Convolutional Recurrent Neural Network for Optical Character Recognition

Importing the packages

```
!pip install craft-text-detector
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting craft-text-detector
  Downloading craft_text_detector-0.4.3-py3-none-any.whl (18 kB)
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: torchvision>=0.7.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: gdown>=3.10.1 in /usr/local/lib/python3.7/dist-packages (
Collecting opencv-python<4.5.4.62,>=3.4.8.29
  Downloading opencv_python-4.5.4.60-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_
     |████████████████████████████████| 60.3 MB 1.2 MB/s
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from gdown
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from gdo
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.7/dist-
Installing collected packages: opencv-python, craft-text-detector
  Attempting uninstall: opencv-python
    Found existing installation: opencv-python 4.6.0.66
    Uninstalling opencv-python-4.6.0.66:
      Successfully uninstalled opencv-python-4.6.0.66
Successfully installed craft-text-detector-0.4.3 opencv-python-4.5.4.60
```

```
!pip install -q transformers
```

```
     |████████████████████████████████| 5.5 MB 4.9 MB/s
     |████████████████████████████████| 182 kB 75.0 MB/s
     |████████████████████████████████| 7.6 MB 54.9 MB/s
```

```
import os
import fnmatch
import cv2
import numpy as np
import string
import time

from tensorflow.keras.preprocessing.sequence import pad_sequences

from keras.layers import Dense, LSTM, Reshape, BatchNormalization, Input, Conv2D, MaxPool2D,
from keras.models import Model
from keras.activations import relu, sigmoid, softmax
import keras.backend as K
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint
```

Dataset

You can directly download the full dataset in this link. Since it is a huge dataset, we have used only the subset of the original dataset. The subset dataset is available in this this link

Preprocessing

```
from google.colab import drive
drive.mount('/content/drive')

     Mounted at /content/drive
```

```
!unzip /content/drive/MyDrive/90kDICT32px.zip -d /content/90kDICT32px/

       inflating: /content/90kDICT32px/8/3/476_Groovy_33853.jpg
       inflating: /content/90kDICT32px/8/3/477_AMMETER_2506.jpg
       inflating: /content/90kDICT32px/8/3/478_obviate_52949.jpg
       inflating: /content/90kDICT32px/8/3/479_fuming_31237.jpg
       inflating: /content/90kDICT32px/8/3/47_SOMETHINGS_72702.jpg
       inflating: /content/90kDICT32px/8/3/480_Tagalogs_77217.jpg
       inflating: /content/90kDICT32px/8/3/481_Locomotive_45035.jpg
       inflating: /content/90kDICT32px/8/3/482_Harshness_35101.jpg
       inflating: /content/90kDICT32px/8/3/483_campaigned_11046.jpg
       inflating: /content/90kDICT32px/8/3/484_Mastitis_47063.jpg
       inflating: /content/90kDICT32px/8/3/485_Mousses_50103.jpg
       inflating: /content/90kDICT32px/8/3/486_ORIGIN_53749.jpg
       inflating: /content/90kDICT32px/8/3/487_DOMINGO_23206.jpg
       inflating: /content/90kDICT32px/8/3/488_buncos_10218.jpg
       inflating: /content/90kDICT32px/8/3/489_CONFINED_15932.jpg
       inflating: /content/90kDICT32px/8/3/48_clasped_13960.jpg
       inflating: /content/90kDICT32px/8/3/490_Carlton_11610.jpg
       inflating: /content/90kDICT32px/8/3/491_TUNDRA_81130.jpg
       inflating: /content/90kDICT32px/8/3/492_disqualify_22597.jpg
       inflating: /content/90kDICT32px/8/3/493_Snood_72263.jpg
```

```
  inflating: /content/90kDICT32px/8/3/494_Dears_19500.jpg  
  inflating: /content/90kDICT32px/8/3/495_genes_32109.jpg  
  inflating: /content/90kDICT32px/8/3/496_EPISODIC_26227.jpg  
  inflating: /content/90kDICT32px/8/3/497_DIXIE_22956.jpg  
  inflating: /content/90kDICT32px/8/3/498_PAIR_55053.jpg  
  inflating: /content/90kDICT32px/8/3/499_CAVIAR_12186.jpg  
  inflating: /content/90kDICT32px/8/3/49_PIGS_57569.jpg  
  inflating: /content/90kDICT32px/8/3/4_DECIDE_19708.jpg  
  inflating: /content/90kDICT32px/8/3/500_loyalty_45458.jpg  
  inflating: /content/90kDICT32px/8/3/501_Bonniest_8687.jpg  
  inflating: /content/90kDICT32px/8/3/50_underpinning_82190.jpg  
  inflating: /content/90kDICT32px/8/3/51_Mtg_50167.jpg  
  inflating: /content/90kDICT32px/8/3/52_Tupungato_81170.jpg  
  inflating: /content/90kDICT32px/8/3/53_Subscribed_75662.jpg  
  inflating: /content/90kDICT32px/8/3/54_repairers_64732.jpg  
  inflating: /content/90kDICT32px/8/3/55_Grasses_33493.jpg  
  inflating: /content/90kDICT32px/8/3/56_Consolidators_16308.jpg  
  inflating: /content/90kDICT32px/8/3/57_garrets_31796.jpg  
  inflating: /content/90kDICT32px/8/3/58_Stockyards_74835.jpg  
  inflating: /content/90kDICT32px/8/3/59_Reflate_63824.jpg  
  inflating: /content/90kDICT32px/8/3/5_asses_4386.jpg  
  inflating: /content/90kDICT32px/8/3/60_FOSTERING_30487.jpg  
  inflating: /content/90kDICT32px/8/3/61_MUDDLED_50199.jpg  
  inflating: /content/90kDICT32px/8/3/62_HOUSEMASTER_37078.jpg  
  inflating: /content/90kDICT32px/8/3/63_DEMOLISHING_20485.jpg  
  inflating: /content/90kDICT32px/8/3/64_Rebuffs_62996.jpg  
  inflating: /content/90kDICT32px/8/3/65_Contraindicating_16616.jpg  
  inflating: /content/90kDICT32px/8/3/66_sailed_67401.jpg  
  inflating: /content/90kDICT32px/8/3/67_Duplicity_24182.jpg  
  inflating: /content/90kDICT32px/8/3/68_Remolds_64547.jpg  
  inflating: /content/90kDICT32px/8/3/69_GREBE_33604.jpg  
  inflating: /content/90kDICT32px/8/3/6_Gustatory_34312.jpg  
  inflating: /content/90kDICT32px/8/3/70_Jeeringly_41357.jpg  
  inflating: /content/90kDICT32px/8/3/71_RELIGIOSITY_64399.jpg  
  inflating: /content/90kDICT32px/8/3/72_Gruffer_33997.jpg  
  inflating: /content/90kDICT32px/8/3/73_tattler_77676.jpg  
  inflating: /content/90kDICT32px/8/3/74_STIPPLE_74769.jpg  
  inflating: /content/90kDICT32px/8/3/75_telescopes_78004.jpg  
```

```
characterset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
# This function converts the characters in the text to numberical ids
def text2ids(txt):
    encoded_list = []
    for index, char in enumerate(txt):
        try:
            encoded_list.append(characterset.index(char))
        except:
            print(char)
    return encoded_list


# This function resizes the image with (32, 128, 1)
def rescale_img(img):
    # convert each image of shape (32, 128, 1)
    w, h = img.shape
    if h > 128 or w > 32:
        return "SKIP"
    if w < 32:
        add_zeros = np.ones((32-w, h))*255
        img = np.concatenate((img, add_zeros))

    if h < 128:
        add_zeros = np.ones((32, 128-h))*255
        img = np.concatenate((img, add_zeros), axis=1)
    img = np.expand_dims(img , axis = 2)

    # Normalize each image
    img = img/255.
    return img
```

```
images_training = []
text_training = []
len_train_input = []
len_training_label = []
original_txt_training = []
```

```
images_testing = []
text_testing = []
len_testing_input = []
len_testing_label = []
original_txt_testing = []
```

```
path = '/content/90kDICT32px'
```

Data Preparation

```
max_label_len = 0

i=0
for root, dirnames, filenames in os.walk(path):

    for name in fnmatch.filter(filenames, '*.jpg'):
        # read input image and convert into gray scale image
        img = cv2.cvtColor(cv2.imread(os.path.join(root, name)), cv2.COLOR_BGR2GRAY)

        img = rescale_img(img)
        if img == "SKIP":
            continue
        # get the text from the image
        txt = name.split('_')[1]
        if len(txt) > max_label_len:
            max_label_len = len(txt)


        if i%10 == 0:
            original_txt_testing.append(txt)
            len_testing_label.append(len(txt))
            len_testing_input.append(31)
            images_testing.append(img)
            text_testing.append(text2ids(txt))
        else:
            original_txt_training.append(txt)
            len_training_label.append(len(txt))
            len_train_input.append(31)
            images_training.append(img)
            text_training.append(text2ids(txt))
        i=i+1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: FutureWarning: elementw
  # This is added back by InteractiveShellApp.init_path()
```

```
train_padded_txt = pad_sequences(text_training, maxlen=max_label_len, padding='post', value =
valid_padded_txt = pad_sequences(text_testing, maxlen=max_label_len, padding='post', value =
```

Model Architecture

```
inputs = Input(shape=(32,128,1))
conv_1 = Conv2D(64, (3,3), activation = 'relu', padding='same')(inputs)
pool_1 = MaxPool2D(pool_size=(2, 2), strides=2)(conv_1)
conv_2 = Conv2D(128, (3,3), activation = 'relu', padding='same')(pool_1)
pool_2 = MaxPool2D(pool_size=(2, 2), strides=2)(conv_2)
conv_3 = Conv2D(256, (3,3), activation = 'relu', padding='same')(pool_2)
conv_4 = Conv2D(256, (3,3), activation = 'relu', padding='same')(conv_3)
pool_4 = MaxPool2D(pool_size=(2, 1))(conv_4)
conv_5 = Conv2D(512, (3,3), activation = 'relu', padding='same')(pool_4)
batch_norm_5 = BatchNormalization()(conv_5)
conv_6 = Conv2D(512, (3,3), activation = 'relu', padding='same')(batch_norm_5)
batch_norm_6 = BatchNormalization()(conv_6)
pool_6 = MaxPool2D(pool_size=(2, 1))(batch_norm_6)
conv_7 = Conv2D(512, (2,2), activation = 'relu')(pool_6)
squeezed = Lambda(lambda x: K.squeeze(x, 1))(conv_7)
blstm_1 = Bidirectional(LSTM(128, return_sequences=True, dropout = 0.2))(squeezed)
blstm_2 = Bidirectional(LSTM(128, return_sequences=True, dropout = 0.2))(blstm_1)
outputs = Dense(len(characterset)+1, activation = 'softmax')(blstm_2)

actual_model = Model(inputs, outputs)
```

```
actual_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 128, 1)]      0

 conv2d (Conv2D)             (None, 32, 128, 64)       640

 max_pooling2d (MaxPooling2D  (None, 16, 64, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 64, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 8, 32, 128)       0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 32, 256)        295168

 conv2d_3 (Conv2D)           (None, 8, 32, 256)        590080

 max_pooling2d_2 (MaxPooling  (None, 4, 32, 256)       0
 2D)

 conv2d_4 (Conv2D)           (None, 4, 32, 512)        1180160

 batch_normalization (BatchN  (None, 4, 32, 512)       2048
 ormalization)

 conv2d_5 (Conv2D)           (None, 4, 32, 512)        2359808

 batch_normalization_1 (Batc  (None, 4, 32, 512)       2048
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 2, 32, 512)       0
 2D)

 conv2d_6 (Conv2D)           (None, 1, 31, 512)        1049088

 lambda (Lambda)             (None, 31, 512)           0

 bidirectional (Bidirectiona  (None, 31, 256)          656384
 l)

 bidirectional_1 (Bidirectio  (None, 31, 256)          394240
 nal)

 dense (Dense)               (None, 31, 64)            16448

=================================================================
Total params: 6,619,968
Trainable params: 6,617,920
```

```
Non-trainable params: 2,048
```

CTC Loss Function (Spectial loss function for OCR problem)

[ ] ↳ *1 cell hidden*

Model Training

```
model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer = 'adam')

filepath="/content/drive/MyDrive/best_model.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only
callbacks_list = [checkpoint]


images_training = np.array(images_training)
len_train_input = np.array(len_train_input)
len_training_label = np.array(len_training_label)

images_testing = np.array(images_testing)
len_testing_input = np.array(len_testing_input)
len_testing_label = np.array(len_testing_label)


batch_size = 64
epochs = 10
model.fit(x=[images_training, train_padded_txt, len_train_input, len_training_label], y=np.ze
```

```
Epoch 1/10
1459/1459 [==============================] - ETA: 0s - loss: 27.0643
Epoch 1: val_loss improved from inf to 26.98736, saving model to /content/drive/MyDrive/
1459/1459 [==============================] - 142s 86ms/step - loss: 27.0643 - val_loss:
Epoch 2/10
1459/1459 [==============================] - ETA: 0s - loss: 25.8035
Epoch 2: val_loss improved from 26.98736 to 24.95718, saving model to /content/drive/MyD
1459/1459 [==============================] - 128s 87ms/step - loss: 25.8035 - val_loss:
Epoch 3/10
1459/1459 [==============================] - ETA: 0s - loss: 22.3205
Epoch 3: val_loss improved from 24.95718 to 19.34612, saving model to /content/drive/MyD
1459/1459 [==============================] - 121s 83ms/step - loss: 22.3205 - val_loss:
Epoch 4/10
1459/1459 [==============================] - ETA: 0s - loss: 13.3495
Epoch 4: val_loss improved from 19.34612 to 9.14548, saving model to /content/drive/MyDr
1459/1459 [==============================] - 123s 84ms/step - loss: 13.3495 - val_loss:
Epoch 5/10
1459/1459 [==============================] - ETA: 0s - loss: 7.2285
Epoch 5: val_loss improved from 9.14548 to 6.40544, saving model to /content/drive/MyDri
1459/1459 [==============================] - 122s 84ms/step - loss: 7.2285 - val_loss: 6
Epoch 6/10
1459/1459 [==============================] - ETA: 0s - loss: 5.6500
```

```
Epoch 6: val_loss improved from 6.40544 to 5.46001, saving model to /content/drive/MyDri
1459/1459 [==============================] - 123s 84ms/step - loss: 5.6500 - val_loss: 5
Epoch 7/10
1459/1459 [==============================] - ETA: 0s - loss: 4.8698
Epoch 7: val_loss improved from 5.46001 to 4.81709, saving model to /content/drive/MyDri
1459/1459 [==============================] - 121s 83ms/step - loss: 4.8698 - val_loss: 4
Epoch 8/10
1459/1459 [==============================] - ETA: 0s - loss: 4.3819
Epoch 8: val_loss improved from 4.81709 to 4.61188, saving model to /content/drive/MyDri
1459/1459 [==============================] - 123s 84ms/step - loss: 4.3819 - val_loss: 4
Epoch 9/10
1459/1459 [==============================] - ETA: 0s - loss: 4.0279
Epoch 9: val_loss improved from 4.61188 to 4.51954, saving model to /content/drive/MyDri
1459/1459 [==============================] - 121s 83ms/step - loss: 4.0279 - val_loss: 4
Epoch 10/10
1459/1459 [==============================] - ETA: 0s - loss: 3.8390
Epoch 10: val_loss improved from 4.51954 to 4.03375, saving model to /content/drive/MyDr
1459/1459 [==============================] - 122s 84ms/step - loss: 3.8390 - val_loss: 4
<keras.callbacks.History at 0x7f35a01ad8d0>
```

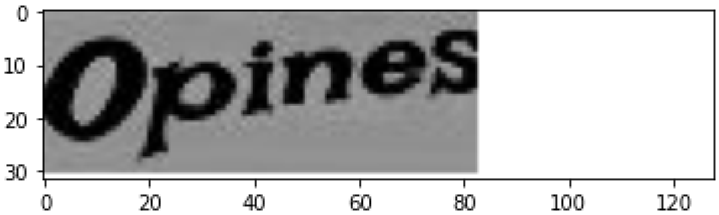
Predictions

```
# load the saved best model weights
actual_model.load_weights('/content/drive/MyDrive/best_model_10_epochs.hdf5')

# predict outputs on validation images
predicted = actual_model.predict(images_testing[50:60])

# use CTC decoder
output = K.get_value(K.ctc_decode(predicted, input_length=np.ones(predicted.shape[0])*predict
                         greedy=True)[0][0])
```

```
1/1 [==============================] - 0s 35ms/step
```

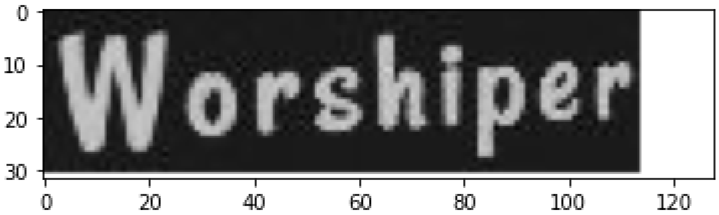
```
import matplotlib.pyplot as plt

i = 50
for word in output:
    print("-----------------Input Image-------------------")
    plt.imshow(images_testing[i].reshape(32,128), cmap='gray')
    plt.show()
    print("predicted text = ", end = '')
    for char in word:
        if int(char) != -1:
            print(characterset[int(char)], end = '')
    print('\n')
    i+=1
```

```
-----------------Input Image-------------------
```



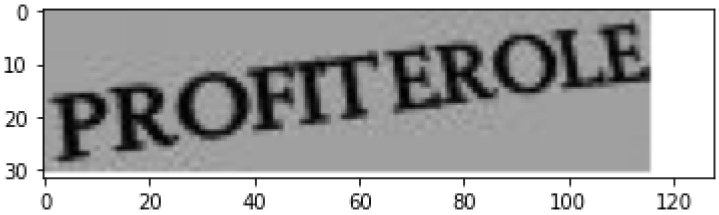
```
predicted text = Opines
```

```
-----------------Input Image-------------------
```



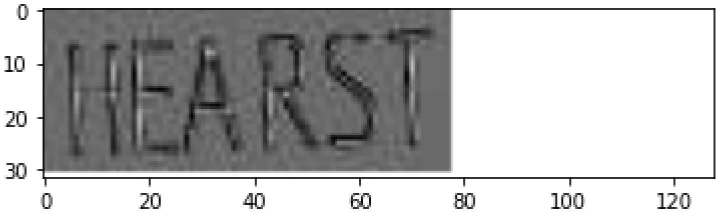
```
predicted text = Worshiper
```

```
-----------------Input Image-------------------
```



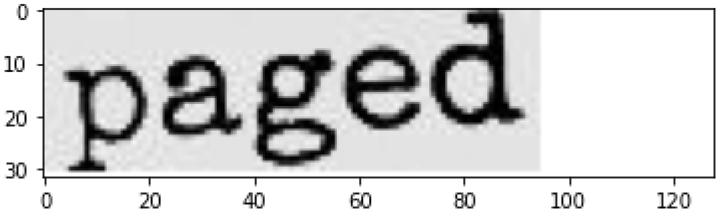
```
predicted text = PROFITEROLE
```

```
-----------------Input Image-------------------
```



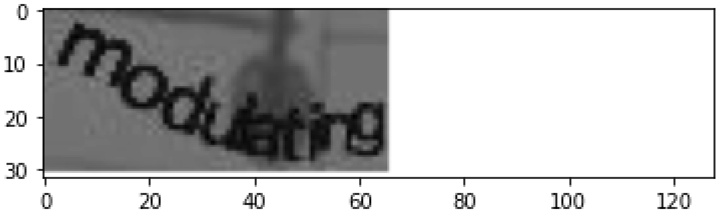
```
predicted text = TEARST
```

```
-----------------Input Image-------------------
```

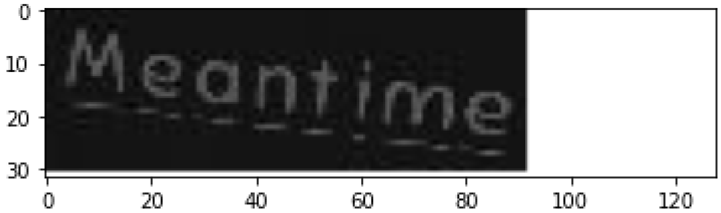


```
predicted text = paged
```

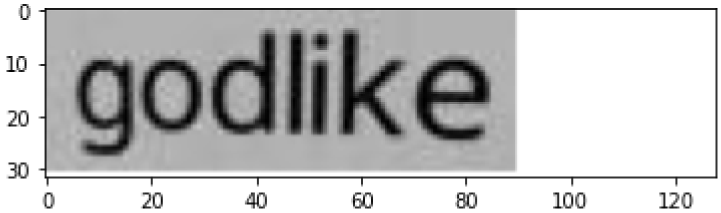
```
-----------------Input Image-------------------
```



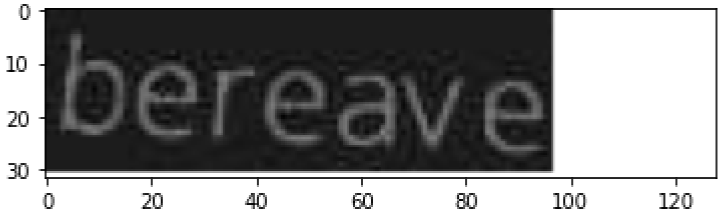
```
predicted text = mociatirg

-----------------Input Image-------------------
```



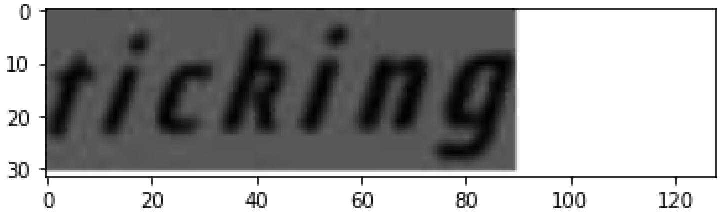
```
predicted text = Meantime

-----------------Input Image-------------------
```



```
predicted text = godlike

-----------------Input Image-------------------
```



```
predicted text = bereave

-----------------Input Image-------------------
```



```
predicted text = ticking
```

Sentiment Analysis

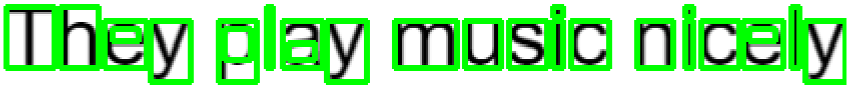
```
from craft_text_detector import Craft

def get_image_crop(image):
  output_dir = 'outputs/'
  # create a craft instance
  craft = Craft(crop_type="poly", cuda=False,output_dir=output_dir)

  # apply craft text detection and export detected regions to output directory
  prediction_result = craft.detect_text(image)
  print("Image adjusted crop saved")
  print("Image path is : /content/outputs/{}_crops/crop_0.png".format(image.split(".")[0]))
  return "/content/outputs/"+image.split(".")[0]+"_crops/crop_0.png"


image = 'positive.png'
cropped_path = get_image_crop(image)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:253: UserWarning: Ac
  "Accessing the model URLs via the internal dictionary of the module is deprecated sinc
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserWarning: Th
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be removed in 0
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWarning: Ar
  warnings.warn(msg)
Image adjusted crop saved
Image path is : /content/outputs/positive_crops/crop_0.png
```

```
cropped_path
```

```
'/content/outputs/positive_crops/crop_0.png'
```

```
import cv2
from google.colab.patches import cv2_imshow
import pandas as pd

def get_character_coordinates(cropped_path):
  img = cv2.imread(cropped_path)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  ret,thresh = cv2.threshold(gray,50,255,0)
  contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
  boxes = []
  for con in contours:
      if len(con)!=4:
          x,y,w,h = cv2.boundingRect(con)
          boxes.append([x,y,w,h])
          img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
  cv2_imshow(img)
  df = pd.DataFrame(boxes, columns =['x','y','w','h'])
  df.sort_values(by=['x'],inplace=True)
  return df



df = get_character_coordinates(cropped_path)
df
```



	x	y	w	h
24	1	1	24	30
23	28	1	19	30
17	50	8	21	23
18	54	12	13	6
16	73	8	20	30
14	107	8	19	30
15	110	12	13	15
22	131	1	3	30
12	139	8	20	23
13	142	19	13	8
11	161	8	20	30
10	195	8	31	23
9	230	8	18	23
8	252	8	19	23
7	274	8	5	23
21	274	1	5	5
6	283	8	19	23
5	316	8	18	23
4	340	8	4	23
20	340	1	4	5
3	348	8	19	23
1	369	8	20	23
2	372	12	13	6

```
import shutil

def get_cropped_words(cropped_path,df):
  if os.path.exists('words'):
    shutil.rmtree('words', ignore_errors=True)
  os.mkdir('words')
  image = cv2.imread(cropped_path)
  x_min=0
  y_min=0
  y_max=image.shape[0]
  word_count=1
  last = len(df.values)
  for i,row in enumerate(df.values):
      if i==0:
          x_max=row[0]+row[2]
      elif i==last-1:
          cropped_image = image[y_min:y_max, x_min:image.shape[1]]
          cv2.imwrite('words/word{}.png'.format(word_count),cropped_image)
          print("The word saved in path : words/word{}.png".format(word_count))
      else:
          if row[0]-x_max>10:
              mid = (x_max+row[0])//2
              cropped_image = image[y_min:y_max, x_min:mid]
              x_min=mid
              x_max=row[0]+row[2]
              cv2.imwrite('words/word{}.png'.format(word_count),cropped_image)
              print("The word saved in path : words/word{}.png".format(word_count))
              word_count+=1
          else :
              x_max=row[0]+row[2]
```

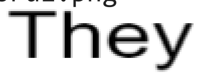
```
get_cropped_words(cropped_path,df)
```

```
The word saved in path : words/word1.png
The word saved in path : words/word2.png
The word saved in path : words/word3.png
The word saved in path : words/word4.png
```

```
def image_padding(words_path):
  words = sorted(os.listdir(words_path))
  for word in words:
    image = cv2.imread('/content/words/'+str(word))
    print(word)
    if image.shape[1]<128:
      padding_range = 128 - image.shape[1]
      x = padding_range//2
      white = [255,255,255]
      image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      constant= cv2.copyMakeBorder(image.copy(),0,0,x,x,cv2.BORDER_CONSTANT,value=white)
      constant = cv2.resize(constant, (128, 32))
      cv2_imshow(constant)
      print(constant.shape)
      cv2.imwrite('words/'+str(word),constant)
    else :
      constant = cv2.resize(image, (128, 32))
      cv2_imshow(constant)
      print(constant.shape)
      cv2.imwrite('words/'+str(word),constant)
```

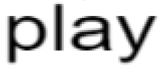
```
image_padding('/content/words/')
```

```
word1.png
```



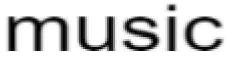
They

```
(32, 128)
word2.png
```



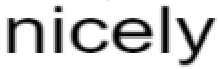
play

```
(32, 128)
word3.png
```



music

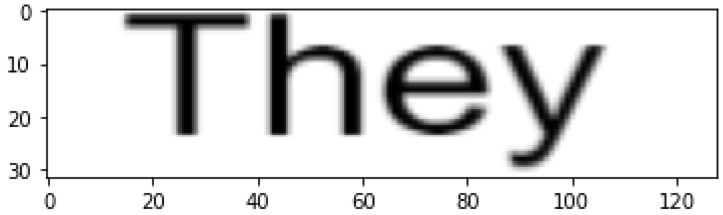
```
(32, 128)
word4.png
```

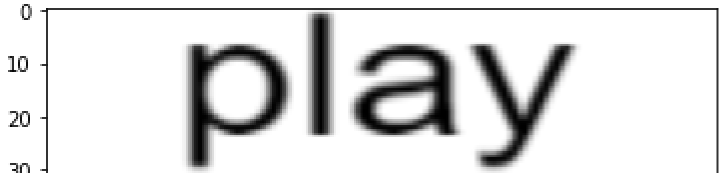


nicely

```
(32, 128)
```

```
def get_validation_images(words_path):
  words = sorted(os.listdir(words_path))
  val_images = []
  for word in words:
    img = cv2.cvtColor(cv2.imread(words_path+word), cv2.COLOR_BGR2GRAY)
    plt.imshow(img, cmap='gray')
    plt.show()
    img = np.expand_dims(img , axis = 2)
    img = img/255.
    val_images.append(img)

  # Normalize each image
  val_images = np.array(val_images)
  print(val_images.shape)
  return val_images


val_images = get_validation_images('/content/words/')
```





```
def get_predictions(val_images):
  prediction = actual_model.predict(val_images)

  # use CTC decoder
  out = K.get_value(K.ctc_decode(prediction, input_length=np.ones(prediction.shape[0])*predic
                        greedy=True)[0][0])
  i = 0
  words = []
  for x in out:
      print("predicted text = ", end = '')
      wrd = []
      for p in x:
          if int(p) != -1:
              print(characterset[int(p)], end = '')
              wrd.extend(characterset[int(p)])
      print('\n')
      words.append(wrd)
      i+=1
  return ["".join(wrd) for wrd in words]
```

```
predicted_words = get_predictions(val_images)
```

```
1/1 [==============================] - 0s 26ms/step
predicted text = They

predicted text = rplay

predicted text = music

predicted text = nicely
```

```
predicted_words
```

```
['They', 'rplay', 'music', 'nicely']
```

```
from textblob import Word
```

```
def spelling_correction(words):
  corrected_sentence = []
  for j in words:
    word = Word(j)
    corrected_sentence.append(word.correct())
  return " ".join(corrected_sentence)

corrected_sentence = spelling_correction(predicted_words)
corrected_sentence
```

```
'They play music nicely'
```

```
from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and
Using a pipeline without specifying a model name and revision in production is not recom
```

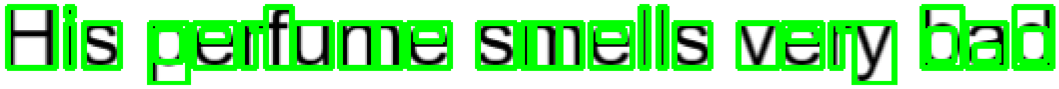
```
print("The input sentence is : ",corrected_sentence)
print(sentiment_pipeline([corrected_sentence]))
```

```
The input sentence is :  They play music nicely
[{'label': 'POSITIVE', 'score': 0.9998751878738403}]
```

```
image = 'negative.png'
cropped_path = get_image_crop(image)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:253: UserWarning: Ac
  "Accessing the model URLs via the internal dictionary of the module is deprecated sinc
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserWarning: Th
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be removed in 0
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWarning: Ar
  warnings.warn(msg)
Image adjusted crop saved
Image path is : /content/outputs/negative_crops/crop_0.png
```

```
df = get_character_coordinates(cropped_path)
df
```



His perfume smells very bad

	x	y	w	h
36	1	1	24	30
27	30	9	4	22
35	30	1	4	5
26	39	9	18	22
24	72	9	19	30
25	75	12	13	16
22	94	9	21	22
23	98	12	13	6
21	118	9	7	22
20	123	9	7	4
34	130	1	13	30
19	144	9	19	22
18	167	9	19	22
17	184	9	14	22
15	201	9	21	22
16	205	12	13	6
14	236	9	19	22
13	258	9	6	22
12	262	9	15	22
11	276	9	12	22
9	292	9	21	22
10	296	12	13	6
33	317	1	4	30
32	326	1	4	30
8	334	9	18	22
7	366	9	20	22
5	388	9	20	22
6	391	12	14	6
4	412	9	6	22

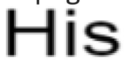
3	416	9	8	4
2	424	9	20	30
30	458	1	20	30
31	461	12	13	16
0	480	9	20	22
1	484	20	12	8

```
get_cropped_words(cropped_path,df)
```

```
The word saved in path : words/word1.png
The word saved in path : words/word2.png
The word saved in path : words/word3.png
The word saved in path : words/word4.png
The word saved in path : words/word5.png
```

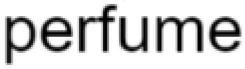
```
image_padding('/content/words/')
```

```
word1.png
```



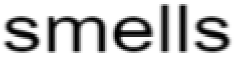
His

```
(32, 128)
word2.png
```



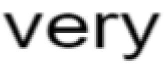
perfume

```
(32, 128, 3)
word3.png
```



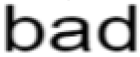
smells

```
(32, 128, 3)
word4.png
```



very

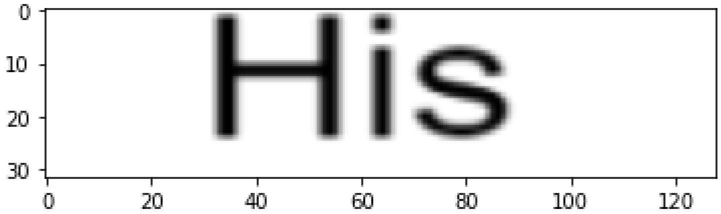
```
(32, 128)
word5.png
```

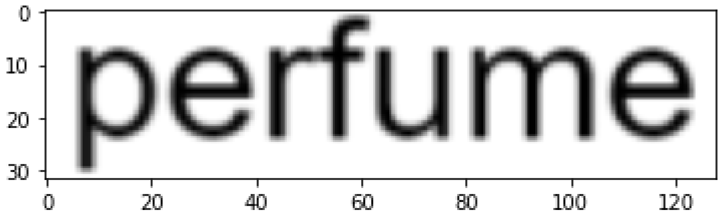


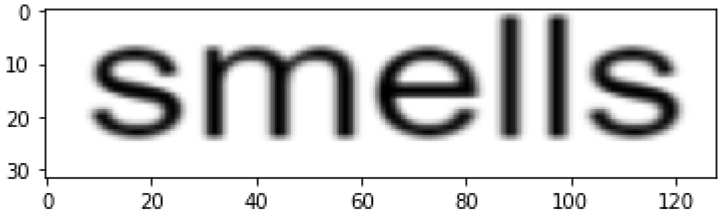
bad

```
(32, 128)
```

```
val_images = get_validation_images('/content/words/')
```







```
predicted_words = get_predictions(val_images)

     1/1 [==============================] - 0s 25ms/step
     predicted text = eHis

     predicted text = perfume

     predicted text = smells

     predicted text = very

     predicted text = ibad
```

```
corrected_sentence = spelling_correction(predicted_words)
corrected_sentence

     'his perfume smells very bad'
```

```
print("The input sentence is : ",corrected_sentence)
print(sentiment_pipeline([corrected_sentence]))

     The input sentence is :  his perfume smells very bad
     [{'label': 'NEGATIVE', 'score': 0.9997778534889221}]
```