BDA Final Notebook

December 5, 2020

1 BigData Analytics Course Project

1.1 Project Topic: Apply ML Algorithms to Dataset

1.1.1 Aim: To recognize Handwriting using CNN + RNN with Keras and TensorFlow package

Presented by,
* Name: Surya Narayanan S
* Reg No: 121015098
* Year & Dept: 4 IT

- Dataset: https://www.kaggle.com/landlord/handwriting-recognition
- Dataset size: 1.3 GB
- No of Images: 3.72 Lakhs
- A CRNN (Convolutional + Recurrent Neural Network) Machine Learning Model is built to recognize and convert Handwritten Images to text using TensorFlow and Keras package in python.
- Accuracy, Support, Confusion matrix, Classification Report, Similarity Histogram are used for evaluation.

1.1.2 Where is it used ?

- To digitalize physical forms (Name, Age etc., details from physical form can be recognized and converted to text)
- Handwriting virtual Keyboard (In mobile phones, we can write in the screen using stylus or hand touch . Which is then converted to text)

[1]:
```
#Basic Imports
import os
import cv2
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
from IPython.display import display
from PIL import Image
import pylab as pl
```

[2]:
```python
#Tech Stack
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Reshape,␣
↪Bidirectional, LSTM, Dense, Lambda, Activation, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, confusion_matrix,␣
↪mean_squared_error
from Levenshtein import jaro_winkler
```

[3]:
```python
os.chdir('C:\\Users\\surya\\Documents\\Lab\\Python\\BigDataProject')

train = pd.read_csv("HandwrittenDataset/written_name_train_v2.csv")
valid = pd.read_csv("HandwrittenDataset/written_name_validation_v2.csv")

display(train.describe(include="all"))
display(train.head())
```

	FILENAME	IDENTITY
count	330961	330396
unique	330961	100539
top	TRAIN_248527.jpg	THOMAS
freq	1	1825

	FILENAME	IDENTITY
0	TRAIN_00001.jpg	BALTHAZAR
1	TRAIN_00002.jpg	SIMON
2	TRAIN_00003.jpg	BENES
3	TRAIN_00004.jpg	LA LOVE
4	TRAIN_00005.jpg	DAPHNE

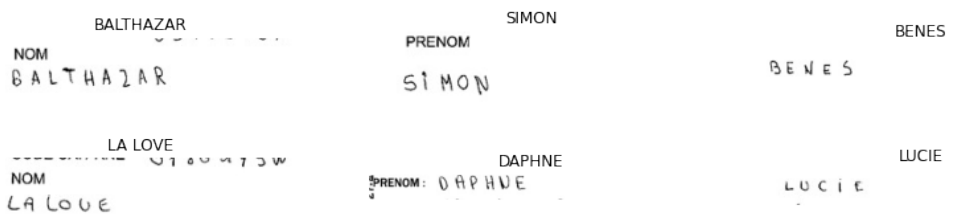
[4]:
```python
print("Number of NaNs in train set      : ", train['IDENTITY'].isnull().sum())
print("Number of NaNs in validation set : ", valid['IDENTITY'].isnull().sum())

train.dropna(axis=0, inplace=True)
valid.dropna(axis=0, inplace=True)
```

```
Number of NaNs in train set      :  565
Number of NaNs in validation set :  78
```

[5]:
```python
#Sample Images
plt.figure(figsize=(15, 10))
print("Sample Images")
for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    img_dir = 'HandwrittenDataset/train_v2/train/'+train.loc[i, 'FILENAME']
```

```
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap = 'gray')
    plt.title(train.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')

plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```

Sample Images

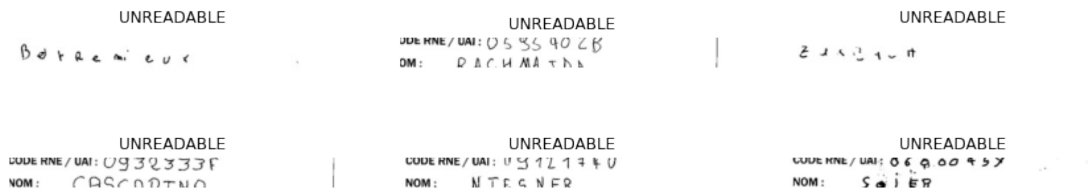


[6]:

```
#Remove Unreadable
unreadable = train[train['IDENTITY'] == 'UNREADABLE']
unreadable.reset_index(inplace = True, drop=True)
print("No of Unreadable Images " + str(len(unreadable)))
print("Sample Unreadable Images")
plt.figure(figsize=(15, 10))

for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    img_dir = 'HandwrittenDataset/train_v2/train/'+unreadable.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap = 'gray')
    plt.title(unreadable.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')

plt.subplots_adjust(wspace=0.2, hspace=-0.8)
plt.show()

train = train[train['IDENTITY'] != 'UNREADABLE']
valid = valid[valid['IDENTITY'] != 'UNREADABLE']

#Convert all Characters to Upper Case

train['IDENTITY'] = train['IDENTITY'].str.upper()
valid['IDENTITY'] = valid['IDENTITY'].str.upper()



train.reset_index(inplace = True, drop=True)
```

```
valid.reset_index(inplace = True, drop=True)
print("After Cleaning")
print("Length of Train Dataset ", len(train))
print("Length of Validation Dataset ", len(valid))
```

```
No of Unreadable Images 102
Sample Unreadable Images
```




```
After Cleaning
Length of Train Dataset  330294
Length of Validation Dataset  41280
```

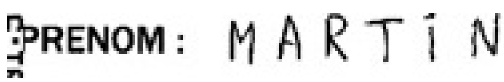
[7]:

```
#Image Preprocessing
def preprocess(img):


    #gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, img = cv2.threshold(img, 0, 255, cv2.THRESH_OTSU | cv2.
↪THRESH_BINARY_INV)
    #rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
    #dilation = cv2.dilate(thresh1, rect_kernel, iterations = 1)
    #display(Image.fromarray(thresh1))
    (h, w) = img.shape
    final_img = np.ones([64, 256])*0 # blank white image

    # crop
    if w > 256:
        img = img[:, :256]

    if h > 64:
        img = img[:64, :]


    final_img[:h, :w] = img
    final_img = final_img.astype("uint8")
    #display(Image.fromarray(final_img))
    final_img = cv2.rotate(final_img, cv2.ROTATE_90_CLOCKWISE)
    #display(Image.fromarray(final_img))
    return final_img
```

[8]:

```python
#Histogram of sample Image before preprocessing
fle = "HandwrittenDataset/test_v2/test/TEST_0006.jpg"
#fle = "HandwrittenDataset/Hello.jpg"
#fle = "HandwrittenDataset/Chandra1.jpg"
img = cv2.imread(fle, cv2.IMREAD_GRAYSCALE)
print("Image Before Pre-Processing")
display(Image.fromarray(img))
img2 = img.flatten()
plt.hist(img2)
plt.title("Histogram before Pre-Processing")
plt.xlabel("GrayScale Value")
plt.ylabel("Frequency")
plt.show()

processed = preprocess(img)
print("Image after Pre-Processing")
display(Image.fromarray(processed))
img2 = processed.flatten()
plt.hist(img2)
plt.title("Histogram after Pre-Processing")
plt.xlabel("GrayScale Value")
plt.ylabel("Frequency")
plt.show()
```

```
Image Before Pre-Processing
```



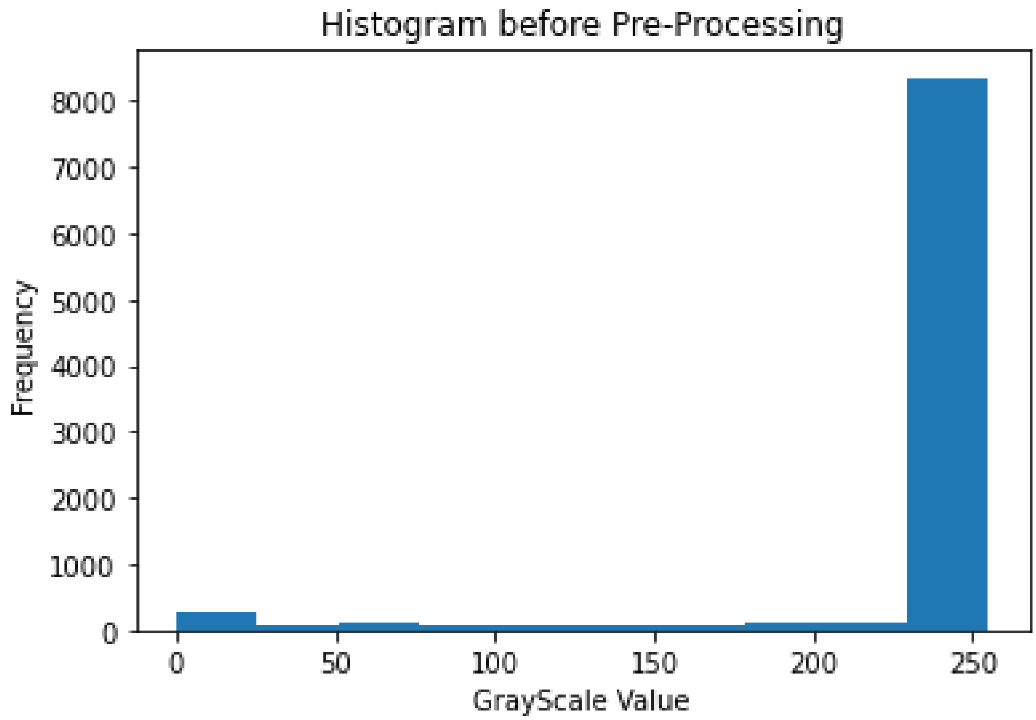
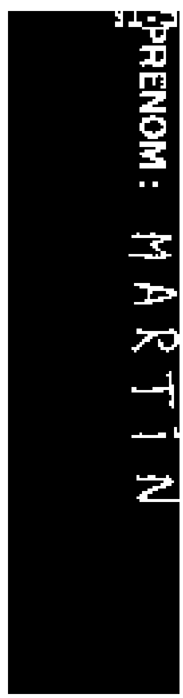
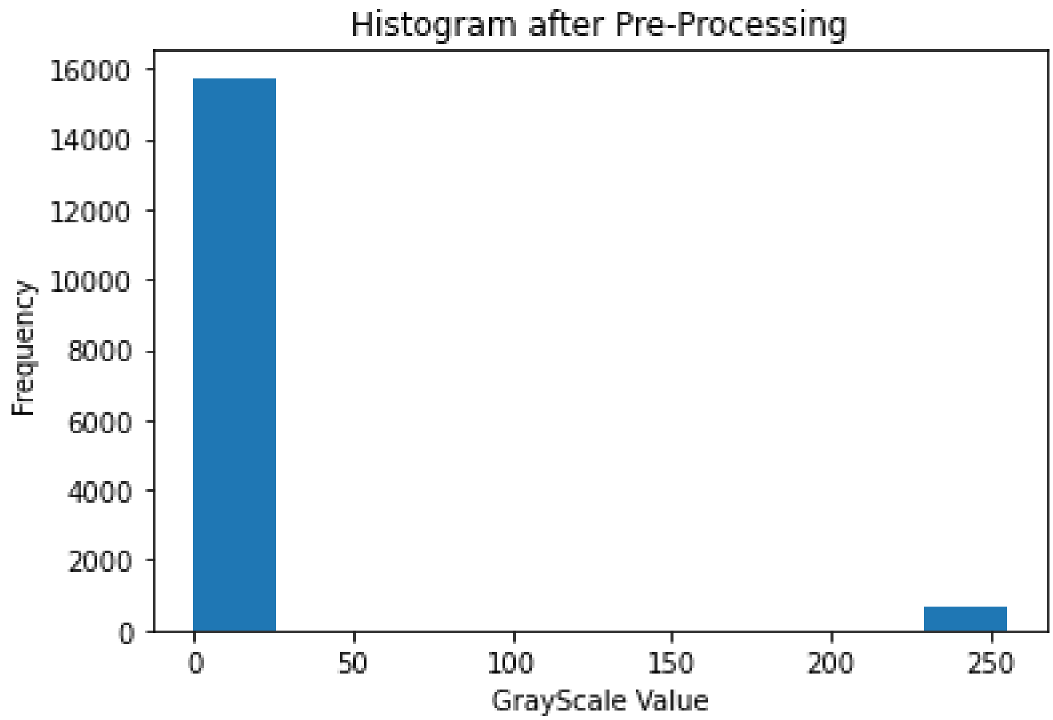


Image after Pre-Processing





[9]:
```
#Read Train Images from a folder and preprocess it using OpenCV

train_size = 30000
train_x = []

for i in range(train_size):
    img_dir = 'HandwrittenDataset/train_v2/train/'+train.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    image = preprocess(image)
    image = image/255.
    train_x.append(image)
train_x = np.array(train_x).reshape(-1, 256, 64, 1)
print("Train Images Size ", train_x.shape)
```

```
Train Images Size  (30000, 256, 64, 1)
```

[10]:
```
#Read Validation Images from a folder and preprocess it using OpenCV

valid_x = []
valid_size= 3000
start_ind = 0
for i in range(start_ind,start_ind+valid_size):
    img_dir = 'HandwrittenDataset/validation_v2/validation/'+valid.loc[i,␣
↪'FILENAME']
```

```
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    image = preprocess(image)
    image = image/255.
    valid_x.append(image)
valid_x = np.array(valid_x).reshape(-1, 256, 64, 1)
print("Validation Images Size ",valid_x.shape)
```

```
Validation Images Size  (3000, 256, 64, 1)
```

[11]:

```
# Map characters to a specific number from 0 to 26
alphabets = u"ABCDEFGHIJKLMNOPQRSTUVWXYZ-' "
max_str_len = 24 # max length of input labels
num_of_characters = len(alphabets) + 1 # +1 for ctc pseudo blank
num_of_timestamps = 64 # max length of predicted labels

#Convert a given label to its corresponding Number
def label_to_num(label):
    label_num = []
    for ch in label:
        label_num.append(alphabets.find(ch))

    return np.array(label_num)

#Convert given number list to its corresponding String
def num_to_label(num):
    ret = ""
    for ch in num:
        if ch == -1:  # CTC Blank
            break
        else:
            ret+=alphabets[ch]
    return ret



name = 'JEBASTIN'
num_list = label_to_num(name)
print("Name to Number List")
print(name, ' to ',label_to_num(name))

print("Number List to name")
print(num_list," to ",num_to_label(num_list))
```

```
Name to Number List
JEBASTIN  to  [ 9  4  1  0 18 19  8 13]
Number List to name
[ 9  4  1  0 18 19  8 13]  to  JEBASTIN
```

[12]:
```
#Create Input matrix to give it to the Model
train_y = np.ones([train_size, max_str_len]) * -1
train_label_len = np.zeros([train_size, 1])
train_input_len = np.ones([train_size, 1]) * (num_of_timestamps-2)
train_output = np.zeros([train_size])

for i in range(train_size):
    train_label_len[i] = len(train.loc[i, 'IDENTITY'])
    train_y[i, 0:len(train.loc[i, 'IDENTITY'])]= label_to_num(train.loc[i,␣
↪'IDENTITY'])



print('True label : ',train.loc[100, 'IDENTITY'] , '\ntrain_y :␣
↪',train_y[100],'\ntrain_label_len : ',train_label_len[100],␣
↪'\ntrain_input_len : ', train_input_len[100])
```

```
True label :  NOUR
train_y :  [13. 14. 20. 17. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
-1.
 -1. -1. -1. -1. -1. -1.]
train_label_len :  [4.]
train_input_len :  [62.]
```

[13]:
```
#Create validation matrix to give it to the model for learning
valid_y = np.ones([valid_size, max_str_len]) * -1
valid_label_len = np.zeros([valid_size, 1])
valid_input_len = np.ones([valid_size, 1]) * (num_of_timestamps-2)
valid_output = np.zeros([valid_size])

for i in range(start_ind,start_ind + valid_size):
    valid_label_len[i-start_ind] = len(valid.loc[i, 'IDENTITY'])
    valid_y[i-start_ind, 0:len(valid.loc[i, 'IDENTITY'])]= label_to_num(valid.
↪loc[i, 'IDENTITY'])

print('True label : ',valid.loc[100, 'IDENTITY'] , '\nvalid_y :␣
↪',valid_y[100],'\nvalid_label_len : ',valid_label_len[100],␣
↪'\nvalid_input_len : ', valid_input_len[100])
```

```
True label :  CLOE
valid_y :  [ 2. 11. 14.  4. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
-1.
 -1. -1. -1. -1. -1. -1.]
valid_label_len :  [4.]
valid_input_len :  [62.]
```

[14]:
```
#Construct the CRNN model
```

```
#https://towardsdatascience.com/
↪a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53


input_data = Input(shape=(256, 64, 1), name='input')

inner = Conv2D(32, (3, 3), padding='same', name='conv1',␣
↪kernel_initializer='he_normal')(input_data)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

inner = Conv2D(64, (3, 3), padding='same', name='conv2',␣
↪kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)
inner = Dropout(0.3)(inner)

inner = Conv2D(128, (3, 3), padding='same', name='conv3',␣
↪kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)
inner = Dropout(0.3)(inner)

# CNN to RNN
inner = Reshape(target_shape=((64, 1024)), name='reshape')(inner)
inner = Dense(64, activation='relu', kernel_initializer='he_normal',␣
↪name='dense1')(inner)

## RNN
#https://colah.github.io/posts/2015-08-Understanding-LSTMs/
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm1')(inner)
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm2')(inner)

## OUTPUT
inner = Dense(num_of_characters,␣
↪kernel_initializer='he_normal',name='dense2')(inner)
y_pred = Activation('softmax', name='softmax')(inner)

model = Model(inputs=input_data, outputs=y_pred)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input (InputLayer)           [(None, 256, 64, 1)]      0

_________________________________________________________________
conv1 (Conv2D)               (None, 256, 64, 32)       320

_________________________________________________________________
batch_normalization (BatchNo (None, 256, 64, 32)       128

_________________________________________________________________
activation (Activation)      (None, 256, 64, 32)       0

_________________________________________________________________
max1 (MaxPooling2D)          (None, 128, 32, 32)       0

_________________________________________________________________
conv2 (Conv2D)               (None, 128, 32, 64)       18496

_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 32, 64)       256

_________________________________________________________________
activation_1 (Activation)    (None, 128, 32, 64)       0

_________________________________________________________________
max2 (MaxPooling2D)          (None, 64, 16, 64)        0

_________________________________________________________________
dropout (Dropout)            (None, 64, 16, 64)        0

_________________________________________________________________
conv3 (Conv2D)               (None, 64, 16, 128)       73856

_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 16, 128)       512

_________________________________________________________________
activation_2 (Activation)    (None, 64, 16, 128)       0

_________________________________________________________________
max3 (MaxPooling2D)          (None, 64, 8, 128)        0

_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 8, 128)        0

_________________________________________________________________
reshape (Reshape)            (None, 64, 1024)          0

_________________________________________________________________
dense1 (Dense)               (None, 64, 64)            65600

_________________________________________________________________
lstm1 (Bidirectional)        (None, 64, 512)           657408

_________________________________________________________________
lstm2 (Bidirectional)        (None, 64, 512)           1574912

_________________________________________________________________
dense2 (Dense)               (None, 64, 30)            15390

_________________________________________________________________
softmax (Activation)         (None, 64, 30)            0
=================================================================
Total params: 2,406,878
Trainable params: 2,406,430
Non-trainable params: 448

_________________________________________________________________
```

[15]:
```
# the ctc loss function
#https://towardsdatascience.com/
↪intuitively-understanding-connectionist-temporal-classification-3797e43a86c

def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    # the 2 is critical here since the first couple outputs of the RNN
    # tend to be garbage
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)
```

[16]:
```
#Compile the built model with Adam Optimizer and CTC Loss function

labels = Input(name='gtruth_labels', shape=[max_str_len], dtype='float32')
input_length = Input(name='input_length', shape=[1], dtype='int64')
label_length = Input(name='label_length', shape=[1], dtype='int64')

ctc_loss = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred,␣
↪labels, input_length, label_length])
model_final = Model(inputs=[input_data, labels, input_length, label_length],␣
↪outputs=ctc_loss)


# the loss calculation occurs elsewhere, so we use a dummy lambda function for␣
↪the loss
model_final.compile(loss={'ctc': lambda y_true, y_pred: y_pred},␣
↪optimizer=Adam(lr = 0.0001))
model_final.summary()
```

```
Model: "model_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input (InputLayer)              [(None, 256, 64, 1)] 0
________________________________________________________________________________
__________________
conv1 (Conv2D)                  (None, 256, 64, 32)  320         input[0][0]
________________________________________________________________________________
__________________
batch_normalization (BatchNorma (None, 256, 64, 32)  128         conv1[0][0]
________________________________________________________________________________
__________________
activation (Activation)         (None, 256, 64, 32)  0
batch_normalization[0][0]
________________________________________________________________________________
```

```
__________________
max1 (MaxPooling2D)             (None, 128, 32, 32)  0
activation[0][0]
________________________________________________________________________________
__________________
conv2 (Conv2D)                  (None, 128, 32, 64)  18496       max1[0][0]
________________________________________________________________________________
__________________
batch_normalization_1 (BatchNor (None, 128, 32, 64)  256         conv2[0][0]
________________________________________________________________________________
__________________
activation_1 (Activation)       (None, 128, 32, 64)  0
batch_normalization_1[0][0]
________________________________________________________________________________
__________________
max2 (MaxPooling2D)             (None, 64, 16, 64)   0
activation_1[0][0]
________________________________________________________________________________
__________________
dropout (Dropout)               (None, 64, 16, 64)   0           max2[0][0]
________________________________________________________________________________
__________________
conv3 (Conv2D)                  (None, 64, 16, 128)  73856       dropout[0][0]
________________________________________________________________________________
__________________
batch_normalization_2 (BatchNor (None, 64, 16, 128)  512         conv3[0][0]
________________________________________________________________________________
__________________
activation_2 (Activation)       (None, 64, 16, 128)  0
batch_normalization_2[0][0]
________________________________________________________________________________
__________________
max3 (MaxPooling2D)             (None, 64, 8, 128)   0
activation_2[0][0]
________________________________________________________________________________
__________________
dropout_1 (Dropout)             (None, 64, 8, 128)   0           max3[0][0]
________________________________________________________________________________
__________________
reshape (Reshape)               (None, 64, 1024)     0           dropout_1[0][0]
________________________________________________________________________________
__________________
dense1 (Dense)                  (None, 64, 64)       65600       reshape[0][0]
________________________________________________________________________________
__________________
lstm1 (Bidirectional)           (None, 64, 512)      657408      dense1[0][0]
________________________________________________________________________________
__________________
```

```
lstm2 (Bidirectional)          (None, 64, 512)      1574912     lstm1[0][0]
--------------------------------------------------------------------------------
------------------
dense2 (Dense)                 (None, 64, 30)       15390       lstm2[0][0]
--------------------------------------------------------------------------------
------------------
softmax (Activation)           (None, 64, 30)       0           dense2[0][0]
--------------------------------------------------------------------------------
------------------
gtruth_labels (InputLayer)     [(None, 24)]         0
--------------------------------------------------------------------------------
------------------
input_length (InputLayer)      [(None, 1)]          0
--------------------------------------------------------------------------------
------------------
label_length (InputLayer)      [(None, 1)]          0
--------------------------------------------------------------------------------
------------------
ctc (Lambda)                   (None, 1)            0           softmax[0][0]
gtruth_labels[0][0]
input_length[0][0]
label_length[0][0]
================================================================================
==================
Total params: 2,406,878
Trainable params: 2,406,430
Non-trainable params: 448
--------------------------------------------------------------------------------
------------------
```

[17]:

```
#Fit (Train) the model with Train dataset and validation dataset
history = model_final.fit(x=[train_x, train_y, train_input_len,␣
↪train_label_len], y=train_output,  validation_data=([valid_x, valid_y,␣
↪valid_input_len, valid_label_len], valid_output), epochs=60, batch_size=128)
#model.save("tfmodelforhandwriting2")
```

```
Train on 30000 samples, validate on 3000 samples
Epoch 1/60
30000/30000 [==============================] - 346s 12ms/sample - loss: 27.4047
- val_loss: 20.6685
Epoch 2/60
30000/30000 [==============================] - 324s 11ms/sample - loss: 20.2041
- val_loss: 20.1329
Epoch 3/60
30000/30000 [==============================] - 322s 11ms/sample - loss: 19.8590
- val_loss: 19.7312
Epoch 4/60
30000/30000 [==============================] - 324s 11ms/sample - loss: 19.3577
```

```
- val_loss: 19.4887
Epoch 5/60
30000/30000 [==============================] - 326s 11ms/sample - loss: 18.3595
- val_loss: 17.8821
Epoch 6/60
30000/30000 [==============================] - 327s 11ms/sample - loss: 16.4965
- val_loss: 15.6969
Epoch 7/60
30000/30000 [==============================] - 331s 11ms/sample - loss: 14.0899
- val_loss: 12.9821
Epoch 8/60
30000/30000 [==============================] - 338s 11ms/sample - loss: 11.5439
- val_loss: 10.2836
Epoch 9/60
30000/30000 [==============================] - 457s 15ms/sample - loss: 9.1600 -
val_loss: 7.6016
Epoch 10/60
30000/30000 [==============================] - 252s 8ms/sample - loss: 7.4566 -
val_loss: 6.3227
Epoch 11/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 6.3708 -
val_loss: 5.4702
Epoch 12/60
30000/30000 [==============================] - 255s 8ms/sample - loss: 5.6287 -
val_loss: 4.9138
Epoch 13/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 5.0709 -
val_loss: 4.4533
Epoch 14/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 4.6538 -
val_loss: 4.0078
Epoch 15/60
30000/30000 [==============================] - 251s 8ms/sample - loss: 4.3035 -
val_loss: 3.8429
Epoch 16/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 4.0188 -
val_loss: 3.5299
Epoch 17/60
30000/30000 [==============================] - 246s 8ms/sample - loss: 3.7916 -
val_loss: 3.3993
Epoch 18/60
30000/30000 [==============================] - 246s 8ms/sample - loss: 3.5884 -
val_loss: 3.1865
Epoch 19/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 3.4148 -
val_loss: 3.1259
Epoch 20/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 3.2682 -
```

```
val_loss: 3.0308
Epoch 21/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 3.1389 -
val_loss: 2.9397
Epoch 22/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 3.0173 -
val_loss: 2.8733
Epoch 23/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 2.8983 -
val_loss: 2.7407
Epoch 24/60
30000/30000 [==============================] - 254s 8ms/sample - loss: 2.8199 -
val_loss: 2.6947
Epoch 25/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 2.7258 -
val_loss: 2.6513
Epoch 26/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 2.6494 -
val_loss: 2.5457
Epoch 27/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 2.5757 -
val_loss: 2.6148
Epoch 28/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 2.5049 -
val_loss: 2.4876
Epoch 29/60
30000/30000 [==============================] - 247s 8ms/sample - loss: 2.4419 -
val_loss: 2.4605
Epoch 30/60
30000/30000 [==============================] - 247s 8ms/sample - loss: 2.3824 -
val_loss: 2.4245
Epoch 31/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 2.3292 -
val_loss: 2.4019
Epoch 32/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 2.2588 -
val_loss: 2.3331
Epoch 33/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 2.2112 -
val_loss: 2.3519
Epoch 34/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 2.1535 -
val_loss: 2.3066
Epoch 35/60
30000/30000 [==============================] - 246s 8ms/sample - loss: 2.1200 -
val_loss: 2.2845
Epoch 36/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 2.0803 -
```

```
val_loss: 2.2995
Epoch 37/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 2.0258 -
val_loss: 2.2557
Epoch 38/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.9942 -
val_loss: 2.2298
Epoch 39/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.9443 -
val_loss: 2.2369
Epoch 40/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.9100 -
val_loss: 2.2292
Epoch 41/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.8683 -
val_loss: 2.1887
Epoch 42/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.8402 -
val_loss: 2.1828
Epoch 43/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.8056 -
val_loss: 2.1601
Epoch 44/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.7624 -
val_loss: 2.1111
Epoch 45/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.7366 -
val_loss: 2.1557
Epoch 46/60
30000/30000 [==============================] - 247s 8ms/sample - loss: 1.6945 -
val_loss: 2.1256
Epoch 47/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.6682 -
val_loss: 2.1421
Epoch 48/60
30000/30000 [==============================] - 251s 8ms/sample - loss: 1.6340 -
val_loss: 2.0936
Epoch 49/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.5954 -
val_loss: 2.1428
Epoch 50/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.5753 -
val_loss: 2.1442
Epoch 51/60
30000/30000 [==============================] - 246s 8ms/sample - loss: 1.5479 -
val_loss: 2.1363
Epoch 52/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.5089 -
```

```
val_loss: 2.0912
Epoch 53/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.4991 -
val_loss: 2.1337
Epoch 54/60
30000/30000 [==============================] - 247s 8ms/sample - loss: 1.4600 -
val_loss: 2.1432
Epoch 55/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 1.4417 -
val_loss: 2.1204
Epoch 56/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.4026 -
val_loss: 2.1428
Epoch 57/60
30000/30000 [==============================] - 246s 8ms/sample - loss: 1.3715 -
val_loss: 2.1028
Epoch 58/60
30000/30000 [==============================] - 249s 8ms/sample - loss: 1.3489 -
val_loss: 2.1268
Epoch 59/60
30000/30000 [==============================] - 248s 8ms/sample - loss: 1.3279 -
val_loss: 2.1332
Epoch 60/60
30000/30000 [==============================] - 250s 8ms/sample - loss: 1.3042 -
val_loss: 2.1523
```

[38]:
```
#To Load the saved Model
model = tf.keras.models.load_model("tfmodelforhandwriting2")
```

[76]:
```
#Predict handwritten Images
preds = model.predict(valid_x)
decoded = K.get_value(K.ctc_decode(preds, input_length=np.ones(preds.
 ↪shape[0])*preds.shape[1], greedy=True)[0][0])

prediction = []
for i in range(valid_size):
    prediction.append(num_to_label(decoded[i]))
print("No of Predictions: ",len(prediction))
```

```
No of Predictions:  3000
```

[77]:
```
#To Find Accuracy of Predicted Characters and Predicted Word
y_true = valid.loc[0:valid_size-1, 'IDENTITY']
correct_char = 0
total_char = 0
correct = 0
y_true_char = []
y_pred_char = []
```

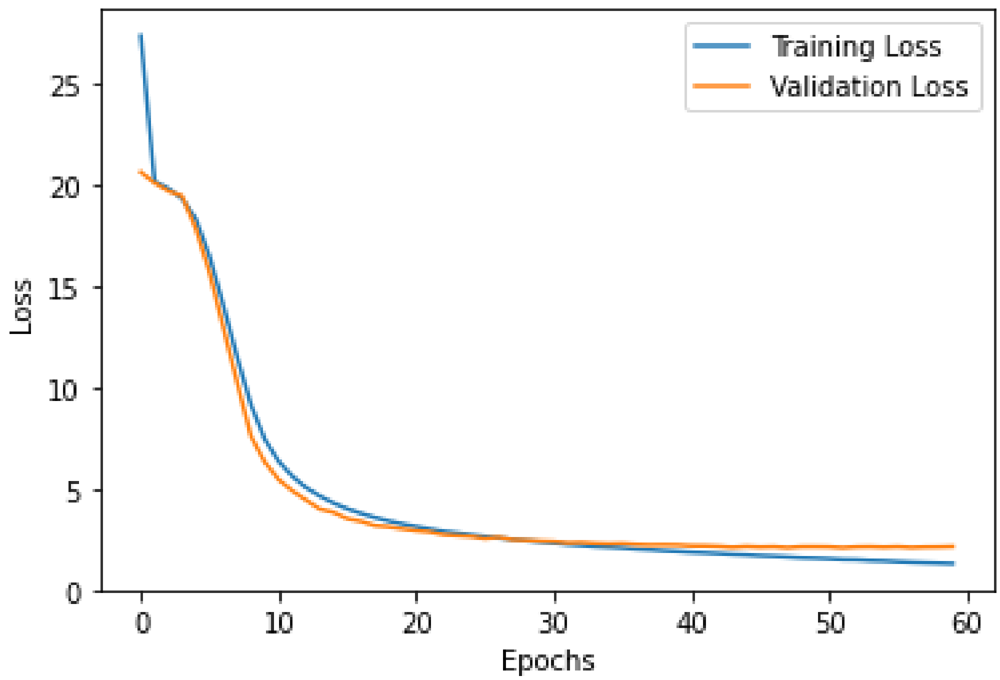
```python
for i in range(len(y_true)):
    pr = prediction[i]
    tr = y_true[i]

    for j in range(min(len(tr), len(pr))):
        if tr[j] == pr[j]:
            correct_char += 1
        y_true_char.append(tr[j])
        y_pred_char.append(pr[j])

    if pr == tr :
        correct += 1
print("No of Images " + str(len(y_true)))
print("No of Characters over all Images " + str(len(y_true_char)))
print('Correct characters predicted : %.2f%%' %(correct_char*100/
 ↪len(y_true_char)))
print('Correct words predicted      : %.2f%%' %(correct*100/len(y_true)))
```

```
No of Images 3000
No of Characters over all Images 19410
Correct characters predicted : 92.51%
Correct words predicted      : 75.17%
```

[33]:

```python
#print((history.history))
print("Loss Over training: Model performance")
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(["Training Loss","Validation Loss"])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

```
Loss Over training: Model performance
```

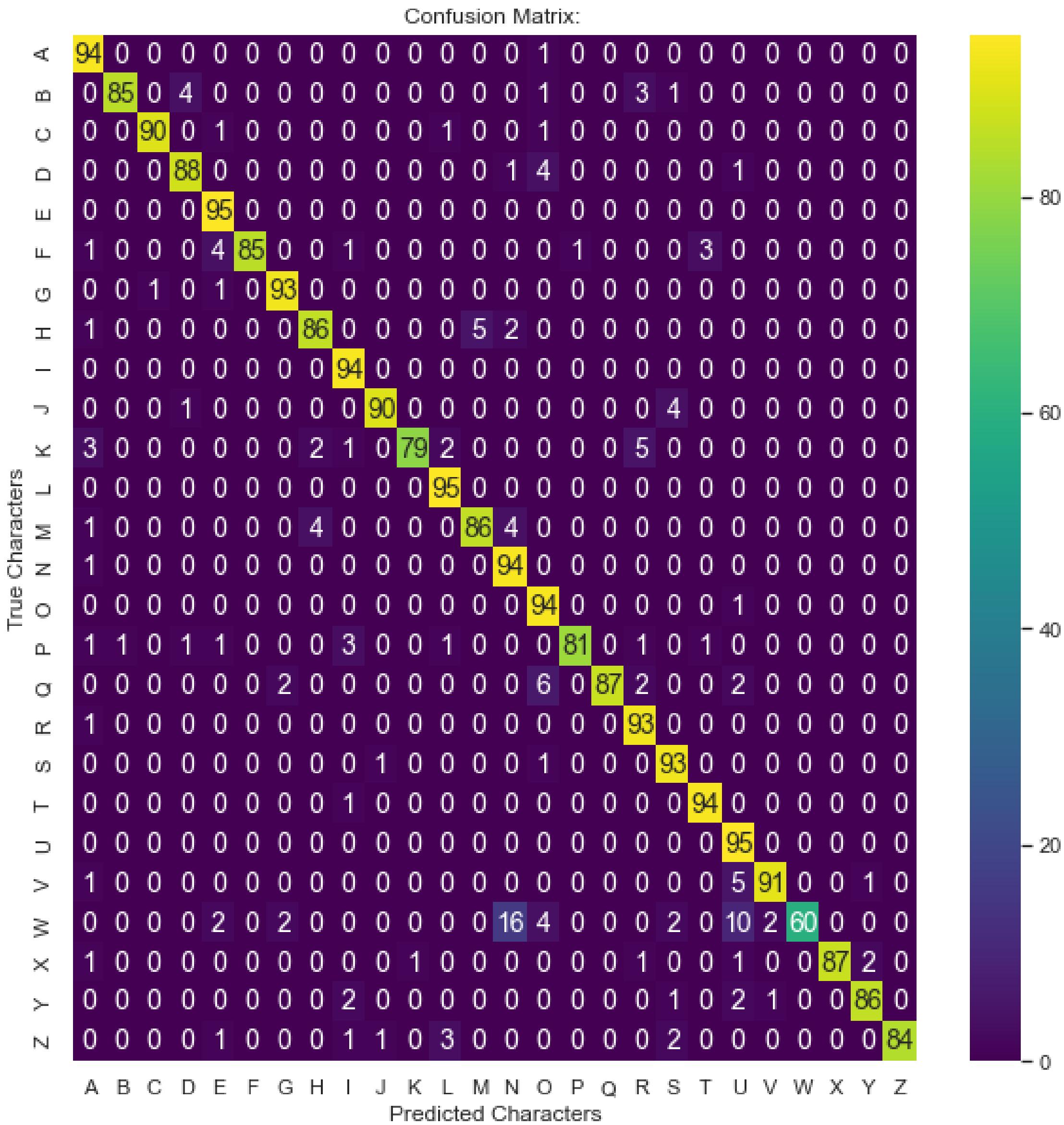


[80]:
```
#Classification Report and Confusion Matrix
print("Classification Report")
print(classification_report(y_true_char,y_pred_char))


cm = (confusion_matrix(y_true_char,y_pred_char,labels =␣
 ↪list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),normalize="true"))*100
df_cm = pd.DataFrame(cm,index = list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),columns =␣
 ↪list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
df_cm = df_cm.astype("int32")
sn.set(font_scale=1,rc={'figure.figsize':(10,10)})
sn.heatmap(df_cm,annot=True, annot_kws={"size": 14},cmap='viridis')
plt.title("Confusion Matrix: ")
plt.ylabel("True Characters")
plt.xlabel("Predicted Characters")
plt.show()


mod = "CRNNPredicted"
print("Mean Squared Error: ",mod," ",mean_squared_error([ord(i) for i in␣
 ↪y_true_char],[ord(i) for i in y_pred_char]))


clean_result = valid.loc[0:valid_size-1, 'IDENTITY']
```

```
validation_df = pd.Series(prediction,name = mod)
# Create 1 dataframe with both actual and OCR labels
ocr_vs_actual = pd.
 ↪merge(validation_df,clean_result,right_index=True,left_index=True)

# Remove labels which do not exist
ocr_vs_actual = ocr_vs_actual.loc[ocr_vs_actual[mod].notnull(), :]

# Remove spaces in OCR output
ocr_vs_actual['IDENTITY'] = ocr_vs_actual['IDENTITY'].str.replace('\\s', '',␣
 ↪regex=True)
#ocr_vs_actual.head(10)

# Create jaro-winkler similarity score
vectorized_jaro_winkler = np.vectorize(jaro_winkler)

ocr_vs_actual['SIMILARITY_SCORE'] = vectorized_jaro_winkler(ocr_vs_actual[mod].
 ↪str.upper(), np.where(ocr_vs_actual['IDENTITY'].isnull(), '', ␣
 ↪ocr_vs_actual['IDENTITY'].str.upper()))
print("Similarity Score between True Label and Predicted Label: " + mod)
display(ocr_vs_actual.head(10))



# Plot histogram of similarity scores to see how well we did
print("Histogram of Similarity: " + mod)
plt.style.use('seaborn-white')
plt.figure(figsize=(8,3), dpi=120)
plt.hist(ocr_vs_actual['SIMILARITY_SCORE'], bins=50, alpha=0.5,␣
 ↪color='steelblue', edgecolor='none')
plt.title('Histogram of Jaro-Winkler similarity score between label and␣
 ↪OCR-results: '+ mod)
plt.xlabel("Similarity")
plt.ylabel("No of Images")
plt.xticks(np.arange(0, 1.1,0.1))
plt.yticks(np.arange(0,len(ocr_vs_actual)+1,len(ocr_vs_actual)/6))
plt.show()
```

```
Classification Report
              precision    recall  f1-score   support

                   0.76      0.67      0.71        84
           '       0.00      0.00      0.00         2
           -       0.66      0.75      0.70        51
           A       0.94      0.94      0.94      2433
           B       0.94      0.85      0.90       446
           C       0.92      0.91      0.91       623
```

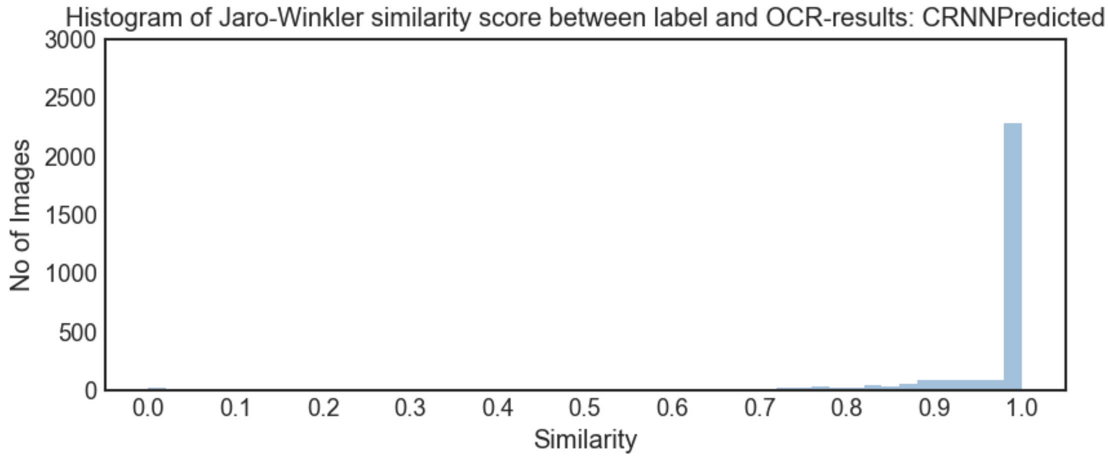
D	0.92	0.88	0.90	588
E	0.95	0.95	0.95	2399
F	0.94	0.86	0.90	166
G	0.90	0.94	0.92	362
H	0.88	0.86	0.87	513
I	0.94	0.95	0.94	1587
J	0.86	0.90	0.88	140
K	0.85	0.80	0.82	119
L	0.95	0.95	0.95	1439
M	0.90	0.86	0.88	740
N	0.93	0.94	0.93	1506
O	0.90	0.94	0.92	1240
P	0.88	0.82	0.85	251
Q	0.88	0.88	0.88	48
R	0.93	0.94	0.93	1314
S	0.94	0.93	0.93	805
T	0.94	0.94	0.94	914
U	0.89	0.95	0.92	869
V	0.93	0.91	0.92	225
W	0.76	0.60	0.67	48
X	0.96	0.87	0.92	118
Y	0.88	0.87	0.87	267
Z	0.90	0.85	0.87	113
accuracy			0.93	19410
macro avg	0.86	0.84	0.85	19410
weighted avg	0.93	0.93	0.92	19410




```
Mean Squared Error:  CRNNPredicted   11.183153013910356
Similarity Score between True Label and Predicted Label: CRNNPredicted

  CRNNPredicted      IDENTITY  SIMILARITY_SCORE
0         BILEL         BILEL          1.000000
1     LAUMONIER    LAUMIONIER          0.946667
2           LEA           LEA          1.000000
3     JEAN-ROCH     JEAN-ROCH          1.000000
4          RUPP          RUPP          1.000000
5        PICHON        PICHON          1.000000
6        DANIEL        DANIEL          1.000000
7        JEREMY        JEREMY          1.000000
```

```
8   JEAN-MICHEL  JEAN-MICHEL          1.000000
9        JULIEN       JULIEN          1.000000
```

```
Histogram of Similarity: CRNNPredicted
```



[81]:
```python
#To test sample Images. To view Sample Images along with predicted Label
#test = pd.read_csv("HandwrittenDataset/written_name_train_v2.csv")
test = pd.read_csv('HandwrittenDataset/written_name_test_v2.csv')

plt.figure(figsize=(15, 10))
start_img_ind = 3000
for i in range(start_img_ind,start_img_ind+12):
    ax = plt.subplot(4, 3, i+1-start_img_ind)
    img_dir = 'HandwrittenDataset/test_v2/test/'+test.loc[i, 'FILENAME']
    #img_dir = 'HandwrittenDataset/train_v2/train/'+test.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap='gray')

    image = preprocess(image)
    image = image/255.
    pred = model.predict(image.reshape(1, 256, 64, 1))
    decoded = K.get_value(K.ctc_decode(pred, input_length=np.ones(pred.
 ↪shape[0])*pred.shape[1], greedy=True)[0][0])
    plt.title(num_to_label(decoded[0]), fontsize=12)

    plt.axis('off')

plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```

[7]:
```python
#To Load the saved Model
CNNmodel = tf.keras.models.load_model("CNNModel")
print("CNN Model")
print(CNNmodel.summary())
RNNmodel = tf.keras.models.load_model("RNNModel4")
print("RNN Model")
print(RNNmodel.summary())
```

```
CNN Model
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 256, 64, 1)]      0
_________________________________________________________________
conv1 (Conv2D)               (None, 256, 64, 32)       320
_________________________________________________________________
batch_normalization_3 (Batch (None, 256, 64, 32)       128
_________________________________________________________________
activation_3 (Activation)    (None, 256, 64, 32)       0
_________________________________________________________________
max1 (MaxPooling2D)          (None, 128, 32, 32)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 128, 32, 64)       18496
_________________________________________________________________
batch_normalization_4 (Batch (None, 128, 32, 64)       256
_________________________________________________________________
activation_4 (Activation)    (None, 128, 32, 64)       0
_________________________________________________________________
max2 (MaxPooling2D)          (None, 64, 16, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 16, 64)        0
_________________________________________________________________
conv3 (Conv2D)               (None, 64, 16, 128)       73856
_________________________________________________________________
```

```
batch_normalization_5 (Batch (None, 64, 16, 128)       512
_________________________________________________________________
activation_5 (Activation)    (None, 64, 16, 128)       0
_________________________________________________________________
max3 (MaxPooling2D)          (None, 64, 8, 128)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64, 8, 128)        0
_________________________________________________________________
reshape (Reshape)            (None, 64, 1024)          0
_________________________________________________________________
dense2 (Dense)               (None, 64, 30)            30750
_________________________________________________________________
softmax (Activation)         (None, 64, 30)            0
=================================================================
Total params: 124,318
Trainable params: 123,870
Non-trainable params: 448
_________________________________________________________________
None
RNN Model
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 16384)]           0
_________________________________________________________________
reshape_2 (Reshape)          (None, 64, 256)           0
_________________________________________________________________
dense_3 (Dense)              (None, 64, 64)            16448
_________________________________________________________________
lstm_4 (LSTM)                (None, 64, 256)           328704
_________________________________________________________________
lstm_5 (LSTM)                (None, 64, 256)           525312
_________________________________________________________________
dense_4 (Dense)              (None, 64, 30)            7710
_________________________________________________________________
activation_2 (Activation)    (None, 64, 30)            0
=================================================================
Total params: 878,174
Trainable params: 878,174
Non-trainable params: 0
_________________________________________________________________
None
```

[82]:

```
valid = pd.read_csv("TrainTable.csv")
#display(valid.describe(include="all"))
valid.fillna('', inplace=True)
```

```
display(valid.head())
display(valid.tail())
print("No of Images Tested with ", len(valid))
```

```
   Index          FILENAME   IDENTITY CRNNPredicted CNNPredicted RNNPredicted
0      0  TRAIN_00001.jpg  BALTHAZAR     BALTHAZAR    BALTHAZAR   BLALTHAIAR
1      1  TRAIN_00002.jpg      SIMON         SIMON        SIMON       BLIMON
2      2  TRAIN_00003.jpg      BENES         BENES        BENES       BVENES
3      3  TRAIN_00004.jpg    LA LOVE        LALOUE       LALOUE      BLALOUE
4      4  TRAIN_00005.jpg     DAPHNE        DAPHNE       DAPHNE      CMAPHNE
```

```
         Index          FILENAME       IDENTITY  CRNNPredicted  CNNPredicted  \
330287  330287  TRAIN_330957.jpg          LENNY          LENNY         LENNY
330288  330288  TRAIN_330958.jpg        TIFFANY        TIFFANY       TIEEANY
330289  330289  TRAIN_330959.jpg  COUTINHO DESA  COUTINHO DESA  COUTINHODESA
330290  330290  TRAIN_330960.jpg         MOURAD         MOURAD        AOURAD
330291  330291  TRAIN_330961.jpg        HELOISE        HELOISE       HELOISE

        RNNPredicted
330287        BLENNY
330288      BLIEFANY
330289  BCOUTINO DEA
330290       BAOURAD
330291      BLELOISE
```

```
No of Images Tested with  330292
```

[84]:

```
col_names = ["CRNNPredicted","CNNPredicted","RNNPredicted"]
hsh_class = {}
for mod in col_names:
    print("Prediction Result for ",mod)
    prediction = valid[mod]
    y_true = valid['IDENTITY']
    correct_char = 0
    total_char = 0
    correct = 0
    y_true_char = []
    y_pred_char = []
    for i in range(len(y_true)):
        pr = (prediction[i])
        tr = y_true[i]
        total_char += min(len(tr), len(pr))
        #print(tr,pr)
        for j in range(min(len(tr), len(pr))):
            if mod == "RNNPredicted":
                if tr[-j] == pr[-j]:
                    correct_char += 1
                y_true_char.append(tr[-j])
```

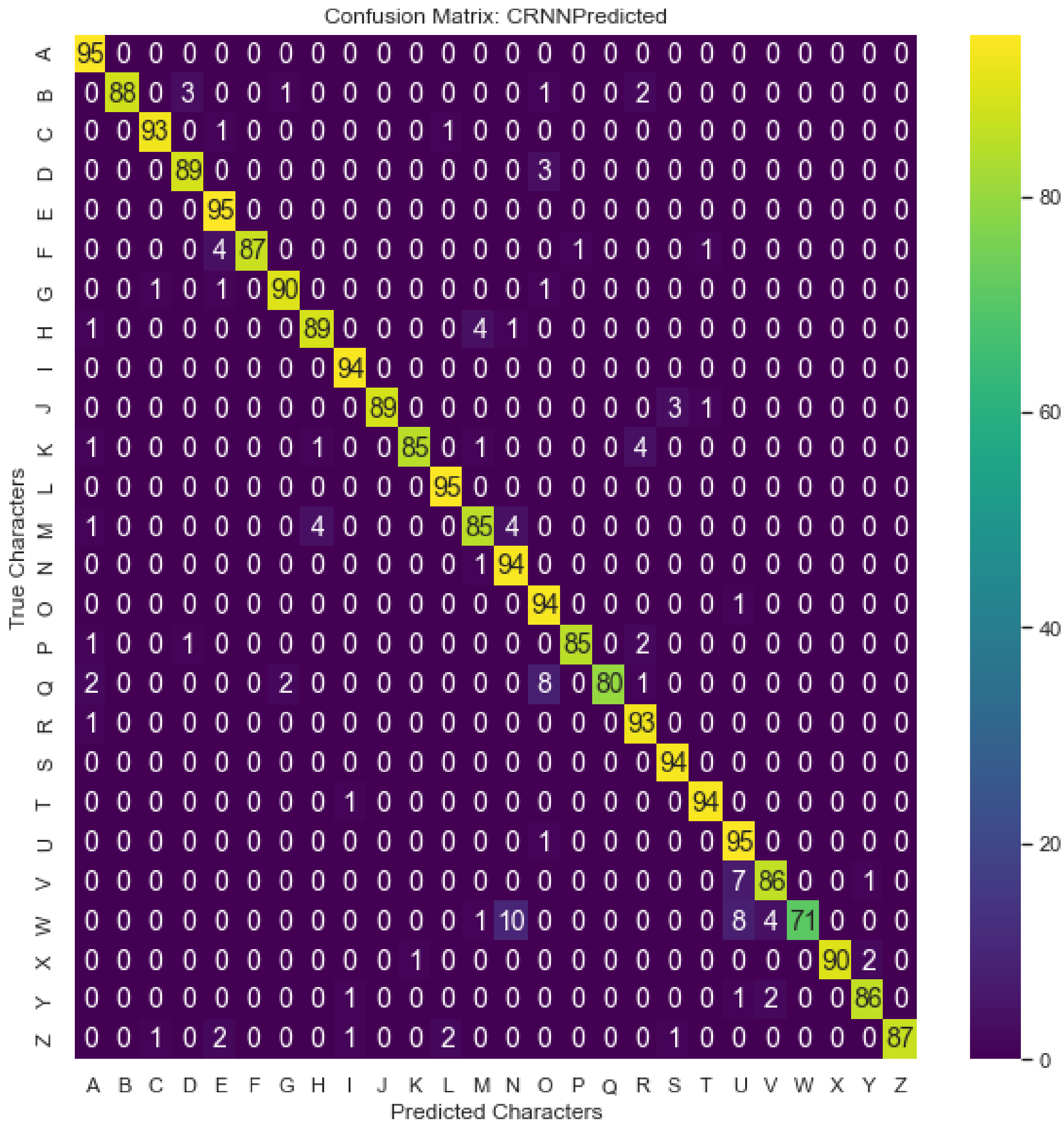
```
                y_pred_char.append(pr[-j])
            else:
                if tr[j] == pr[j]:
                    correct_char += 1
                y_true_char.append(tr[j])
                y_pred_char.append(pr[j])

        if pr == tr :
            correct += 1

    print("No of Images " + str(len(y_true)))
    print("No of Characters over all Images " + str(len(y_true_char)))
    print('Correct characters predicted : %.2f%%' %(correct_char*100/
↪len(y_true_char)))
    print('Correct words predicted      : %.2f%%' %(correct*100/len(y_true)))
    print("Classification Report: " + mod)
    print(classification_report(y_true_char,y_pred_char,zero_division=0))
    hsh_class[mod] =␣
↪classification_report(y_true_char,y_pred_char,zero_division=0,output_dict =␣
↪True)
    print("Confusion Matrix: " + mod)
    cm = (confusion_matrix(y_true_char,y_pred_char,labels =␣
↪list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),normalize="true"))*100
    df_cm = pd.DataFrame(cm,index = list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),columns␣
↪= list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
    df_cm = df_cm.astype("int32")
    sn.set(font_scale=1,rc={'figure.figsize':(10,10)})
    sn.heatmap(df_cm,annot=True, annot_kws={"size": 14},cmap='viridis')
    plt.title("Confusion Matrix: " + mod)
    plt.ylabel("True Characters")
    plt.xlabel("Predicted Characters")
    plt.show()

    print("Mean Squared Error: ",mod," ",mean_squared_error([ord(i) for i in␣
↪y_true_char],[ord(i) for i in y_pred_char]))


    clean_result = valid["IDENTITY"]
    validation_df = pd.Series(prediction,name = mod)
    # Create 1 dataframe with both actual and OCR labels
    ocr_vs_actual = pd.
↪merge(validation_df,clean_result,right_index=True,left_index=True)

    # Remove labels which do not exist
    ocr_vs_actual = ocr_vs_actual.loc[ocr_vs_actual[mod].notnull(), :]
```

```
    # Remove spaces in OCR output
    ocr_vs_actual['IDENTITY'] = ocr_vs_actual['IDENTITY'].str.replace('\\s',␣
↪'', regex=True)
    #ocr_vs_actual.head(10)

    # Create jaro-winkler similarity score
    vectorized_jaro_winkler = np.vectorize(jaro_winkler)

    ocr_vs_actual['SIMILARITY_SCORE'] =␣
↪vectorized_jaro_winkler(ocr_vs_actual[mod].str.upper(), np.
↪where(ocr_vs_actual['IDENTITY'].isnull(), '',  ocr_vs_actual['IDENTITY'].str.
↪upper()))
    print("Similarity Score between True Label and Predicted Label: " + mod)
    display(ocr_vs_actual.head(10))



    # Plot histogram of similarity scores to see how well we did
    print("Histogram of Similarity: " + mod)
    plt.style.use('seaborn-white')
    plt.figure(figsize=(8,3), dpi=120)
    plt.hist(ocr_vs_actual['SIMILARITY_SCORE'], bins=50, alpha=0.5,␣
↪color='steelblue', edgecolor='none')
    plt.title('Histogram of Jaro-Winkler similarity score: '+ mod)
    plt.xlabel("Similarity")
    plt.ylabel("No of Images")
    plt.xticks(np.arange(0, 1.1,0.1))
    #plt.yticks(np.arange(0,len(ocr_vs_actual)+1,len(ocr_vs_actual)/6))
    plt.show()
```

```
Prediction Result for  CRNNPredicted
No of Images 330292
No of Characters over all Images 2134747
Correct characters predicted : 92.98%
Correct words predicted      : 75.56%
Classification Report: CRNNPredicted
              precision    recall  f1-score   support

                   0.76      0.63      0.69     10100
           #       0.00      0.00      0.00         1
           '       0.61      0.22      0.32       230
           -       0.66      0.72      0.69      6493
           ?       0.00      0.00      0.00         0
           A       0.95      0.95      0.95    266965
           B       0.93      0.88      0.91     45634
           C       0.91      0.93      0.92     67580
           D       0.91      0.89      0.90     59587
```

E	0.95	0.96	0.95	266361
F	0.93	0.88	0.90	18135
G	0.90	0.90	0.90	38807
H	0.90	0.89	0.89	61232
I	0.95	0.95	0.95	171683
J	0.90	0.89	0.90	15745
K	0.91	0.86	0.88	13880
L	0.95	0.95	0.95	159677
M	0.90	0.86	0.88	83273
N	0.93	0.94	0.94	164645
O	0.92	0.94	0.93	134912
P	0.90	0.86	0.88	29106
Q	0.88	0.80	0.84	5131
R	0.94	0.93	0.94	146178
S	0.95	0.95	0.95	89762
T	0.93	0.94	0.94	99376
U	0.90	0.95	0.93	95253
V	0.91	0.87	0.89	24865
W	0.85	0.71	0.78	5464
X	0.94	0.90	0.92	11116
Y	0.88	0.86	0.87	29630
Z	0.92	0.88	0.90	13925
`	0.00	0.00	0.00	1
accuracy			0.93	2134747
macro avg	0.81	0.78	0.79	2134747
weighted avg	0.93	0.93	0.93	2134747

Confusion Matrix: CRNNPredicted

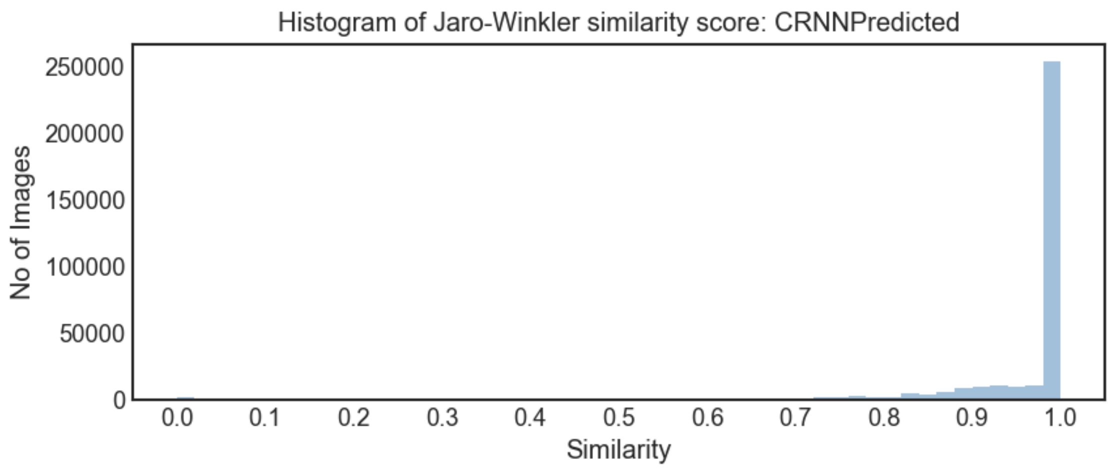


Mean Squared Error: CRNNPredicted 10.623822869876383
Similarity Score between True Label and Predicted Label: CRNNPredicted

	CRNNPredicted	IDENTITY	SIMILARITY_SCORE
0	BALTHAZAR	BALTHAZAR	1.000000
1	SIMON	SIMON	1.000000
2	BENES	BENES	1.000000
3	LALOUE	LALOVE	0.933333
4	DAPHNE	DAPHNE	1.000000
5	LUCIE	LUCIE	1.000000
6	NASSIM	NASSIM	1.000000
7	ASSRAOUI	ASSRAOUI	1.000000

```
8       VLAVIAN     LAVIAN          0.869048
9         MAEVA      MAEVA          1.000000
```

Histogram of Similarity: CRNNPredicted

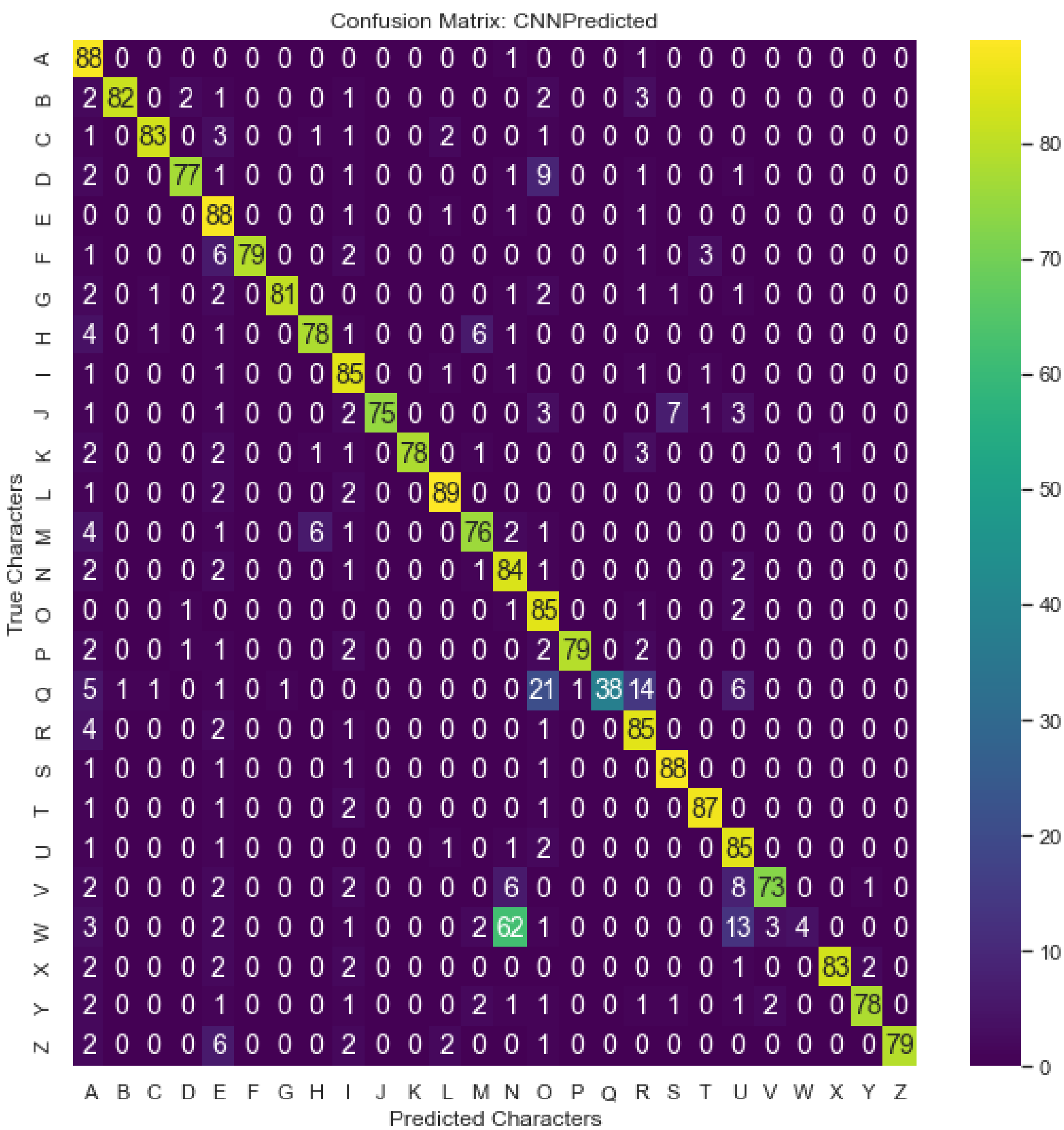



```
Prediction Result for  CNNPredicted
No of Images 330292
No of Characters over all Images 2090486
Correct characters predicted : 84.25%
Correct words predicted      : 59.54%
Classification Report: CNNPredicted
```

	precision	recall	f1-score	support
	0.01	0.00	0.00	9874
#	0.00	0.00	0.00	1
'	0.00	0.00	0.00	217
-	0.58	0.51	0.55	6406
A	0.86	0.88	0.87	262418
B	0.85	0.82	0.83	45311
C	0.83	0.84	0.84	66669
D	0.83	0.77	0.80	58235
E	0.87	0.88	0.88	258264
F	0.82	0.79	0.81	17890
G	0.85	0.81	0.83	38457
H	0.77	0.79	0.78	60455
I	0.84	0.86	0.85	168646
J	0.86	0.75	0.80	15684
K	0.84	0.78	0.81	13612
L	0.89	0.89	0.89	157627
M	0.80	0.77	0.78	81691
N	0.84	0.85	0.84	159273
O	0.80	0.86	0.83	132511

P	0.84	0.80	0.82	28657
Q	0.77	0.39	0.51	5065
R	0.84	0.85	0.85	143343
S	0.87	0.88	0.87	86890
T	0.88	0.88	0.88	96796
U	0.79	0.86	0.82	93805
V	0.79	0.73	0.76	24637
W	0.57	0.04	0.08	5405
X	0.84	0.83	0.84	10624
Y	0.84	0.78	0.81	28490
Z	0.85	0.79	0.82	13532
`	0.00	0.00	0.00	1
accuracy			0.84	2090486
macro avg	0.71	0.67	0.68	2090486
weighted avg	0.84	0.84	0.84	2090486

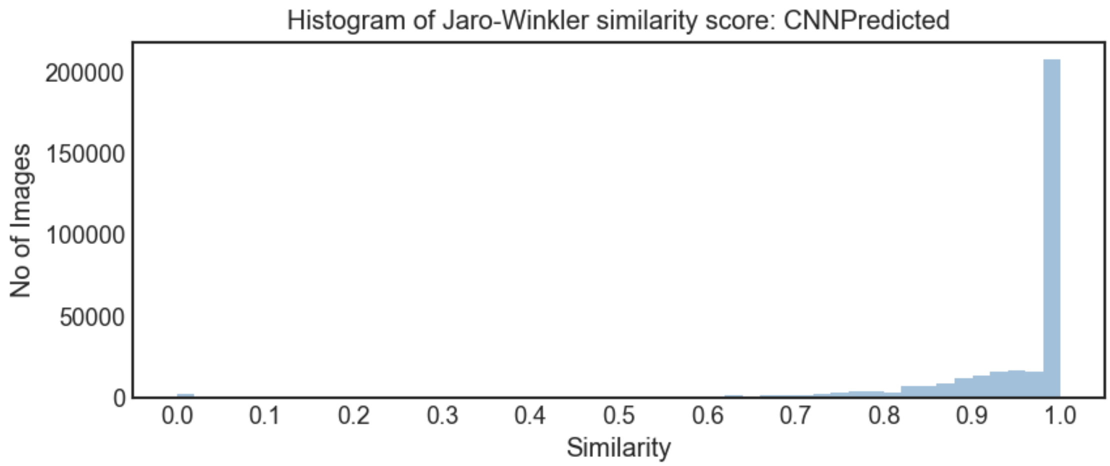
Confusion Matrix: CNNPredicted




```
Mean Squared Error:  CNNPredicted   25.33519956603393
Similarity Score between True Label and Predicted Label: CNNPredicted

  CNNPredicted   IDENTITY  SIMILARITY_SCORE
0    BALTHAZAR  BALTHAZAR          1.000000
1        SIMON      SIMON          1.000000
2        BENES      BENES          1.000000
3       LALOUE     LALOVE          0.933333
4       DAPHNE     DAPHNE          1.000000
5        LUCIE      LUCIE          1.000000
6       NASSIM     NASSIM          1.000000
7     ASSRAOUI   ASSRAOUI          1.000000
```

8	MLAVIAN	LAVIAN	0.952381
9	MAEVA	MAEVA	1.000000

Histogram of Similarity: CNNPredicted

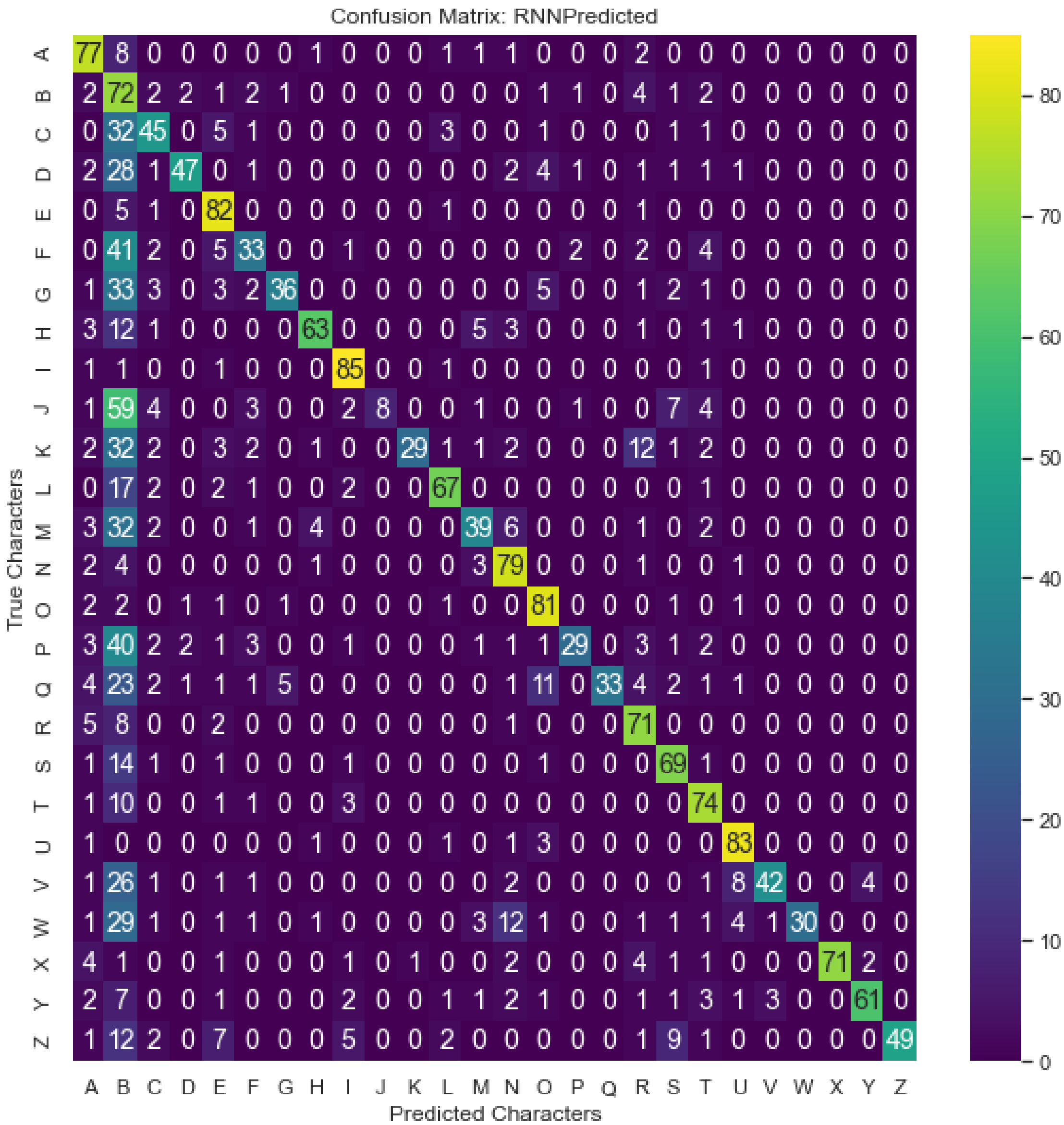


Prediction Result for RNNPredicted
No of Images 330292
No of Characters over all Images 2144334
Correct characters predicted : 69.81%
Correct words predicted : 0.26%
Classification Report: RNNPredicted

	precision	recall	f1-score	support
	0.56	0.42	0.48	9877
#	0.00	0.00	0.00	1
'	0.45	0.10	0.16	216
-	0.40	0.44	0.42	6474
?	0.00	0.00	0.00	1
A	0.84	0.77	0.81	266860
B	0.12	0.72	0.20	45613
C	0.52	0.45	0.48	67799
D	0.73	0.48	0.58	60078
E	0.87	0.82	0.85	268372
F	0.24	0.33	0.28	18198
G	0.64	0.37	0.47	38919
H	0.73	0.63	0.67	61191
I	0.87	0.85	0.86	172547
J	0.27	0.08	0.13	15759
K	0.59	0.30	0.39	14022
L	0.83	0.68	0.74	160140
M	0.60	0.39	0.47	82771
N	0.84	0.79	0.81	166194

O	0.83	0.81	0.82	134731
P	0.47	0.29	0.36	29025
Q	0.56	0.33	0.41	5134
R	0.79	0.72	0.75	146995
S	0.78	0.69	0.73	90876
T	0.75	0.74	0.74	100754
U	0.83	0.83	0.83	95309
V	0.56	0.42	0.48	24921
W	0.57	0.31	0.40	5481
X	0.82	0.71	0.76	11300
Y	0.78	0.61	0.69	30703
Z	0.69	0.50	0.58	14072
`	0.00	0.00	0.00	1
accuracy			0.70	2144334
macro avg	0.58	0.49	0.51	2144334
weighted avg	0.76	0.70	0.72	2144334

Confusion Matrix: RNNPredicted



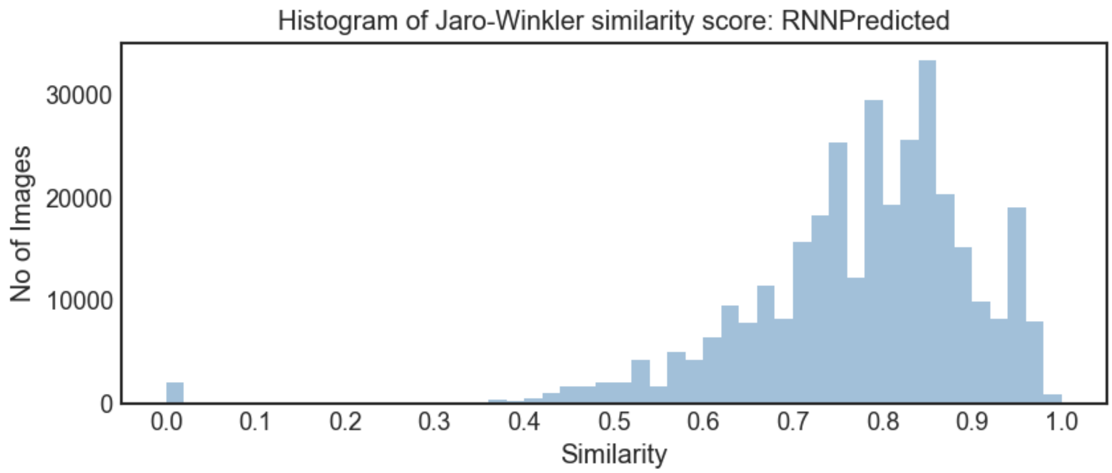
Mean Squared Error: RNNPredicted   41.535935166816365

Similarity Score between True Label and Predicted Label: RNNPredicted

	RNNPredicted	IDENTITY	SIMILARITY_SCORE
0	BLALTHAIAR	BALTHAZAR	0.869167
1	BLIMON	SIMON	0.822222
2	BVENES	BENES	0.950000
3	BLALOUE	LALOVE	0.849206
4	CMAPHNE	DAPHNE	0.849206
5	BLUCIE	LUCIE	0.944444
6	BCMASSIM	NASSIM	0.652778
7	TFBSSRAOUI	ASSRAOUI	0.858333

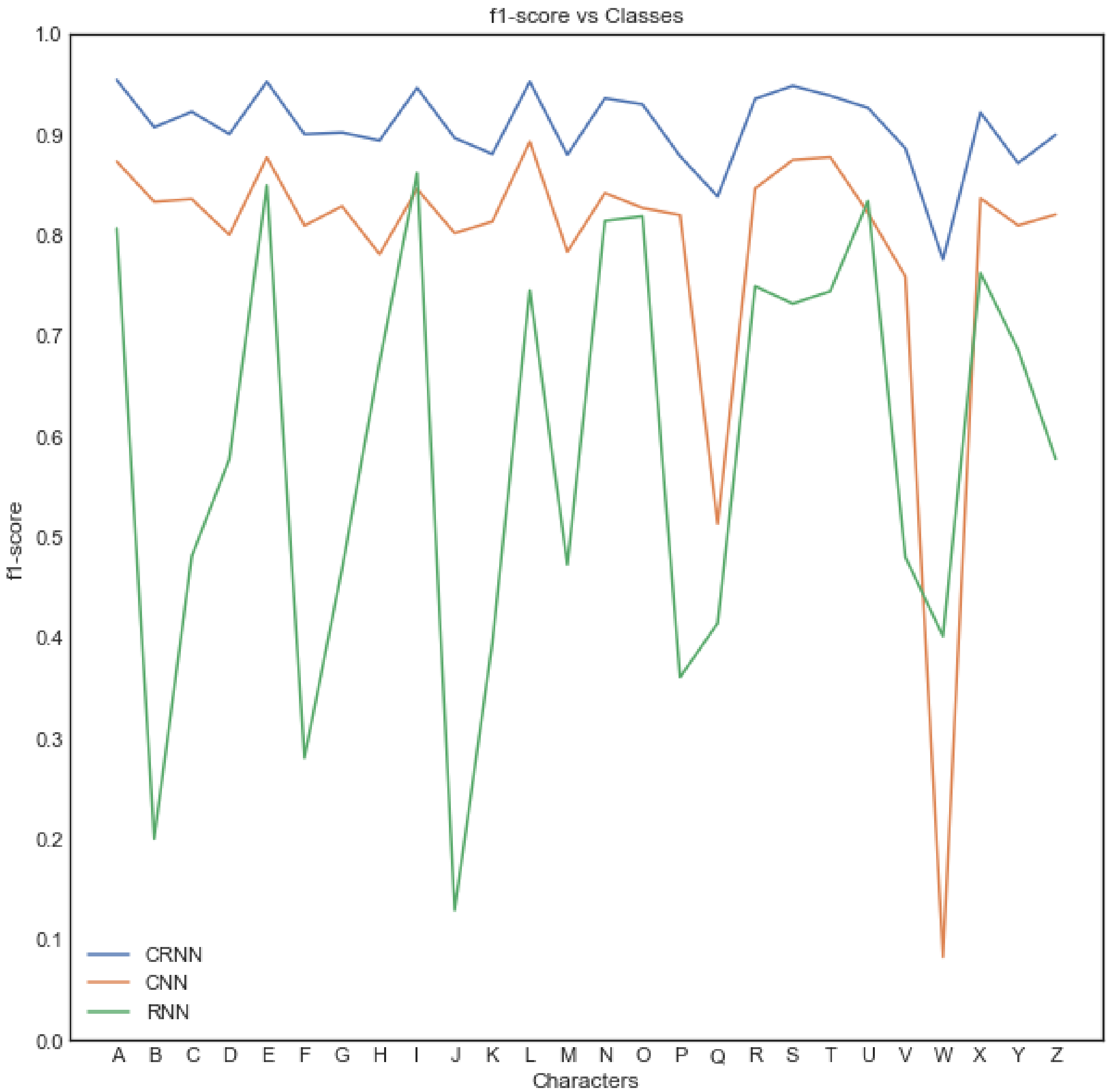
```
8      BALAVRAN      LAVIAN          0.686111
9        BLAEVA       MAEVA          0.822222
```

Histogram of Similarity: RNNPredicted



[68]:

```
fig,ax = plt.subplots()

ax.
 ↪plot(list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),[hsh_class["CRNNPredicted"][i]["f1-score"]␣
 ↪for i in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"])

ax.
 ↪plot(list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),[hsh_class["CNNPredicted"][i]["f1-score"]␣
 ↪for i in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"])

ax.
 ↪plot(list(u"ABCDEFGHIJKLMNOPQRSTUVWXYZ"),[hsh_class["RNNPredicted"][i]["f1-score"]␣
 ↪for i in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"])
ax.set_title('f1-score vs Classes')
ax.legend(['CRNN','CNN','RNN'])
ax.xaxis.set_label_text('Characters')
ax.yaxis.set_label_text('f1-score')
plt.yticks(np.arange(0,1.1, 0.1))
plt.show()
```


f1-score vs Classes
f1-score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
CRNN
CNN
RNN
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Characters