = '/home/username/PycharmProjects/videos/'

finish = []

for file in os.listdir('./result'):

if file.endswith('.xlsx'):

finish.append(file[:-5] + '.mp4')

flag = 0

for file in os.listdir(videopath):

if file not in finish:

flag += 1

print('Processing the {} video...'.format(flag))

size = os.path.getsize(videopath + file)

if size == 0:

zong = dict([("Video\_Name", file[:-4]), ("Video\_Time", 0), ("Total\_Frames", 0),("Video\_Ocr", 0)])

else:

clip = VideoFileClip(videopath + file)

file\_time = clip.duration

cap = cv2.VideoCapture(videopath + file)

vnum = int(cap.get(7))

zong = dict([("Video\_Name", file[:-4]), ("Video\_Time", file\_time), ("Total\_Frames", vnum)])

ocr = work(vnum)

# focr = dict(zip(zh, ocr))

print("focr:", ocr)

zong["Video\_Ocr"] = ocr

df = pd.DataFrame(zong, index=[0])

cols = ['Video\_Name', 'Video\_Time', 'Total\_Frames', 'Video\_Ocr']

df = df.ix[:, cols]

df.to\_excel("./result/"+file[:-4]+".xlsx", index=False)

df.to\_hdf("./result/"+file[:-4]+".h5", "data")

print(df.head())

1. **Emojify.py**

**This is a project building a LSTM model to predict the emotion of a sentence and hence add a emoji.**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jan 31 17:01:19 2019

@author: Dequan

"""

import pandas as pd

import csv

from pandas import read\_csv

import numpy as np

# from emo\_utils import \*

import emoji

import matplotlib.pyplot as plt

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

def read\_glove\_vecs(glove\_file):

"""

read gloveVec file, initial encoding

"""

with open(glove\_file, 'r', encoding='UTF-8') as f:

words = set()

word\_to\_vec\_map = {}

for line in f:

line = line.strip().split()

curr\_word = line[0]

words.add(curr\_word)

word\_to\_vec\_map[curr\_word] = np.array(line[1:], dtype=np.float64)

i = 1

words\_to\_index = {}

index\_to\_words = {}

for w in sorted(words):

words\_to\_index[w] = i

index\_to\_words[i] = w

i = i + 1

return words\_to\_index, index\_to\_words, word\_to\_vec\_map

def softmax(x):

"""Compute softmax values for each sets of scores in x."""

e\_x = np.exp(x - np.max(x))

return e\_x / e\_x.sum()

def read\_csv(filename='data/emojify\_data.csv'):

phrase = []

emoji = []

with open(filename) as csvDataFile:

csvReader = csv.reader(csvDataFile)

for row in csvReader:

phrase.append(row[0])

emoji.append(row[1])

X = np.asarray(phrase)

Y = np.asarray(emoji, dtype=int)

return X, Y

def convert\_to\_one\_hot(Y, C):

Y = np.eye(C)[Y.reshape(-1)]

return Y

emoji\_dictionary = {"0": ":heart:", # :heart: prints a black instead of red heart depending on the font

"1": ":baseball:",

"2": ":smile:",

"3": ":disappointed:",

"4": ":fork\_and\_knife:"}

def label\_to\_emoji(label):

"""

Converts a label (int or string) into the corresponding emoji code (string)

ready to be printed

"""

return emoji.emojize(emoji\_dictionary[str(label)], use\_aliases=True)

def print\_predictions(X, pred):

print()

for i in range(X.shape[0]):

print(X[i], label\_to\_emoji(int(pred[i])))

def plot\_confusion\_matrix(y\_actu, y\_pred, title='Confusion matrix', cmap=plt.cm.gray\_r):

"""

plot the confusion matrix

"""

df\_confusion = pd.crosstab(y\_actu, y\_pred.reshape(y\_pred.shape[0],), rownames=['Actual'], colnames=['Predicted'], margins=True)

df\_conf\_norm = df\_confusion / df\_confusion.sum(axis=1)

plt.matshow(df\_confusion, cmap=cmap) # imshow

# plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(df\_confusion.columns))

plt.xticks(tick\_marks, df\_confusion.columns, rotation=45)

plt.yticks(tick\_marks, df\_confusion.index)

# plt.tight\_layout()

plt.ylabel(df\_confusion.index.name)

plt.xlabel(df\_confusion.columns.name)

def predict(X, Y, W, b, word\_to\_vec\_map):

"""

Given X (sentences) and Y (emoji indices), predict emojis and compute the

accuracy of your model over the given set.

Arguments:

X -- input data containing sentences, numpy array of shape (m, None)

Y -- labels, containing index of the label emoji, numpy array of shape (m, 1)

Returns:

pred -- numpy array of shape (m, 1) with your predictions

"""

m = X.shape[0]

pred = np.zeros((m, 1))

for j in range(m): # Loop over training examples

# Split jth test example (sentence) into list of lower case words

words = X[j].lower().split()

# Average words' vectors

avg = np.zeros((50,))

for w in words:

avg += word\_to\_vec\_map[w]

avg = avg/len(words)

# Forward propagation

Z = np.dot(W, avg) + b

A = softmax(Z)

pred[j] = np.argmax(A)

print("Accuracy: "+str(np.mean((pred[:] == Y.reshape(Y.shape[0], 1)[:]))))

return pred

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

X\_train, Y\_train = read\_csv('data/train\_emoji.csv')

X\_test, Y\_test = read\_csv('data/tesss.csv')

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

# test

index = 1

print('test==>', X\_train[index], label\_to\_emoji(Y\_train[index]))

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

maxLen = len(max(X\_train, key=len).split())

Y\_oh\_train = convert\_to\_one\_hot(Y\_train, C=5)

Y\_oh\_test = convert\_to\_one\_hot(Y\_test, C=5)

index = 50

print(Y\_train[index], "is converted into one hot", Y\_oh\_train[index])

word\_to\_index, index\_to\_word, word\_to\_vec\_map = read\_glove\_vecs('data/glove.6B.50d.txt')

# load world-to-vec map

def sentence\_to\_avg(sentence, word\_to\_vec\_map):

"""

Converts a sentence (string) into a list of words (strings). Extracts the

GloVe representation of each word and averages its value into a single vector

encoding the meaning of the sentence.

Arguments:

sentence -- string, one training example from X

word\_to\_vec\_map -- dictionary mapping every word in a vocabulary into its

50-dimensional vector representation

Returns:

avg -- average vector encoding information about the sentence, numpy-array of shape (50,)

"""

# Step 1: Split sentence into list of lower case words (≈ 1 line)

words = (sentence.lower()).split()

# Initialize the average word vector, should have the same shape as your word vectors.

avg = np.zeros((50,))

# Step 2: average the word vectors. You can loop over the words in the list "words".

for w in words:

avg += word\_to\_vec\_map[w]

avg = avg / len(words)

return avg

avg = sentence\_to\_avg("Morrocan couscous is my favorite dish", word\_to\_vec\_map)

print("avg = ", avg)

def model(X, Y, word\_to\_vec\_map, learning\_rate=0.01, num\_iterations=400):

"""

Model to train word vector representations in numpy.

Arguments:

X -- input data, numpy array of sentences as strings, of shape (m, 1)

Y -- labels, numpy array of integers between 0 and 7, numpy-array of shape (m, 1)

word\_to\_vec\_map -- dictionary mapping every word in a vocabulary into its

50-dimensional vector representation learning\_rate -- learning\_rate for the

stochastic gradient descent algorithm num\_iterations -- number of iterations

Returns:

pred -- vector of predictions, numpy-array of shape (m, 1)

W -- weight matrix of the softmax layer, of shape (n\_y, n\_h)

b -- bias of the softmax layer, of shape (n\_y,)

"""

np.random.seed(1)

# Define number of training examples

m = Y.shape[0] # number of training examples

n\_y = 5 # number of classes

n\_h = 50 # dimensions of the GloVe vectors

# Initialize parameters using Xavier initialization

W = np.random.randn(n\_y, n\_h) / np.sqrt(n\_h)

b = np.zeros((n\_y,))

# Convert Y to Y\_onehot with n\_y classes

Y\_oh = convert\_to\_one\_hot(Y, C=n\_y)

# Optimization loop

for t in range(num\_iterations): # Loop over the number of iterations

for i in range(m): # Loop over the training examples

# Average the word vectors of the words from the i'th training example

avg = sentence\_to\_avg(X[i], word\_to\_vec\_map)

# Forward propagate the avg through the softmax layer

z = np.dot(W, avg) + b

a = softmax(z)

# Compute cost using the i'th training label's one hot

# representation and "A" (the output of the softmax)

cost = - np.sum(Y\_oh[i] \* np.log(a))

# Compute gradients

dz = a - Y\_oh[i]

dW = np.dot(dz.reshape(n\_y, 1), avg.reshape(1, n\_h))

db = dz

# Update parameters with Stochastic Gradient Descent

W = W - learning\_rate \* dW

b = b - learning\_rate \* db

if t % 100 == 0:

print("Epoch: " + str(t) + " --- cost = " + str(cost))

pred = predict(X, Y, W, b, word\_to\_vec\_map)

return pred, W, b

print('X\_train.shape', X\_train.shape)

print('Y\_train.shape', Y\_train.shape)

print(np.eye(5)[Y\_train.reshape(-1)].shape)

print(X\_train[0])

print(type(X\_train))

Y = np.asarray([5, 0, 0, 5, 4, 4, 4, 6, 6, 4, 1, 1, 5, 6, 6, 3, 6, 3, 4, 4])

print(Y.shape)

X = np.asarray(['I am going to the bar tonight', 'I love you', 'miss you my dear',

'Lets go party and drinks','Congrats on the new job','Congratulations',

'I am so happy for you', 'Why are you feeling bad', 'What is wrong with you',

'You totally deserve this prize', 'Let us go play football',

'Are you down for football this afternoon', 'Work hard play harder',

'It is suprising how people can be dumb sometimes',

'I am very disappointed','It is the best day in my life',

'I think I will end up alone','My life is so boring','Good job',

'Great so awesome'])

print(X.shape)

print(np.eye(5)[Y\_train.reshape(-1)].shape)

print(type(X\_train))

pred, W, b = model(X\_train, Y\_train, word\_to\_vec\_map)

print(pred)

print("Training set:")

pred\_train = predict(X\_train, Y\_train, W, b, word\_to\_vec\_map)

print('Test set:')

pred\_test = predict(X\_test, Y\_test, W, b, word\_to\_vec\_map)

X\_my\_sentences = np.array(["i adore you", "i love you", "funny lol",

"lets play with a ball", "food is ready", "not feeling happy"])

Y\_my\_labels = np.array([[0], [0], [2], [1], [4], [3]])

pred = predict(X\_my\_sentences, Y\_my\_labels, W, b, word\_to\_vec\_map)

print\_predictions(X\_my\_sentences, pred)

print(Y\_test.shape)

print(' '+ label\_to\_emoji(0)+ ' ' + label\_to\_emoji(1) + ' ' + label\_to\_emoji(2)+ ' ' + label\_to\_emoji(3)+' ' + label\_to\_emoji(4))

print(pd.crosstab(Y\_test, pred\_test.reshape(56,), rownames=['Actual'], colnames=['Predicted'], margins=True))

plot\_confusion\_matrix(Y\_test, pred\_test)

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

# Emojify V2 this version add keras training method using LSTM

import numpy as np

np.random.seed(0)

from keras.models import Model

from keras.layers import Dense, Input, Dropout, LSTM, Activation

from keras.layers.embeddings import Embedding

from keras.preprocessing import sequence

from keras.initializers import glorot\_uniform

np.random.seed(1)

def sentences\_to\_indices(X, word\_to\_index, max\_len):

"""

Converts an array of sentences (strings) into an array of indices corresponding to words in the sentences.

The output shape should be such that it can be given to `Embedding()` (described in Figure 4).

Arguments:

X -- array of sentences (strings), of shape (m, 1)

word\_to\_index -- a dictionary containing the each word mapped to its index

max\_len -- maximum number of words in a sentence. You can assume every sentence in X is no longer than this.

Returns:

X\_indices -- array of indices corresponding to words in the sentences from X, of shape (m, max\_len)

"""

m = X.shape[0] # number of training examples

# Initialize X\_indices as a numpy matrix of zeros and the correct shape (≈ 1 line)

X\_indices = np.zeros((m, max\_len))

for i in range(m): # loop over training examples

# Convert the ith training sentence in lower case and split is into words.

# You should get a list of words.

sentence\_words = X[i].lower().split()

# Initialize j to 0

j = 0

# Loop over the words of sentence\_words

for w in sentence\_words:

# Set the (i,j)th entry of X\_indices to the index of the correct word.

X\_indices[i, j] = word\_to\_index[w]

# Increment j to j + 1

j = j + 1

return X\_indices

def pretrained\_embedding\_layer(word\_to\_vec\_map, word\_to\_index):

"""

Creates a Keras Embedding() layer and loads in pre-trained GloVe 50-dimensional vectors.

Arguments:

word\_to\_vec\_map -- dictionary mapping words to their GloVe vector representation.

word\_to\_index -- dictionary mapping from words to their indices in the vocabulary (400,001 words)

Returns:

embedding\_layer -- pretrained layer Keras instance

"""

vocab\_len = len(word\_to\_index) + 1 # adding 1 to fit Keras embedding (requirement)

emb\_dim = word\_to\_vec\_map["cucumber"].shape[0] # define dimensionality of your GloVe word vectors (= 50)

# Initialize the embedding matrix as a numpy array of zeros of shape

# (vocab\_len, dimensions of word vectors = emb\_dim)

emb\_matrix = np.zeros((vocab\_len, emb\_dim))

# Set each row "index" of the embedding matrix to be the word vector

# representation of the "index"th word of the vocabulary

for word, index in word\_to\_index.items():

emb\_matrix[index, :] = word\_to\_vec\_map[word]

# Define Keras embedding layer with the correct output/input sizes, make it trainable.

# Use Embedding(...). Set trainable=False.

embedding\_layer = Embedding(vocab\_len, emb\_dim)

# Build the embedding layer, it is required before setting the weights of

# the embedding layer.

embedding\_layer.build((None,))

# Set the weights of the embedding layer to the embedding matrix. Your layer is now pretrained.

embedding\_layer.set\_weights([emb\_matrix])

return embedding\_layer

embedding\_layer = pretrained\_embedding\_layer(word\_to\_vec\_map, word\_to\_index)

print("weights[0][1][3] =", embedding\_layer.get\_weights()[0][1][3])

def Emojify\_V2(input\_shape, word\_to\_vec\_map, word\_to\_index):

"""

Function creating the Emojify-v2 model's graph.

Arguments:

input\_shape -- shape of the input, usually (max\_len,)

word\_to\_vec\_map -- dictionary mapping every word in a vocabulary into its

50-dimensional vector representation word\_to\_index -- dictionary mapping from

words to their indices in the vocabulary (400,001 words)

Returns:

model -- a model instance in Keras

"""

# Define sentence\_indices as the input of the graph,

# it should be of shape input\_shape and dtype 'int32' (as it contains indices).

sentence\_indices = Input(shape=input\_shape, dtype=np.int32)

# Create the embedding layer pretrained with GloVe Vectors (≈1 line)

embedding\_layer = pretrained\_embedding\_layer(word\_to\_vec\_map, word\_to\_index)

# Propagate sentence\_indices through your embedding layer, you get back the embeddings

embeddings = embedding\_layer(sentence\_indices)

# Propagate the embeddings through an LSTM layer with 128-dimensional hidden state

# Be careful, the returned output should be a batch of sequences.

X = LSTM(128, return\_sequences=True)(embeddings)

# Add dropout with a probability of 0.5

X = Dropout(0.5)(X)

# Propagate X trough another LSTM layer with 128-dimensional hidden state

# Be careful, the returned output should be a single hidden state, not a batch of sequences.

X = LSTM(128)(X)

# Add dropout with a probability of 0.5

X = Dropout(0.5)(X)

# Propagate X through a Dense layer with softmax activation to get back a batch of 5-dimensional vectors.

X = Dense(5, activation='softmax')(X)

# Add a softmax activation

X = Activation('softmax')(X)

# Create Model instance which converts sentence\_indices into X.

model = Model(sentence\_indices, X)

return model

model = Emojify\_V2((maxLen,), word\_to\_vec\_map, word\_to\_index)

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

X\_train\_indices = sentences\_to\_indices(X\_train, word\_to\_index, maxLen)

Y\_train\_oh = convert\_to\_one\_hot(Y\_train, C=5)

model.fit(X\_train\_indices, Y\_train\_oh, epochs=50, batch\_size=32, shuffle=True)

X\_test\_indices = sentences\_to\_indices(X\_test, word\_to\_index, max\_len=maxLen)

Y\_test\_oh = convert\_to\_one\_hot(Y\_test, C=5)

loss, acc = model.evaluate(X\_test\_indices, Y\_test\_oh)

print()

print("Test accuracy = ", acc)

# Visulization of the mislabelled examples

C = 5

y\_test\_oh = np.eye(C)[Y\_test.reshape(-1)]

X\_test\_indices = sentences\_to\_indices(X\_test, word\_to\_index, maxLen)

pred = model.predict(X\_test\_indices)

for i in range(len(X\_test)):

x = X\_test\_indices

num = np.argmax(pred[i])

if(num != Y\_test[i]):

print('Expected emoji:'+label\_to\_emoji(Y\_test[i]) +' prediction: '+X\_test[i] + label\_to\_emoji(num).strip())

# Change the sentence below to see your prediction. Make sure all the words are in the Glove embeddings.

x\_test = np.array(['not feeling happy'])

X\_test\_indices = sentences\_to\_indices(x\_test, word\_to\_index, maxLen)

print(x\_test[0] + ' '+label\_to\_emoji(np.argmax(model.predict(X\_test\_indices))))