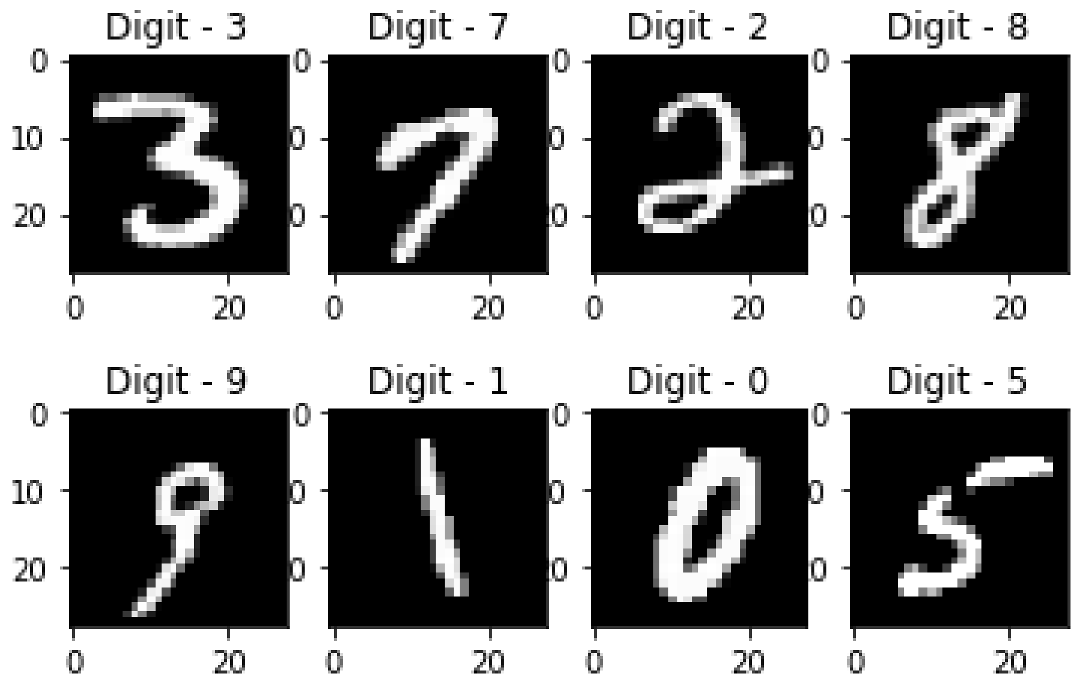
HANDWRITTEN CHARACTER RECOGNITION ✍



About the dataset

This dataset consists of more than four hundred thousand handwritten names collected through charity projects.

Character Recognition utilizes image processing technologies to convert characters on scanned documents into digital forms. It typically performs well in machine-printed fonts. However, it still poses difficult challenges for machines to recognize handwritten characters, because of the huge variation in individual writing styles.

There are 206,799 first names and 207,024 surnames in total. The data was divided into a training set (331,059), testing set (41,382), and validation set (41,382) respectively.

Data Exploration

```
import pandas as pd
import numpy as np
import keras
import keras.layers as L
import keras.models as M
import tensorflow as tf
from PIL import Image
import os
import matplotlib.pyplot as plt
import cv2
from keras.utils import Sequence
```

```
train=pd.read_csv('train.csv')
validation=pd.read_csv('validation.csv')
```

```
train.dropna(inplace=True)
```

```
train.head()
```

```
          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
```

Data Pre-Processing

```
train=train.sample(frac=0.8,random_state=42)
validation=validation.sample(frac=0.1)
```

```
characters=set()
train['IDENTITY']=train['IDENTITY'].apply(lambda x: str(x))
for i in train['IDENTITY'].values:
    for j in i :
        if j not in characters :
            characters.add(j)
characters=sorted(characters)
```

```
# 2 Dictionaries:   Turn all your characters to num and vice versa
char_to_label = {char:label for label,char in enumerate(characters)}
label_to_char = {label:char for label,char in enumerate(characters)}
```

```
path_train='/content/train'
path_validation='/content/validation'
```

```
# Data Generator
class DataGenerator(Sequence):
    def __init__(self,dataframe,path,char_map,batch_size=128,img_size=(256,64),
                 downsample_factor=4,max_length=22,shuffle=True):
        self.dataframe=dataframe
        self.path=path
        self.char_map=char_map
        self.batch_size=batch_size
        self.width=img_size[0]
        self.height=img_size[1]
        self.downsample_factor=downsample_factor
        self.max_length=max_length
        self.shuffle=shuffle
        self.indices = np.arange(len(dataframe))
        self.on_epoch_end()

    def __len__(self):
        return len(self.dataframe)//self.batch_size

    def __getitem__(self,idx):
        curr_batch_idx=self.indices[idx*self.batch_size:(idx+1)*self.batch_size]

batch_images=np.ones((self.batch_size,self.width,self.height,1),dtype=np.float32)
        batch_labels=np.ones((self.batch_size,self.max_length),dtype=np.float32)

input_length=np.ones((self.batch_size,1),dtype=np.float32)*(self.width//self.downsample_factor-2)
        label_length=np.zeros((self.batch_size,1),dtype=np.int64)
        for i,idx in enumerate(curr_batch_idx):
            img_path=self.dataframe['FILENAME'].values[idx]
            img=cv2.imread(self.path+'/'+img_path)
            img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img=cv2.resize(img,(self.width,self.height))
            img=(img/255).astype(np.float32)
            img=img.T
            img=np.expand_dims(img,axis=-1)
            text=self.dataframe['IDENTITY'].values[idx]
            text=str(text)
            label=[]
            for j in text:
                if j in self.char_map :
                    label.append(self.char_map[j])
                else:
                    label.append(100)
            label.extend([100]*(22-len(label)))
            batch_images[i]=img
            batch_labels[i]=label
            label_length[i]=len(label)
        batch_inputs= {
                'input_data':batch_images,
                'input_label':batch_labels,
                'input_length':input_length,
                'label_length':label_length

            }
        return batch_inputs,np.zeros((self.batch_size),dtype=np.float32)
    def on_epoch_end(self):
        if self.shuffle == True :
            np.random.shuffle(self.indices)
```

```
train_generator=DataGenerator(train,path_train,char_to_label)
validation_generator=DataGenerator(validation,path_validation,char_to_label)


# Making CTC Function
class CTCLayer(L.Layer):
    def __init__(self, name=None):
        super().__init__(name=name)
        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred, input_length, label_length):
        # Compute the training-time loss value and add it
        # to the layer using `self.add_loss()`.
        loss = self.loss_fn(y_true, y_pred, input_length, label_length)
        self.add_loss(loss)

        # On test time, just return the computed loss
        return loss
```

Model Creation

```
# Making the Model
def make_model():
    inp=L.Input(shape=(256,64,1),dtype=np.float32,name='input_data')
    labels=L.Input(shape=[22],dtype=np.float32,name='input_label')
    input_length=L.Input(shape=[1],dtype=np.int64,name='input_length')
    label_length=L.Input(shape=[1],dtype=np.int64,name='label_length')

x=L.Conv2D(64,(3,3),activation='relu',padding='same',kernel_initializer='he_normal')(i
np)
    x=L.MaxPooling2D(pool_size=(2,2))(x)
    x=L.Dropout(0.3)(x)

x=L.Conv2D(128,(3,3),activation='relu',padding='same',kernel_initializer='he_normal')(
x)
    x=L.MaxPooling2D(pool_size=(2,2))(x)
    x=L.Dropout(0.3)(x)
    new_shape=((256//4),(64//4)*128)
    x=L.Reshape(new_shape)(x)
    x=L.Dense(64,activation='relu')(x)
    x=L.Dropout(0.2)(x)
    x=L.Bidirectional(L.LSTM(128,return_sequences=True,dropout=0.2))(x)
    x=L.Bidirectional(L.LSTM(64,return_sequences=True,dropout=0.25))(x)

x=L.Dense(len(characters)+1,activation='softmax',kernel_initializer='he_normal',name='
Dense_output')(x)
    output=CTCLayer(name='outputs')(labels,x,input_length,label_length)
    model=M.Model([inp,labels,input_length,label_length],output)
    # Optimizer
    sgd = keras.optimizers.SGD(learning_rate=0.002, decay=1e-6, momentum=0.9,
                               nesterov=True,
                               clipnorm=5)
    model.compile(optimizer=sgd)
    return model
```

```
model=make_model()
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_data (InputLayer)         [(None, 256, 64, 1)] 0
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 256, 64, 64)  640         input_data[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 128, 32, 64)  0           conv2d_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 128, 32, 64)  0           max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 128, 32, 128) 73856       dropout_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 64, 16, 128)  0           conv2d_3[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 64, 16, 128)  0           max_pooling2d_3[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 64, 2048)     0           dropout_4[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64, 64)       131136      reshape_1[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 64, 64)       0           dense_1[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 64, 256)      197632      dropout_5[0][0]
__________________________________________________________________________________________________
bidirectional_3 (Bidirectional) (None, 64, 128)      164352      bidirectional_2[0][0]
__________________________________________________________________________________________________
input_label (InputLayer)        [(None, 22)]         0
__________________________________________________________________________________________________
Dense_output (Dense)            (None, 64, 31)       3999        bidirectional_3[0][0]
__________________________________________________________________________________________________
input_length (InputLayer)       [(None, 1)]          0
__________________________________________________________________________________________________
label_length (InputLayer)       [(None, 1)]          0
__________________________________________________________________________________________________
outputs (CTCLayer)              (None, 1)            0           input_label[0][0]
                                                                 Dense_output[0][0]
                                                                 input_length[0][0]
                                                                 label_length[0][0]
==================================================================================================
Total params: 571,615
Trainable params: 571,615
Non-trainable params: 0
__________________________________________________________________________________________________
```

Model Training

```python
# Add early stopping
es = keras.callbacks.EarlyStopping(monitor='val_loss',
                                   patience=5,
                                   restore_best_weights=True)

# Train the model
if 'prediction_model_ocr.h5' not in os.listdir('./'):
    history =
model.fit(train_generator,steps_per_epoch=1000,validation_data=validation_generator,
                        epochs=6)
```

```
Epoch 1/6
1000/1000 [==============================] - 1435s 1s/step - loss: 23.9950 - val_loss:
18.9719
Epoch 2/6
1000/1000 [==============================] - 840s 840ms/step - loss: 18.5256 -
val_loss: 16.3491
Epoch 3/6
1000/1000 [==============================] - 578s 578ms/step - loss: 15.5241 -
val_loss: 9.9704
Epoch 4/6
1000/1000 [==============================] - 462s 462ms/step - loss: 9.9157 -
val_loss: 5.5094
Epoch 5/6
1000/1000 [==============================] - 332s 332ms/step - loss: 6.3807 -
val_loss: 4.0310
Epoch 6/6
1000/1000 [==============================] - 318s 318ms/step - loss: 4.8064 -
val_loss: 3.0615
```

```python
prediction_model = keras.models.Model(model.get_layer(name='input_data').input,
                                        model.get_layer(name='Dense_output').output)
prediction_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_data (InputLayer)      [(None, 256, 64, 1)]      0
_________________________________________________________________
conv2d (Conv2D)              (None, 256, 64, 64)       640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 32, 64)       0
_________________________________________________________________
dropout (Dropout)            (None, 128, 32, 64)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 32, 128)      73856
```

```
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 16, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 16, 128)       0
_________________________________________________________________
reshape (Reshape)            (None, 64, 2048)          0
_________________________________________________________________
dense (Dense)                (None, 64, 64)            131136
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 64)            0
_________________________________________________________________
bidirectional (Bidirectional (None, 64, 256)           197632
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64, 128)           164352
_________________________________________________________________
Dense_output (Dense)         (None, 64, 31)            3999
=================================================================
Total params: 571,615
Trainable params: 571,615
Non-trainable params: 0
_________________________________________________________________
```

```python
if 'prediction_model_ocr.h5' not in os.listdir('./'):
    prediction_model.save('prediction_model_ocr.h5')
    prediction_model=M.load_model('prediction_model_ocr.h5')
```

```python
label_to_char[100]=''
```

```python
# A utility to decode the output of the network
def decode_batch_predictions(pred):
    pred = pred[:, :-2]
    input_len = np.ones(pred.shape[0])*pred.shape[1]

    # Use greedy search. For complex tasks, you can use beam search
    results = keras.backend.ctc_decode(pred,
                                       input_length=input_len,
                                       greedy=True)[0][0]

    # Iterate over the results and get back the text
    output_text = []
    for res in results.numpy():
        outstr = ''
        for c in res:
            if c < len(characters) and c >=0:
                outstr += label_to_char[c]
        output_text.append(outstr)

    # return final text results
    return output_text
```

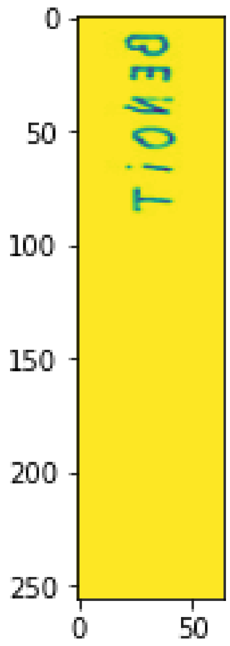
Model Evaluation

```
for p, (inp_value, _) in enumerate(validation_generator):
    bs = inp_value['input_data'].shape[0]
    X_data = inp_value['input_data']
    labels = inp_value['input_label']
    plt.imshow(X_data[0])
    preds = prediction_model.predict(X_data)
    pred_texts = decode_batch_predictions(preds)


    orig_texts = []
    for label in labels:
        text = ''.join([label_to_char[int(x)] for x in label])
        orig_texts.append(text)

    for i in range(bs):
        print(f'Ground truth: {orig_texts[i]} \t Predicted: {pred_texts[i]}')
    break
```

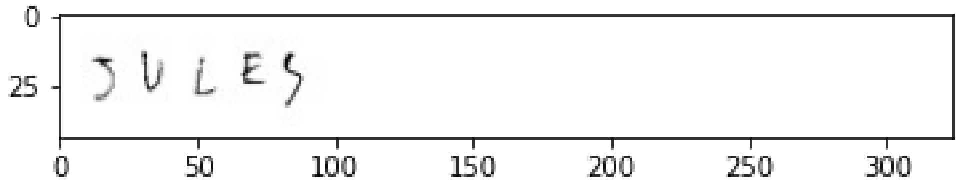
```
Ground truth: BENOIT     Predicted: BENOIT
Ground truth: ANGELINE   Predicted: ANGELINE
Ground truth: LEELOU     Predicted: LEELOU
Ground truth: VERDELET   Predicted: VERDELET
Ground truth: MAZVA      Predicted: MAEVA
Ground truth: JENNA      Predicted: JENNA
Ground truth: SAIDA      Predicted: SAIDA
Ground truth: YASSINE    Predicted: YASSINE
Ground truth: ALICE      Predicted: ALICE
Ground truth: ABASSA     Predicted: ABASIA
Ground truth: BOSIO      Predicted: ROSIO
Ground truth: SHARKAWI   Predicted: SHARAI
Ground truth: DRUOD      Predicted: BRUOR
Ground truth: JOHANNA    Predicted: JOHANNA
Ground truth: ALICIA     Predicted: ALICIA
Ground truth: FLORIAN    Predicted: FLORIAN
Ground truth: LEPROUX    Predicted: LEPROUX
Ground truth: CLEMENT    Predicted: CLEMENT
Ground truth: GALONNET   Predicted: GACONNET
Ground truth: BAMMEZ     Predicted: BAMMEZ
Ground truth: ISMAEL     Predicted: FSMAEL
Ground truth: ANDRES     Predicted: ANDRES
Ground truth: ORHAN      Predicted: BRHAN
Ground truth: LILA       Predicted: LILA
Ground truth: LUNTALA    Predicted: LUNTALA
Ground truth: FERAGO     Predicted: FARAGD
Ground truth: PAGANI     Predicted: DAGANI
Ground truth: LUCAS      Predicted: LUCAS
Ground truth: MAXIMILIAN       Predicted: MAXIMILIAN
Ground truth: LISA       Predicted: LISA
Ground truth: HUNDILU    Predicted: HUNDILU
Ground truth: LEULLIETTE       Predicted: LEULLIETTE
Ground truth: PARADOWSKI       Predicted: PARADONSRI
Ground truth: FRANCHI    Predicted: FRANCHZ
Ground truth: SAINTMARS        Predicted: SAINTHARS
Ground truth: GERVAIS    Predicted: GERVAIS
Ground truth: LISA       Predicted: ELSA
Ground truth: ERDNA      Predicted: ERDNA
Ground truth: REGGAM     Predicted: RESGAH
Ground truth: CLIPET     Predicted: CLIBET
Ground truth: SORIN      Predicted: GORIN
Ground truth: ETHAN      Predicted: ETHAN
Ground truth: FAHFOUHI   Predicted: FAAFOUMI
Ground truth: LOUIS      Predicted: LOUIS
Ground truth: VERDIER    Predicted: VEROIER
Ground truth: LEVY-DAUCHEZ     Predicted: LEVY-DAUCHEZ
Ground truth: DUSSAULE   Predicted: DUSSAULE
Ground truth: JADE       Predicted: SADE
Ground truth: GABOREAU   Predicted: GABOGEAU
Ground truth: JULIEN     Predicted: JULIEN
Ground truth: LOPES      Predicted: LOBES
Ground truth: ELSA       Predicted: ELSA
Ground truth: ELLIOT     Predicted: ELLIOT
Ground truth: MATHYS     Predicted: MATHIS
Ground truth: CORDANI    Predicted: CORDANI
Ground truth: KELYAN     Predicted: RLYAN
Ground truth: NICOLAS    Predicted: NICOLAS
Ground truth: JOREAN     Predicted: MASA
Ground truth: LANA       Predicted: LANA
Ground truth: ASWINN     Predicted: ROUINN
Ground truth: CORENTAN   Predicted: CORENTAN
Ground truth: CHLOE      Predicted: CHLLE
Ground truth: KYLIAN     Predicted: EILIAN
Ground truth: LORIANE    Predicted: LORIANE
Ground truth: LE QUERE   Predicted: LE QUERE
Ground truth: PEDANOU    Predicted: PEDANOU
Ground truth: STEPHAN    Predicted: STEHAN
Ground truth: DUPLAND    Predicted: SAMO
Ground truth: KERJOUAN   Predicted: KERSOUAN
Ground truth: HANON      Predicted: MANON
Ground truth: GRAMONT    Predicted: GRANONT
Ground truth: JOUBERT    Predicted: JOUBERT
Ground truth: RATAUO     Predicted: RATAUS
Ground truth: MATHWEO    Predicted: MATHEO
Ground truth: VALEZ BEAUPORT   Predicted: VELEIZ-BEAUFORT
Ground truth: AEGO       Predicted: REGO
Ground truth: LEA        Predicted: LEA
Ground truth: PAUL       Predicted: TOUL
Ground truth: CUNHA      Predicted: CUMAS
Ground truth: PRUNE      Predicted: PRUNE
Ground truth: NAUEAU     Predicted: NAUEAU
Ground truth: TEIXEIRA   Predicted: TEISEIRA
Ground truth: NACIM      Predicted: MACIM
Ground truth: LOEVANN    Predicted: LOEVAUN
Ground truth: LISSARDY   Predicted: CISSARDU
Ground truth: NATHAN     Predicted: NATHAN
Ground truth: LASNIER    Predicted: LASNIER
Ground truth: CHLOE      Predicted: CHLOE
Ground truth: CLAVEL     Predicted: ELPIES
Ground truth: CLAUDIE    Predicted: CLAUDIE
Ground truth: ALLICIO    Predicted: ALLICIO
Ground truth: GEIGER     Predicted: GEIGER
Ground truth: ANNE       Predicted: ANNE
Ground truth: MATTEO     Predicted: MATTS
Ground truth: SONNT      Predicted: SONNT
Ground truth: PIERRE     Predicted: DIEHAE
Ground truth: HERVO      Predicted: MERVO
Ground truth: TARTU      Predicted: TARTU
Ground truth: CLEMENT    Predicted: CLEMENT
Ground truth: DJODY      Predicted: DIDDY
Ground truth: COUDOUX    Predicted: COUDOUX
Ground truth: RAMDANI    Predicted: RAMDANI
Ground truth: BENJAMIN   Predicted: BENSAMIN
Ground truth: DEXHEIMER        Predicted: DEXMEIMER
Ground truth: EDOUARD    Predicted: EOOUARD
Ground truth: GUILLET    Predicted: GUILLET
Ground truth: LOANE      Predicted: LOANE
Ground truth: LUCILE     Predicted: LUCILE
Ground truth: PINCAU     Predicted: PINCAU
Ground truth: ALEXIS     Predicted: ALEXIS
Ground truth: MATTHIEU   Predicted: MATTHIEU
Ground truth: SUBRA      Predicted: SUBRA
Ground truth: HOCHART    Predicted: MOCHART
Ground truth: FONTAINE   Predicted: FONTAINE
Ground truth: LUDMILA    Predicted: LUONMILA
Ground truth: CAPUCINE   Predicted: CAPUCINE
Ground truth: DORIAN     Predicted: DORIAN
Ground truth: CHAUVEAU   Predicted: CHAUVEAU
Ground truth: HONA       Predicted: MAMA
Ground truth: LOS        Predicted: LOS
Ground truth: LECARDOWWEL      Predicted: LECARBONVEL
Ground truth: GHISKIER   Predicted: GHISRIER
Ground truth: LASNIER    Predicted: LASHIER
Ground truth: HELOISE    Predicted: MELOISE
Ground truth: MATHIS     Predicted: MATHIS
Ground truth: TEDDY      Predicted: TEDDY
Ground truth: CAMILLE    Predicted: CAMILLE
Ground truth: TOMMY      Predicted: TOMAS
```



Model Testing

```
batch_images=np.ones((128,256,64,1),dtype=np.float32)
img=cv2.imread('../input/handwriting-recognition/test_v2/test/TEST_0004.jpg')
img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img=cv2.resize(img,(256,64))
img=(img/255).astype(np.float32)
img=img.T
img=np.expand_dims(img,axis=-1)
batch_images[0]=img
x=prediction_model.predict(batch_images)
pred_texts = decode_batch_predictions(x)
pred_texts = pred_texts[0]
im=cv2.imread('../input/handwriting-recognition/test_v2/test/TEST_0004.jpg')
plt.imshow(im)
print('Predicted Text:',pred_texts)
```

```
Predicted Text: JULES
```


JULES