```
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('../input/handwriting-recognition/written_name_train_v2.csv')
validation=pd.read_csv('../input/handwriting-recognition/written_name_validation_v2.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

```
train[train['IDENTITY']=='zucchi']
```

	FILENAME	IDENTITY
20507	TRAIN_20508.jpg	zucchi

```
train['Length']=train['IDENTITY'].apply(lambda x : len(str(x)))
train=train[train['Length']<=21]
train['IDENTITY']=train['IDENTITY'].str.upper()
train[train['Length']==max(train['Length'])]
```

	FILENAME	IDENTITY	Length
10278	TRAIN_10279.jpg	DOSSANTOS LASCONCELOS	21
108567	TRAIN_108568.jpg	O ZUARRDI EREBEVITRAC	21
156500	TRAIN_156501.jpg	ANDRIANNARISOA–DEGHI	21
160575	TRAIN_160576.jpg	LOUISIANE - EVANGELIN	21
174121	TRAIN_174122.jpg	GU ILLOSSOO - OARRIBA	21
212583	TRAIN_212584.jpg	DE MALEZIEUX DU HAMEL	21
220135	TRAIN_220136.jpg	COSTREL DE CORAINVILL	21
229529	TRAIN_229530.jpg	ROUSSEAV -- CHAUDIERE	21
230011	TRAIN_230012.jpg	BEKNARD DE LAVERNETTE	21
308769	TRAIN_308770.jpg	DATE NAISSANCE CLASSE	21
325215	TRAIN_325216.jpg	BEHARY - LAUL - SIRDE	21

```
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)
```

```
char_to_label = {char:label for label,char in enumerate(characters)}
label_to_char = {label:char for label,char in enumerate(characters)}
```

```
path_train='../input/handwriting-recognition/train_v2/train'
path_validation='../input/handwriting-recognition/validation_v2/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)


# 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
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')(inp)
    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"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_data (InputLayer)         [(None, 256, 64, 1)] 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 256, 64, 64)  640         input_data[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 128, 32, 64)  0           conv2d[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 128, 32, 64)  0           max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 128, 32, 128) 73856       dropout[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 64, 16, 128)  0           conv2d_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 64, 16, 128)  0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 64, 2048)     0           dropout_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 64, 64)       131136      reshape[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 64, 64)       0           dense[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 64, 256)      197632      dropout_2[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 64, 128)      164352      bidirectional[0][0]
__________________________________________________________________________________________________
input_label (InputLayer)        [(None, 22)]         0
__________________________________________________________________________________________________
Dense_output (Dense)            (None, 64, 31)       3999        bidirectional_1[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
__________________________________________________________________________________________________
```

```
# 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=8)
```

```
Epoch 1/8
1000/1000 [==============================] - 1350s 1s/step - loss: 23.5019 - val_loss: 19.3841
Epoch 2/8
1000/1000 [==============================] - 827s 826ms/step - loss: 19.0949 - val_loss: 18.2144
Epoch 3/8
1000/1000 [==============================] - 601s 601ms/step - loss: 17.6263 - val_loss: 15.4947
Epoch 4/8
1000/1000 [==============================] - 472s 472ms/step - loss: 14.7576 - val_loss: 10.0791
Epoch 5/8
1000/1000 [==============================] - 401s 401ms/step - loss: 9.7509 - val_loss: 5.6907
Epoch 6/8
1000/1000 [==============================] - 379s 379ms/step - loss: 6.4620 - val_loss: 4.1072
Epoch 7/8
1000/1000 [==============================] - 338s 338ms/step - loss: 4.9822 - val_loss: 3.2935
Epoch 8/8
1000/1000 [==============================] - 319s 319ms/step - loss: 4.1453 - val_loss: 2.7474
```

```
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
_________________________________________________________________
```

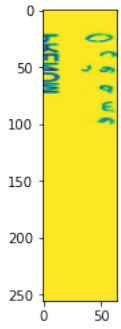
```
if 'prediction_model_ocr.h5' not in os.listdir('./'):
    prediction_model.save('prediction_model_ocr.h5')
    prediction_model=M.load_model('prediction_model_ocr.h5')
```

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

```
# Output decoding
def decode_batch_predictions(pred):
    pred = pred[:, :-2]
    input_len = np.ones(pred.shape[0])*pred.shape[1]

    # greedy search
    results = keras.backend.ctc_decode(pred,
                                       input_length=input_len,
                                       greedy=True)[0][0]

    # Iterate over the results
    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 output_text


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: OCEAME     Predicted: OESSI
Ground truth: NICOLAS    Predicted: NICOLAS
Ground truth: LAURENT    Predicted: LAURENT
Ground truth: ALEXIS     Predicted: ALEXIS
Ground truth: CHAILLOU   Predicted: CHAILLOU
Ground truth: VIRGINIE   Predicted: VIRGINIE
Ground truth: LEANA      Predicted: LEANA
Ground truth: MATHIS     Predicted: MATHIS
Ground truth: BOUQUACHMA         Predicted: BOUGURCHMA
Ground truth: CLEMENT    Predicted: CLEMENT
Ground truth: DJESSY     Predicted: DJESSY
Ground truth: SAGNIERDONOSO      Predicted: SAGNIERDONOSO
Ground truth: MAZINE     Predicted: MAZINE
Ground truth: SIBEL      Predicted: SIBEL
Ground truth: ENNA       Predicted: EMMA
Ground truth: BECHARAT   Predicted: BELHARAT
Ground truth: EBER       Predicted: EBER
Ground truth: BAHAA      Predicted: BAMAA
Ground truth: CLEMENTINE         Predicted: CLEMEUTIVE
Ground truth: NATHAN     Predicted: NATHAN
Ground truth: VASSAL     Predicted: VASSAL
Ground truth: WALKER     Predicted: NALKER
Ground truth: GARNIER    Predicted: GARNIER
Ground truth: TANT       Predicted: TANT
Ground truth: BALTHAZARD         Predicted: ALTHAZARD
Ground truth: ROMANE     Predicted: ROHANE
Ground truth: KRAYEM     Predicted: KRHIEN
Ground truth: ROLLION    Predicted: ROLLION
Ground truth: ARNOUX     Predicted: ARNOUX
Ground truth: ENZO       Predicted: EMRO
Ground truth: LUCAS      Predicted: LUCAS
Ground truth: LOUISE     Predicted: LOUISE
Ground truth: MACE       Predicted: MACE
Ground truth: LAURA      Predicted: LAURA
Ground truth: TOUBAL     Predicted: TOUBAL
Ground truth: LORRIAUX   Predicted: LORRIAUX
Ground truth: CHARLIZE   Predicted: CHARLIZE
Ground truth: GALTIE     Predicted: GALTIE
Ground truth: THIBAULT-LAURENT   Predicted: LEA
Ground truth: THEOPHILE          Predicted: THEOPHILE
Ground truth: BELARBI    Predicted: BELARBI
Ground truth: JULIETTE   Predicted: JULIETTE
Ground truth: DANDOLO    Predicted: DANDOLD
Ground truth: ZOLLINGER          Predicted: ZOLLINGER
Ground truth: AIRASS     Predicted: ANAEI
Ground truth: PIERRE     Predicted: PIERRE
Ground truth: MAXINE     Predicted: HAXIME
Ground truth: IRIS       Predicted: TRIS
Ground truth: MESSAOUR   Predicted: MESSROUR
Ground truth: JULIETTE   Predicted: JUCLETTE
Ground truth: BELDJOUDI          Predicted: BELOJOUDI
Ground truth: FRANCESCA          Predicted: FRANCESCR
Ground truth: GRAND      Predicted: GRAND
Ground truth: DIANE      Predicted: DIANE
Ground truth: ERWANN     Predicted: ERMANN
Ground truth: DIRNINGER          Predicted: DIRNINGGR
Ground truth: JADE       Predicted: JADE
Ground truth: DEMIR      Predicted: DEMIR
Ground truth: NICOLAS    Predicted: NICOLAS
Ground truth: NICOLAS    Predicted: NICOLAS
Ground truth: LEO-PAUL   Predicted: LEE
Ground truth: CAZIN      Predicted: CACIN
Ground truth: CREPIEUX   Predicted: CREPIEUX
Ground truth: COLOMBE    Predicted: COLOMDE
Ground truth: MAZURER    Predicted: MAZURER
Ground truth: BRICHLER   Predicted: BRICHLER
Ground truth: BELHOMME   Predicted: BELHOMHE
Ground truth: MAELYS     Predicted: MAELYS
Ground truth: DAMIEN     Predicted: DAMIEN
Ground truth: DESOUTTER          Predicted: DESOUTTER
Ground truth: TOM        Predicted: TOM
Ground truth: ROUGERIE   Predicted: ROUGERIE
Ground truth: JULIEN     Predicted: JULIEN
Ground truth: POISSONNET         Predicted: POISSONNET
Ground truth: MANON      Predicted: MANON
Ground truth: NATHAN     Predicted: NATHAN
Ground truth: MATHIS     Predicted: MATHIS
Ground truth: BAROUH     Predicted: GAROUH
Ground truth: ANTONIN    Predicted: ANTONIN
Ground truth: NOFMY      Predicted: NOENU
Ground truth: TALIBI     Predicted: TALIBI
Ground truth: LISA       Predicted: LISA
Ground truth: VICTOR     Predicted: VICTOR
Ground truth: EMMA       Predicted: EMAA
Ground truth: DUROT      Predicted: DUROT
Ground truth: LOUISE     Predicted: LOUISE
Ground truth: SARAH      Predicted: SARAN
Ground truth: YASSER     Predicted: YASSER
Ground truth: PIERRE     Predicted: PIERRE
Ground truth: ALYCIA     Predicted: ALYCIA
```

```
Ground truth: BOURDIN    Predicted: BOURDIN
Ground truth: CONTRERAS          Predicted: CONTRERAS
Ground truth: FILLEAU    Predicted: FILLEAU
Ground truth: ROMANE     Predicted: ROWANE
Ground truth: MANON      Predicted: MOMOI
Ground truth: TUEUX      Predicted: TUEUX
Ground truth: ELODIE     Predicted: ELOBIE
Ground truth: CARON      Predicted: CARON
Ground truth: MARTIN     Predicted: MARTIN
Ground truth: CHAUMON    Predicted: CHAUMON
Ground truth: THIBAULT   Predicted: THCBAULT
Ground truth: FOLIARD    Predicted: FOLIARD
Ground truth: MER        Predicted: HER
Ground truth: DELICIA    Predicted: MELICIA
Ground truth: GERVAIS    Predicted: GERUAIS
Ground truth: DRAGO      Predicted: ORAEO
Ground truth: CLARENCE   Predicted: CLARENCE
Ground truth: JEREMY     Predicted: JEREMY
Ground truth: JADE       Predicted: JADE
Ground truth: ALEXANDRE          Predicted: ALEXANDRE
Ground truth: BEAUFILS   Predicted: REAUEILE
Ground truth: REA        Predicted: REA
Ground truth: MAXENCE    Predicted: MAXENCE
Ground truth: ASMAHAN    Predicted: AOMAHAN
Ground truth: BINTZ      Predicted: BINTS
Ground truth: MATHIS     Predicted: MATHIS
Ground truth: LUCAS      Predicted: LUCAS
Ground truth: PEREZ      Predicted: BERRT
Ground truth: MARINE     Predicted: MARINE
Ground truth: VANDOMME   Predicted: VANDOMNE
Ground truth: ROCHETTE   Predicted: ROCHETTE
Ground truth: NICOLAS    Predicted: NICOLAS
Ground truth: ARNOULT    Predicted: ARNOULT
Ground truth: INES       Predicted: INES
Ground truth: INES       Predicted: INES
Ground truth: MUYNH      Predicted: HUYNH
Ground truth: VOGOUROUX          Predicted: VIGOUROUX
Ground truth: NOEMIE     Predicted: NOEMIE
```




```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Ignore UndefinedMetricWarning
import warnings
warnings.filterwarnings("ignore")

# Prepare the ground truth and predicted values
y_true = []
y_pred = []

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

    for i, label in enumerate(labels):
        true_text = ''.join([label_to_char[int(x)] for x in label])
        y_true.append(true_text)
        y_pred.append(pred_texts[i])

# Calculate the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Calculate the classification report
print("Classification Report:")
```

```
print(classification_report(y_true, y_pred, zero_division=0))

# Print the confusion matrix
print("Confusion Matrix:")
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=characters + [''], yticklabels=characters + [''])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
                precision    recall  f1-score   support

     ALEXANDRA       1.00      0.50      0.67         2
     ALEXANDRE       0.91      1.00      0.95        10
     ALEXANDRO       0.00      0.00      0.00         1
    ALEXANDROS       1.00      1.00      1.00         1
     ALEXANORA       0.00      0.00      0.00         0
        ALEXIA       1.00      1.00      1.00         2
       ALEXIAN       1.00      1.00      1.00         1
      ALEXIANE       1.00      1.00      1.00         1
        ALEXIS       1.00      0.90      0.95        10
       ALEXSET       0.00      0.00      0.00         0
        ALFANO       0.00      0.00      0.00         1
  ALFARO PAIMA       0.00      0.00      0.00         1
     ALFAZAZI       0.00      0.00      0.00         1
     ALHAJEED       0.00      0.00      0.00         1
     ALHAJEEO       0.00      0.00      0.00         0
        ALICE       1.00      1.00      1.00         4
       ALICIA       1.00      1.00      1.00         7
     ALIERIET       0.00      0.00      0.00         0
        ALINE       1.00      1.00      1.00         1
        ALING       0.00      0.00      0.00         0
       ALISSA       1.00      1.00      1.00         1
      ALIYYAH       0.00      0.00      0.00         1
       ALLAIR       0.00      0.00      0.00         0
      ALLAIRE       1.00      1.00      1.00         1
       ALLAIX       0.00      0.00      0.00         1
      ALLAOUI       1.00      1.00      1.00         1
     ALLIERES       1.00      1.00      1.00         1
         ALLO       1.00      1.00      1.00         1
        ALOIS       1.00      1.00      1.00         1
      ALOISIO       1.00      1.00      1.00         1
       ALONGO       0.00      0.00      0.00         0
       ALONSO       0.00      0.00      0.00         1
        ALRIO       0.00      0.00      0.00         0
        ALRIQ       0.00      0.00      0.00         1
     ALTERIET       0.00      0.00      0.00         1
    ALTHAZARD       0.00      0.00      0.00         0
        ALVES       0.00      0.00      0.00         1
       ALYCIA       1.00      0.50      0.67         2
       ALYLIA       0.00      0.00      0.00         0
      ALYVVAM       0.00      0.00      0.00         0
        AMAIS       0.00      0.00      0.00         1
```