#coding=utf-8

# show GPU info

from tensorflow.python.client import device\_lib

print(device\_lib.list\_local\_devices())

#Import modules

import cv2

import pandas as pd

import numpy as np

import os

from numpy import asarray

from tqdm import tqdm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, f1\_score, recall\_score, hamming\_loss, zero\_one\_loss

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input,Dense, Dropout, Activation, Flatten, Conv2D, BatchNormalization, Reshape, MaxPooling2D, CuDNNLSTM, Embedding, Concatenate

import keras

from keras.utils import to\_categorical

from keras.preprocessing.image import ImageDataGenerator

from keras import callbacks

#import data

import pickle

X = pickle.load(open("X\_500.pickle", "rb")) #import preprocessed 500\*500 data

Y = pickle.load(open("Y.pickle", "rb"))

X = X/255

width, height, channels = X.shape[1:]

Ycate = [np.argmax(i) for i in Y]

from sklearn.utils import class\_weight

class\_weights = class\_weight.compute\_class\_weight('balanced',np.unique(Ycate),Ycate)

data = pd.read\_csv("data\_processed.csv")

# prepare text data

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

np.random.RandomState(0)

t = Tokenizer()

X\_text = data['corrected\_text']

X\_text = [str(i) for i in X\_text]

t.fit\_on\_texts(X\_text)

vocab\_size = len(t.word\_index) + 1

encoded\_train = t.texts\_to\_sequences(X\_text)

max\_length = max(list(map(lambda x: len(x), encoded\_train)))

padded\_train = pad\_sequences(encoded\_train, maxlen=max\_length, padding='post')

print("Image data shape:",X.shape)

print("Text data length:",padded\_train.shape)

print("Y shape:",Y.shape)

#Build the text+img model

a = input\_text = Input(shape = (max\_length,))

a = Embedding(vocab\_size, 100)(a)

a = CuDNNLSTM(128, return\_sequences=True)(a)

a = CuDNNLSTM(128)(a)

a = Flatten()(a)

a = Dense(128, activation='relu')(a)

b = input\_img = Input(shape = (width, height, channels))

b = Conv2D(32, (3, 3), activation='relu')(b)

b = MaxPooling2D((2,2))(b)

b = Conv2D(64, (3, 3), activation='relu')(b)

b = MaxPooling2D((2,2))(b)

b = Flatten()(b)

b = Dense(128, activation='relu')(b)

combine = Concatenate(axis=-1,)([a,b])

# combine = Dense(64, activation='relu')(combine)

# combine = Dropout(0.2)(combine)

# combine = Dense(32, activation='relu')(combine)

# combine = Dropout(0.2)(combine)

output = Dense(5, activation='softmax')(combine)

Model\_text\_img\_clr = Model([input\_text,input\_img],output)

Model\_text\_img\_clr.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

Model\_text\_img\_clr.summary()

results = Model\_text\_img\_clr.fit([padded\_train,X], Y,

epochs=20,

batch\_size=64,

shuffle=True,

validation\_split=0.2,

class\_weight=class\_weights

)

plt.plot(list(results.history.values())[0],'b',label='Train-Loss')

plt.plot(list(results.history.values())[1],'g',label='Train-Accuracy')

plt.legend(loc='upper center', shadow=True)

plt.plot(list(results.history.values())[2],'b',label='Test-Loss')

plt.plot(list(results.history.values())[3],'g',label='Test-Accuracy')

plt.legend(loc='upper center', shadow=True)