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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import cv2
import os
import pandas as pd
import string
import matplotlib.pyplot as plt
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

from keras.layers import Dense, LSTM, Reshape, BatchNormalization, Input, Conv2D, MaxPool2D, Lambda, Bidirectional
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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
import tensorflow as tf

#ignore warnings in the output
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

```
!unzip '/content/drive/MyDrive/archive (6).zip'
```

```
Archive:  /content/drive/MyDrive/archive (6).zip
replace iam_words/words.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
with open('/content/words_new.txt') as f:
    contents = f.readlines()

lines = [line.strip() for line in contents]
lines[0]
```

```
max_label_len = 0

char_list = "!\"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

# string.ascii_letters + string.digits (Chars & Digits)
# or
# "!\"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

print(char_list, len(char_list))

def encode_to_labels(txt):
    # encoding each output word into digits
    dig_lst = []
    for index, chara in enumerate(txt):
        dig_lst.append(char_list.index(chara))

    return dig_lst
```

```
images = []
labels = []

RECORDS_COUNT = 10000
```

```
train_images = []
train_labels = []
train_input_length = []
train_label_length = []
train_original_text = []

valid_images = []
valid_labels = []
valid_input_length = []
valid_label_length = []
valid_original_text = []

inputs_length = []
labels_length = []
```

```
def process_image(img):
    """
    Converts image to shape (32, 128, 1) & normalize
    """
    w, h = img.shape

#     _, img = cv2.threshold(img,
#                            128,
#                            255,
```

```
#                           cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    # Aspect Ratio Calculation
    new_w = 32
    new_h = int(h * (new_w / w))
    img = cv2.resize(img, (new_h, new_w))
    w, h = img.shape

    img = img.astype('float32')

    # Converts each to (32, 128, 1)
    if w < 32:
        add_zeros = np.full((32-w, h), 255)
        img = np.concatenate((img, add_zeros))
        w, h = img.shape

    if h < 128:
        add_zeros = np.full((w, 128-h), 255)
        img = np.concatenate((img, add_zeros), axis=1)
        w, h = img.shape

    if h > 128 or w > 32:
        dim = (128,32)
        img = cv2.resize(img, dim)

    img = cv2.subtract(255, img)

    img = np.expand_dims(img, axis=2)

    # Normalize
    img = img / 255

    return img


for index, line in enumerate(lines):
    splits = line.split(' ')
    status = splits[1]

    if status == 'ok':
        word_id = splits[0]
        word = "".join(splits[8:])

        splits_id = word_id.split('-')
        filepath = '/content/iam_words/words/{}/{}-{}/{}.png'.format(splits_id[0],
                                                splits_id[0],
                                                splits_id[1],
                                                word_id)

        # process image
        img = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
        try:
            img = process_image(img)
        except:
            continue

        # process label
        try:
            label = encode_to_labels(word)
        except:
            continue

        if index % 10 == 0:
            valid_images.append(img)
            valid_labels.append(label)
            valid_input_length.append(31)
            valid_label_length.append(len(word))
            valid_original_text.append(word)
        else:
            train_images.append(img)
            train_labels.append(label)
            train_input_length.append(31)
            train_label_length.append(len(word))
            train_original_text.append(word)

        if len(word) > max_label_len:
            max_label_len = len(word)

    if index >= RECORDS_COUNT:
        break


train_padded_label = pad_sequences(train_labels,
                             maxlen=max_label_len,
                             padding='post',
                             value=len(char_list))

valid_padded_label = pad_sequences(valid_labels,
                             maxlen=max_label_len,
                             padding='post',
                             value=len(char_list))


train_padded_label.shape, valid_padded_label.shape
```

```
train_images = np.asarray(train_images)
train_input_length = np.asarray(train_input_length)
train_label_length = np.asarray(train_label_length)

valid_images = np.asarray(valid_images)
valid_input_length = np.asarray(valid_input_length)
valid_label_length = np.asarray(valid_label_length)


train_images.shape


# input with shape of height=32 and width=128
inputs = Input(shape=(32,128,1))

# convolution layer with kernel size (3,3)
conv_1 = Conv2D(64, (3,3), activation = 'relu', padding='same')(inputs)
# poolig layer with kernel size (2,2)
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)
# poolig layer with kernel size (2,1)
pool_4 = MaxPool2D(pool_size=(2, 1))(conv_4)

conv_5 = Conv2D(512, (3,3), activation = 'relu', padding='same')(pool_4)
# Batch normalization layer
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)

# bidirectional LSTM layers with units=128
blstm_1 = Bidirectional(LSTM(256, return_sequences=True, dropout = 0.5))(squeezed)
blstm_2 = Bidirectional(LSTM(256, return_sequences=True, dropout = 0.5))(blstm_1)

outputs = Dense(len(char_list)+1, activation = 'softmax')(blstm_2)

# model to be used at test time
act_model = Model(inputs, outputs)
act_model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 32, 128, 1)]      0

 conv2d_7 (Conv2D)           (None, 32, 128, 64)       640

 max_pooling2d_4 (MaxPooling  (None, 16, 64, 64)       0
 2D)

 conv2d_8 (Conv2D)           (None, 16, 64, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 8, 32, 128)       0
 2D)

 conv2d_9 (Conv2D)           (None, 8, 32, 256)        295168

 conv2d_10 (Conv2D)          (None, 8, 32, 256)        590080

 max_pooling2d_6 (MaxPooling  (None, 4, 32, 256)       0
 2D)

 conv2d_11 (Conv2D)          (None, 4, 32, 512)        1180160

 batch_normalization_2 (Batc  (None, 4, 32, 512)       2048
 hNormalization)

 conv2d_12 (Conv2D)          (None, 4, 32, 512)        2359808

 batch_normalization_3 (Batc  (None, 4, 32, 512)       2048
 hNormalization)

 max_pooling2d_7 (MaxPooling  (None, 2, 32, 512)       0
 2D)

 conv2d_13 (Conv2D)          (None, 1, 31, 512)        1049088

 lambda_1 (Lambda)           (None, 31, 512)           0

 bidirectional_2 (Bidirectio  (None, 31, 512)          1574912
 nal)

 bidirectional_3 (Bidirectio  (None, 31, 512)          1574912
 nal)

 dense_1 (Dense)             (None, 31, 79)            40527

=================================================================
```

```
Total params: 8,743,247
Trainable params: 8,741,199
Non-trainable params: 2,048
_________________________________________________________________
```

```
the_labels = Input(name='the_labels', shape=[max_label_len], dtype='float32')
input_length = Input(name='input_length', shape=[1], dtype='int64')
label_length = Input(name='label_length', shape=[1], dtype='int64')

def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args

    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)

loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([outputs, the_labels, input_length, label_length])

#model to be used at training time
model = Model(inputs=[inputs, the_labels, input_length, label_length], outputs=loss_out)


batch_size = 32
epochs = 50
e = str(epochs)
optimizer_name = 'adam'


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

filepath="model.h5"

checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='auto')
callbacks_list = [checkpoint]


history = model.fit(x=[train_images, train_padded_label, train_input_length, train_label_length],
                    y=np.zeros(len(train_images)),
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=([valid_images, valid_padded_label, valid_input_length, valid_label_length], [np.zeros(len(valid_images))]),
                    verbose=2,
                    callbacks=callbacks_list)
```

```
Epoch 1/50

Epoch 1: val_loss improved from inf to 15.24071, saving model to model.h5
246/246 - 26s - loss: 15.6412 - val_loss: 15.2407 - 26s/epoch - 107ms/step
Epoch 2/50

Epoch 2: val_loss did not improve from 15.24071
246/246 - 14s - loss: 13.2970 - val_loss: 17.4308 - 14s/epoch - 56ms/step
Epoch 3/50

Epoch 3: val_loss improved from 15.24071 to 11.87891, saving model to model.h5
246/246 - 14s - loss: 11.9153 - val_loss: 11.8789 - 14s/epoch - 57ms/step
Epoch 4/50
```

```
!pip install Levenshtein
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting Levenshtein
  Downloading Levenshtein-0.20.9-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (175 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 175.5/175.5 KB 5.1 MB/s eta 0:00:00
Collecting rapidfuzz<3.0.0,>=2.3.0
  Downloading rapidfuzz-2.13.7-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 37.9 MB/s eta 0:00:00
Installing collected packages: rapidfuzz, Levenshtein
Successfully installed Levenshtein-0.20.9 rapidfuzz-2.13.7
```

```
# load the saved best model weights
act_model.load_weights(filepath)

# predict outputs on validation images
prediction = act_model.predict(valid_images)

# use CTC decoder
decoded = K.ctc_decode(prediction,
                       input_length=np.ones(prediction.shape[0]) * prediction.shape[1],
                       greedy=True)[0][0]
out = K.get_value(decoded)

import Levenshtein as lv

total_jaro = 0
total_rati = 0
# see the results
for i, x in enumerate(out):
    letters=''
    for p in x:
        if int(p) != -1:
            letters+=char_list[int(p)]
    total_jaro+=lv.jaro(letters, valid_original_text[i])
    total_rati+=lv.ratio(letters, valid_original_text[i])

print('jaro :', total_jaro/len(out))
print('ratio:', total_rati/len(out))
```

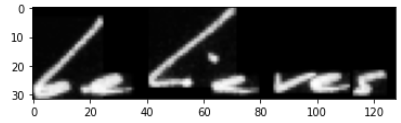
```
     28/28 [==============================] - 2s 17ms/step
     jaro : 0.9103633819672641
     ratio: 0.877084181957554
```

```
train_images[110:130].shape
```

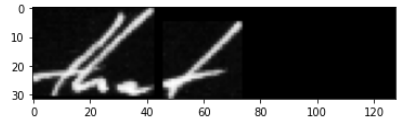
```
     (20, 32, 128, 1)
```

```
# predict outputs on validation images
prediction = act_model.predict(train_images[110:130])

# use CTC decoder
decoded = K.ctc_decode(prediction,
                       input_length=np.ones(prediction.shape[0]) * prediction.shape[1],
                       greedy=True)[0][0]

out = K.get_value(decoded)

# see the results
for i, x in enumerate(out):
    print("original_text =  ", train_original_text[110+i])
    print("predicted text = ", end = '')
    for p in x:
        if int(p) != -1:
            print(char_list[int(p)], end = '')
    plt.imshow(train_images[110+i].reshape(32,128), cmap=plt.cm.gray)
    plt.show()
    print('\n')
```

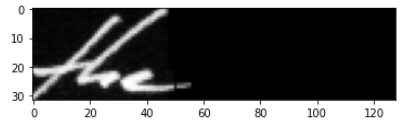
.



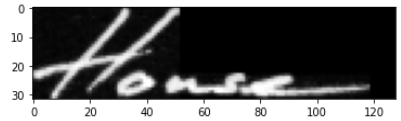
original_text = that
predicted text = that



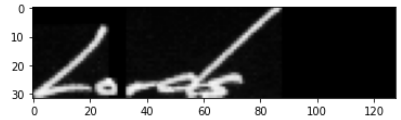
original_text = the
predicted text = the



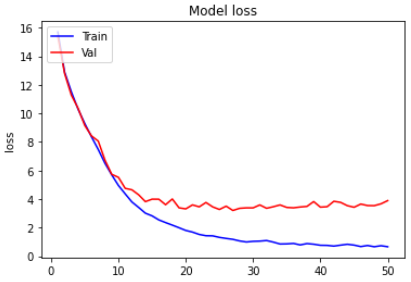
original_text = House
predicted text = House



original_text = Lords
predicted text = Loods



```
# plot accuracy and loss
def plotgraph(epochs, loss, val_loss):
    # Plot training & validation loss values
    plt.plot(epochs, loss, 'b')
    plt.plot(epochs, val_loss, 'r')
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()


loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)


plotgraph(epochs, loss, val_loss)
```




```
import cv2
import numpy as np


# Load the character list used for training
char_list = "!\"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

# Function to preprocess the image
def process_image(img):
    w, h = img.shape

    # Aspect Ratio Calculation
    new_w = 32
    new_h = int(h * (new_w / w))
    img = cv2.resize(img, (new_h, new_w))
    w, h = img.shape

    img = img.astype('float32')

    # Converts each to (32, 128, 1)
    if w < 32:
        add_zeros = np.full((32-w, h), 255)
        img = np.concatenate((img, add_zeros))
        w, h = img.shape

    if h < 128:
        add_zeros = np.full((w, 128-h), 255)
        img = np.concatenate((img, add_zeros), axis=1)
        w, h = img.shape

    if h > 128 or w > 32:
        dim = (128,32)
        img = cv2.resize(img, dim)

    img = cv2.subtract(255, img)

    img = np.expand_dims(img, axis=2)

    # Normalize
    img = img / 255

    return img

# Function to encode the output word into digits
def encode_to_labels(txt):
    # encoding each output word into digits
    dig_lst = []
    for index, chara in enumerate(txt):
        dig_lst.append(char_list.index(chara))

    return dig_lst

# Load the raw image
img = cv2.imread('/content/iam_words/words/a01/a01-007/a01-007-00-06.png', cv2.IMREAD_GRAYSCALE)


# Preprocess the image
img = process_image(img)

plt.imshow(img.reshape(32,128), cmap=plt.cm.gray)
```