Automate Identification And Recognition Of Handwritten Text from An Image.

(Using Convolutional Recurrent Neural Network)

Name Of The Student : SWAGAT KUMAR TRIPATHY

Internship Project Topic : Automate identification and recognition of handwritten text from an image

Name of the Organization : TCS iON

Name of the Industry Mentor : Debashis Roy

Introduction

An optical character recognition problem is basically a type of image-based sequence recognition problem. And for sequence recognition problem, most suited neural networks are recurrent neural networks(RNN) while for an image-based problem most suited are convolution neural networks(CNN). To cop up with the OCR problems we need to combine both of these CNN and RNN.

We can break the implementation of CRNN network into following steps:

1. Setting Up kaggle.
2. Collecting Dataset
3. Preprocessing Data
4. Creating Network Architecture
5. Defining Loss Function
6. Training Model
7. Testing and Prediction
8. Plot Accuracy and Loss.
9. Get Best Model Index
10. Save the Model.

Setting Up Kaggle in Google Colab.

This is optional method to run this model. This method is only for use of GPU on Google Colab. If one wants to use GPU on local machine then this step is not required. I used kaggle to load dataset in Google Colab. There are 4 steps to setting up kaggle in google colab.

1. Install Kaggle.
2. Create token.
3. Create Folder.
4. Get API link and download dataset.
5. Unzip the File.

Dataset

we used IAM handwritten datset. This is good dataset total of 1.16 GB images. Here I have used only 7850 images for the training set and 876 images for validation dataset.

To download the dataset either you can directly download from this link or use the following commands to download the data and unzip.

Installing Kaggle to use kaggle dataset on Google Colab.

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from k
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
```

First grab your token from kaggle.

For more details of creating API on kaggle click here.

Upload the json file got from kaggle.

```
from google.colab import files
files.upload() #upload kaggle.json
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"cse071","key":"57b8f55875ae93d360e6ec702e0e496a"}'}
```

Creat a folder to store kaggle dataset on colab.

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!chmod 600 /root/.kaggle/kaggle.json
```

Copy the API link and paste with '!' to download the datset.

```
!kaggle datasets download -d cse071/datafromiam
```

```
Downloading datafromiam.zip to /content
 97% 521M/535M [00:06<00:00, 96.8MB/s]
100% 535M/535M [00:06<00:00, 92.0MB/s]
```

This code for unzip the file.

```
from zipfile import ZipFile
file_name = "datafromiam.zip"
with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')
```

```
Done
```

Importing necessary packages.

Installing Keras_tqdm.

```
!pip install keras_tqdm
```

```
Collecting keras_tqdm
  Downloading https://files.pythonhosted.org/packages/16/5c/ac63c65b79a895b8994474de2
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from k
Requirement already satisfied: Keras in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (fror
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from k
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from h5
Installing collected packages: keras-tqdm
Successfully installed keras-tqdm-2.0.1
```

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

import os
from google.colab import drive #To use googel drive to get files.
```

```
from keras.preprocessing.sequence import pad_sequences

from keras.layers import Dense, LSTM, Reshape, BatchNormalization, Input, Conv2D, MaxPool2
from keras.models import Model
from keras.activations import relu, sigmoid, softmax
import keras.backend as K
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint
from keras_tqdm import TQDMNotebookCallback

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


import tensorflow as tf

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

Tensorflow GPU

We used Google Colab GPU.

```
from tensorflow.python.client import device_lib

# Check all available devices if GPU is available
print(device_lib.list_local_devices())
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True))
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17397697640172474202
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14674281152
locality {
  bus_id: 1
  links {
  }
}
incarnation: 18180369956942475455
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute c
]
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pci bus ic
```

This step is to check GPU is available or not.

```
tf.config.experimental.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

Preprocessing

Now we are having our dataset, to make it acceptable for our model we need to use some preprocessing. We need to preprocess both the input image and output labels. To preprocess our input image we will use followings: Read the image and convert into a gray-scale image Make each image of size (128,32) by using padding Expand image dimension as (128,32,1) to make it compatible with the input shape of architecture Normalize the image pixel values by dividing it with 255.

To preprocess the output labels use the followings: Read the text from the words.txt file because it contains text written inside the image. Which is in the format 'a01-000u-00-00 ok 154 408 768 27 51 AT A'.

Compute the maximum length from words and pad every output label to make it of the same size as the maximum length. This is done to make it compatible with the output shape of our RNN architecture. Then convert to numpy array.

Loading words.txt file in this function.

```
drive.mount('/content/gdrive')
with open('gdrive/My Drive/words.txt') as f:
    contents = f.readlines()

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

```
Mounted at /content/gdrive
'a01-000u-00-00 ok 154 408 768 27 51 AT A'
```

```
max_label_len = 0

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

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

print(char_list, len(char_list))
```

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

    return dig_lst
```

```
 !"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 78
```

```
images = []
labels = []

RECORDS_COUNT = 10000


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

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

inputs_length = []
labels_length = []


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

#     _, img = cv2.threshold(img,
#                            128,
#                            255,
#                            cv2.THRESH_BINARY | cv2.THRESH_OTSU)

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

    img = img.astype('float32')

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

```
        w, h = img.shape

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

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

    img = cv2.subtract(255, img)

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

    # Normalize
    img = img / 255

    return img
```

Generate Train and Validation set.

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

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

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

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

        # processing on label
        try:
            label = encode_to_labels(word)
        except:
            continue

        if index % 10 == 0:
            valid_images.append(img)
            valid_labels.append(label)
            valid_input_length.append(31)
            valid_label_length.append(len(word))
```

```
                valid_original_text.append(word)
            else:
                train_images.append(img)
                train_labels.append(label)
                train_input_length.append(31)
                train_label_length.append(len(word))
                train_original_text.append(word)

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

        if index >= RECORDS_COUNT:
            break
```

Generate Padded label

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

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

```
train_padded_label.shape, valid_padded_label.shape
```

```
((7850, 16), (876, 16))
```

Convert to numpy array.

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

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

```
train_images.shape
```

```
(7850, 32, 128, 1)
```

Build Model (Network Archtecture).

(Using Convolutional Recurrent Neural Network)

This network architecture is inspired by [this paper](). Let's see the steps that we used to create the architecture:

Input shape for our architecture having an input image of height 32 and width 128. Here we used seven convolution layers of which 6 are having kernel size (3,3) and the last one is of size (2.2). And the number of filters is increased from 64 to 512 layer by layer. Two max-pooling layers are added with size (2,2) and then two max-pooling layers of size (2,1) are added to extract features with a larger width to predict long texts. Also, we used batch normalization layers after fifth and sixth convolution layers which accelerates the training process. Then we used a lambda function to squeeze the output from conv layer and make it compatible with LSTM layer. Then used two Bidirectional LSTM layers each of which has 128 units. This RNN layer gives the output of size (batch_size, 31, 63). Where 63 is the total number of output classes including blank character.

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

# convolution layer with kernel size (3,3)
conv_1 = Conv2D(64, (3,3), activation = 'relu', padding='same')(inputs)
# poolig layer with kernel size (2,2)
pool_1 = MaxPool2D(pool_size=(2, 2), strides=2)(conv_1)

conv_2 = Conv2D(128, (3,3), activation = 'relu', padding='same')(pool_1)
pool_2 = MaxPool2D(pool_size=(2, 2), strides=2)(conv_2)

conv_3 = Conv2D(256, (3,3), activation = 'relu', padding='same')(pool_2)

conv_4 = Conv2D(256, (3,3), activation = 'relu', padding='same')(conv_3)

# poolig layer with kernel size (2,1)
pool_4 = MaxPool2D(pool_size=(2, 1))(conv_4)

conv_5 = Conv2D(512, (3,3), activation = 'relu', padding='same')(pool_4)

# Batch normalization layer
batch_norm_5 = BatchNormalization()(conv_5)

conv_6 = Conv2D(512, (3,3), activation = 'relu', padding='same')(batch_norm_5)
batch_norm_6 = BatchNormalization()(conv_6)
pool_6 = MaxPool2D(pool_size=(2, 1))(batch_norm_6)

conv_7 = Conv2D(512, (2,2), activation = 'relu')(pool_6)

squeezed = Lambda(lambda x: K.squeeze(x, 1))(conv_7)

# bidirectional LSTM layers with units=128
blstm_1 = Bidirectional(LSTM(256, return_sequences=True, dropout = 0.2))(squeezed)
blstm_2 = Bidirectional(LSTM(256, return_sequences=True, dropout = 0.2))(blstm_1)

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

# model to be used at test time
```

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


act_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 128, 1)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 128, 64)       640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 64, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 32, 128)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 32, 256)        295168
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 32, 256)        590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 32, 256)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 32, 512)        1180160
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 32, 512)        2048
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 32, 512)        2359808
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 32, 512)        2048
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 32, 512)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 1, 31, 512)        1049088
_________________________________________________________________
lambda_1 (Lambda)            (None, 31, 512)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 31, 512)           1574912
_________________________________________________________________
bidirectional_2 (Bidirection (None, 31, 512)           1574912
_________________________________________________________________
dense_1 (Dense)              (None, 31, 79)            40527
=================================================================
Total params: 8,743,247
Trainable params: 8,741,199
Non-trainable params: 2,048
_________________________________________________________________
```

Loss Function

Here, we are using the CTC loss function. CTC loss is very helpful in text recognition problems. It helps us to prevent annotating each time step and help us to get rid of the problem where a single character can span multiple time step which needs further processing if we do not use CTC.

A CTC loss function requires four arguments to compute the loss, predicted outputs, ground truth labels, input sequence length to LSTM and ground truth label length. To get this we need to create a custom loss function and then pass it to the model. To make it compatible with our model, we will create a model which takes these four inputs and outputs the loss. This model will be used for training and for testing we will use the model that we have created earlier "act_model". Let's see the code:

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

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

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

loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([outputs, the_labels, in

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

Train the Model

To train the model we will use Adam optimizer. Also, we can use Keras callbacks functionality to save the weights of the best model on the basis of validation loss. In model.compile(), you can see that I have only taken y_pred and neglected y_true. This is because I have already taken labels as input to the model earlier. labels as input to the model earlier.

Now train your model on 7850 training images and 876 validation images.

```
batch_size = 8
epochs = 30
e = str(epochs)
optimizer_name = 'sgd'


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

filepath="gdrive/My Drive/TcsInternship/parser/{}o-{}r-{}e-{}t-{}v.hdf5".format(optimizer_
                                          str(RECORDS_COUNT),
                                          str(epochs),
                                          str(train_images.shape[0]),
                                          str(valid_images.shape[0]))

checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_o
callbacks_list = [checkpoint]


history = model.fit(x=[train_images, train_padded_label, train_input_length, train_label_l
```

```
                  y=np.zeros(len(train_images)),
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=([valid_images, valid_padded_label, valid_input_length
                  verbose=2,
                  callbacks=callbacks_list)
```

```
Train on 7850 samples, validate on 876 samples
Epoch 1/30
 - 229s - loss: 15.7982 - accuracy: 0.0000e+00 - val_loss: 22.7894 - val_accuracy:

Epoch 00001: val_loss improved from inf to 22.78938, saving model to gdrive/My Dri
Epoch 2/30
 - 224s - loss: 13.4703 - accuracy: 0.0089 - val_loss: 12.8756 - val_accuracy: 0.0

Epoch 00002: val_loss improved from 22.78938 to 12.87564, saving model to gdrive/M
Epoch 3/30
 - 222s - loss: 11.9471 - accuracy: 0.0466 - val_loss: 11.9610 - val_accuracy: 0.0

Epoch 00003: val_loss improved from 12.87564 to 11.96102, saving model to gdrive/M
Epoch 4/30
 - 221s - loss: 10.4667 - accuracy: 0.0685 - val_loss: 16.7443 - val_accuracy: 0.0

Epoch 00004: val_loss did not improve from 11.96102
Epoch 5/30
 - 217s - loss: 8.9951 - accuracy: 0.1096 - val_loss: 13.5724 - val_accuracy: 0.08

Epoch 00005: val_loss did not improve from 11.96102
Epoch 6/30
 - 216s - loss: 7.4734 - accuracy: 0.1517 - val_loss: 18.8284 - val_accuracy: 0.02

Epoch 00006: val_loss did not improve from 11.96102
Epoch 7/30
 - 216s - loss: 5.8994 - accuracy: 0.1939 - val_loss: 5.4707 - val_accuracy: 0.229

Epoch 00007: val_loss improved from 11.96102 to 5.47070, saving model to gdrive/My
Epoch 8/30
 - 215s - loss: 4.7142 - accuracy: 0.2414 - val_loss: 19.6279 - val_accuracy: 0.00

Epoch 00008: val_loss did not improve from 5.47070
Epoch 9/30
 - 215s - loss: 3.8069 - accuracy: 0.2860 - val_loss: 4.2376 - val_accuracy: 0.301

Epoch 00009: val_loss improved from 5.47070 to 4.23760, saving model to gdrive/My
Epoch 10/30
 - 216s - loss: 3.1251 - accuracy: 0.3362 - val_loss: 5.3671 - val_accuracy: 0.238

Epoch 00010: val_loss did not improve from 4.23760
Epoch 11/30
 - 216s - loss: 2.5277 - accuracy: 0.3862 - val_loss: 5.2800 - val_accuracy: 0.282

Epoch 00011: val_loss did not improve from 4.23760
Epoch 12/30
 - 214s - loss: 2.0851 - accuracy: 0.4413 - val_loss: 3.5682 - val_accuracy: 0.372

Epoch 00012: val_loss improved from 4.23760 to 3.56821, saving model to gdrive/My
Epoch 13/30
 - 212s - loss: 1.6707 - accuracy: 0.4926 - val_loss: 3.4039 - val_accuracy: 0.452

Epoch 00013: val_loss improved from 3.56821 to 3.40386, saving model to gdrive/My
Epoch 14/30
```

```
 - 212s - loss: 1.3409 - accuracy: 0.5468 - val_loss: 3.4710 - val_accuracy: 0.453

Epoch 00014: val_loss did not improve from 3.40386
Epoch 15/30
```

Test the Model

Our model is now trained with 7850 images. Now its time to test the model. We can not use our training model because it also requires labels as input and at test time we can not have labels. So to test the model we will use " act_model " that we have created earlier which takes only one input: test images.

As our model predicts the probability for each class at each time step, we need to use some transcription function to convert it into actual texts. Here we used the CTC decoder to get the output text. Let's see the code:

We use Jaro Distance & Ratio method to test accuracy.

Installing Levenshtein package in google colab.

```
!pip install python-levenshtein
```

```
Collecting python-levenshtein
  Downloading https://files.pythonhosted.org/packages/42/a9/d1785c85ebf9b7dfacd08938
     |████████████████████████████████| 51kB 2.3MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Building wheels for collected packages: python-levenshtein
  Building wheel for python-levenshtein (setup.py) ... done
  Created wheel for python-levenshtein: filename=python_Levenshtein-0.12.0-cp36-cp36m
  Stored in directory: /root/.cache/pip/wheels/de/c2/93/660fd5f7559049268ad2dc6d81c4e
Successfully built python-levenshtein
Installing collected packages: python-levenshtein
Successfully installed python-levenshtein-0.12.0
```

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

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

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

import Levenshtein as lv

total_jaro = 0
total_rati = 0
```

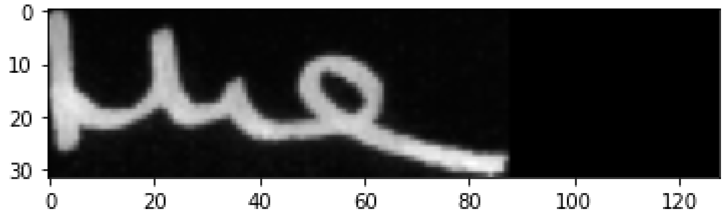
```
# see the results
for i, x in enumerate(out):
    letters=''
    for p in x:
        if int(p) != -1:
            letters+=char_list[int(p)]
    total_jaro+=lv.jaro(letters, valid_original_text[i])
    total_rati+=lv.ratio(letters, valid_original_text[i])

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

```
jaro : 0.9172121632595385
ratio: 0.8879421627081149
```

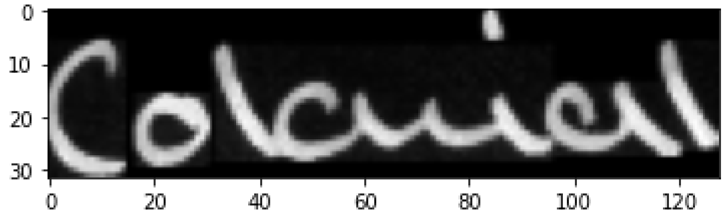
Prediction.

```
# predict outputs on validation images
prediction =act_model.predict(train_images[542:645])

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

out = K.get_value(decoded)

# see the results
for i, x in enumerate(out):
    print("original_text =  ", train_original_text[542+i])
    print("predicted text = ", end = '')
    for p in x:
        if int(p) != -1:
            print(char_list[int(p)], end = '')
    plt.imshow(train_images[542+i].reshape(32,128), cmap=plt.cm.gray)
    plt.show()
    print('\n')
```

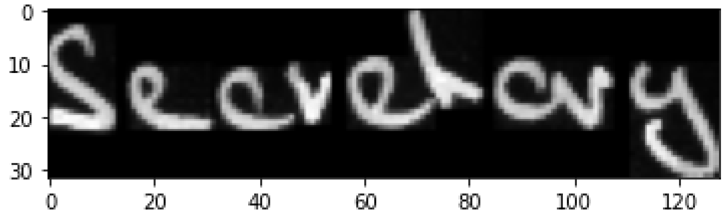
```
original_text =   the
predicted text = the
```



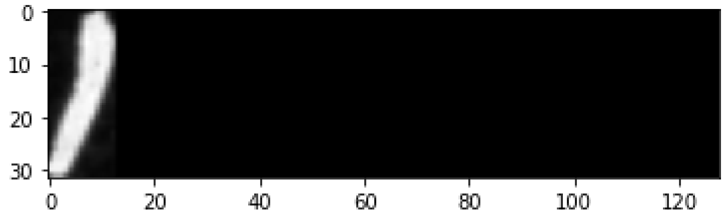

```
original_text =   Colonial
predicted text = Colonial
```



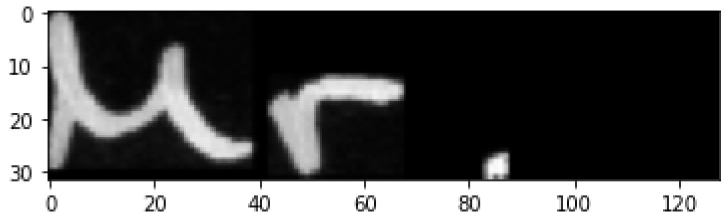

```
original_text =   Secretary
predicted text = Secretary
```



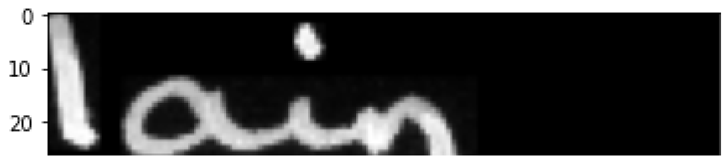

```
original_text =   ,
predicted text = ,
```



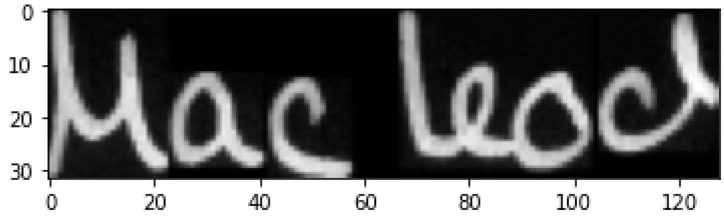

```
original_text =   Mr.
predicted text = Mr.
```



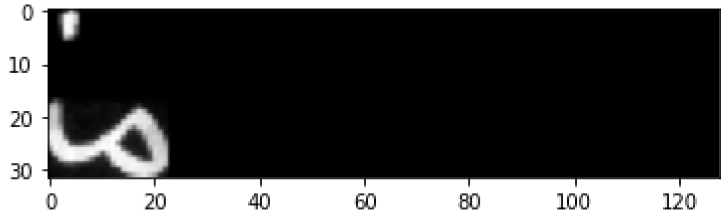

```
original_text =   Iain
predicted text = Iain
```



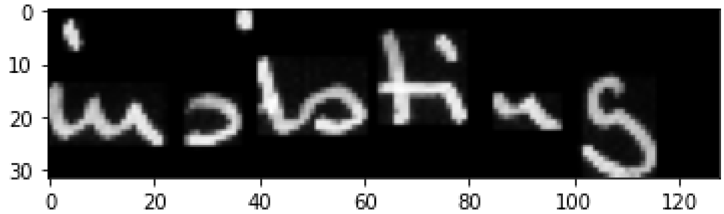
```
original_text =   Macleod
predicted text = Macleod
```



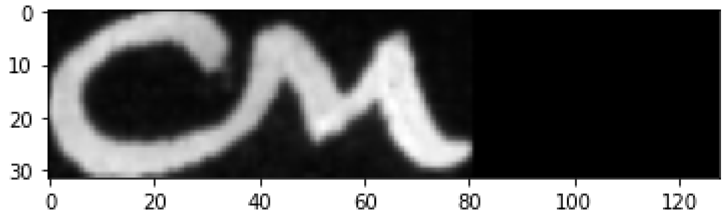
```
original_text =   is
predicted text = is
```



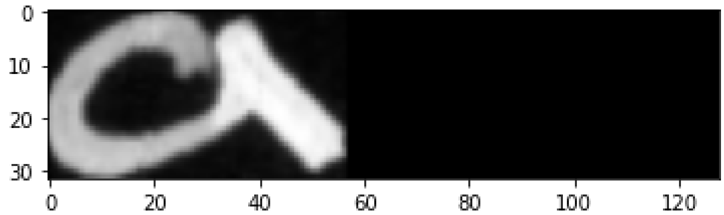
```
original_text =   insisting
predicted text = insisting
```



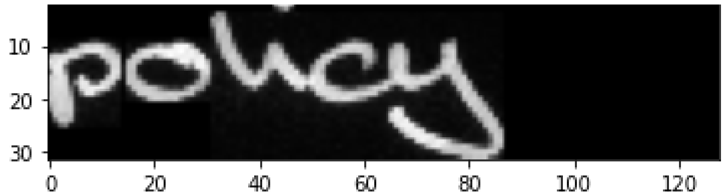
```
original_text =   on
predicted text = on
```



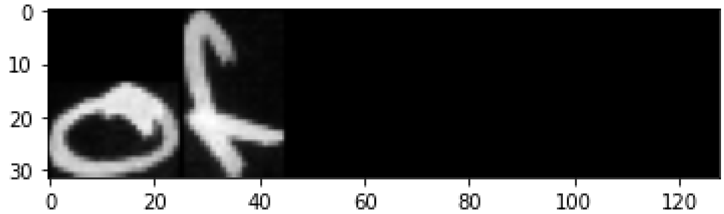
```
original_text =   a
predicted text = a
```



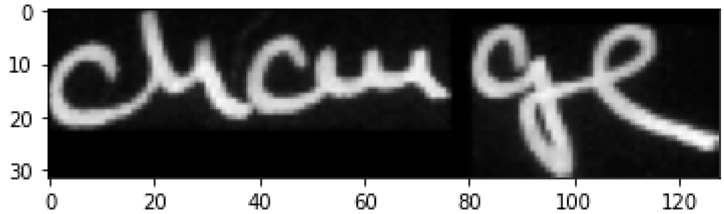
```
original_text =   policy
predicted text = policy
```



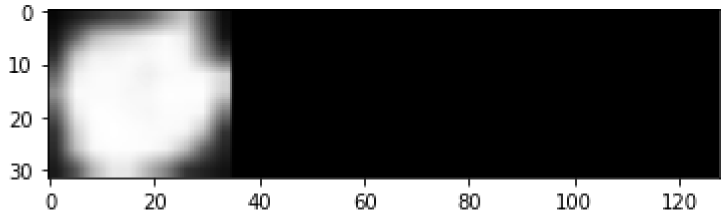
```
original_text =   of
predicted text = of
```



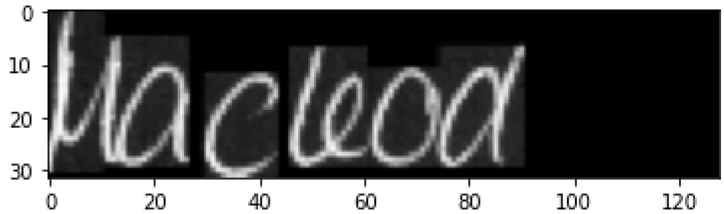
```
original_text =   change
predicted text = change
```



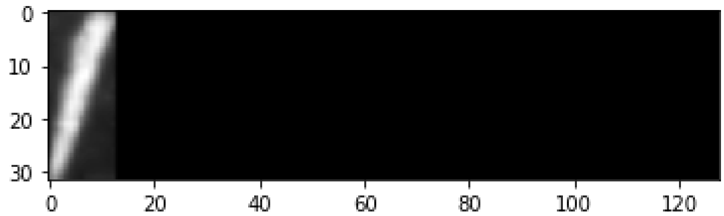
```
original_text =   .
predicted text = .
```



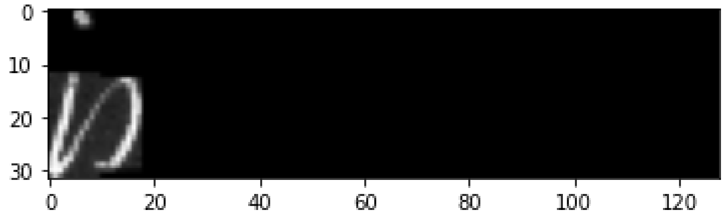
```
original_text =   Macleod
predicted text = Macleod
```



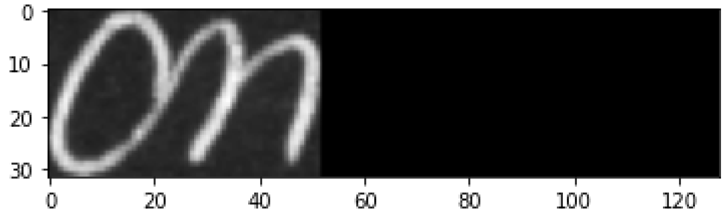
```
original_text =   ,
predicted text = ,
```



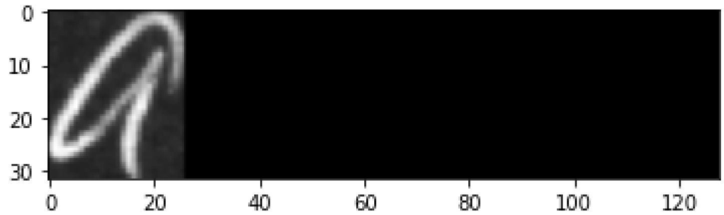
```
original_text =   is
predicted text = is
```



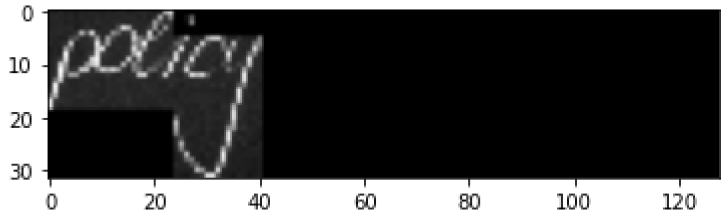
```
original_text =   on
predicted text = on
```



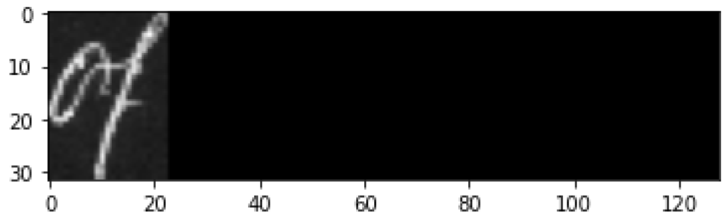
```
original_text =   a
predicted text = a
```



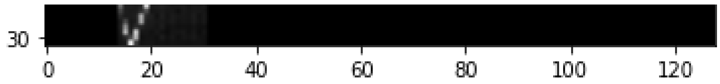
```
original_text =   policy
predicted text = policy
```



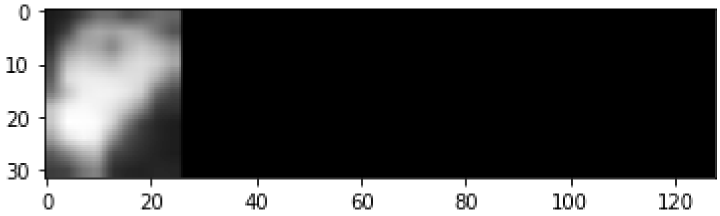
```
original_text =   of
predicted text = of
```



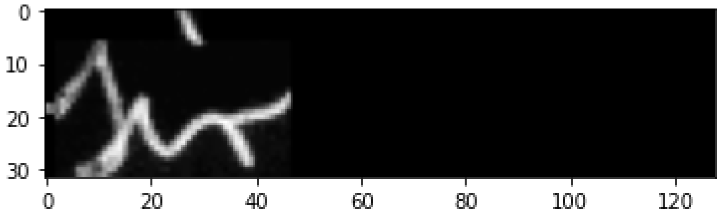
```
original_text =   change
predicted text = change
```



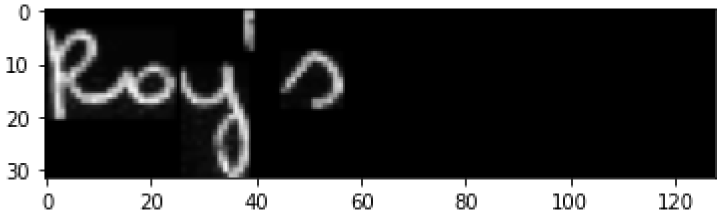
```
original_text =   .
predicted text = .
```



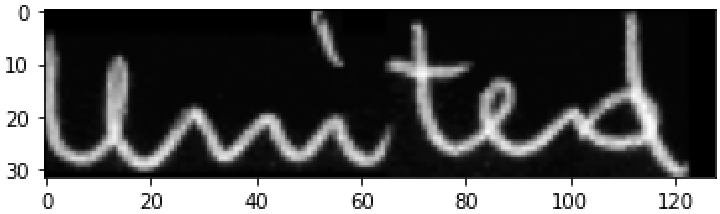
```
original_text =   Sir
predicted text = Sir
```



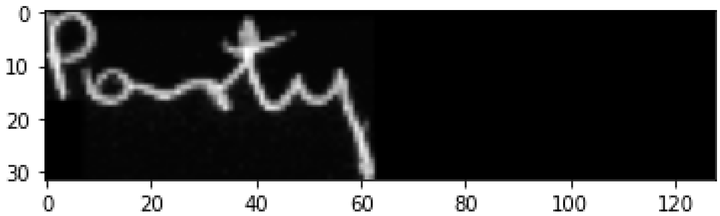
```
original_text =   Roy's
predicted text = Roy's
```



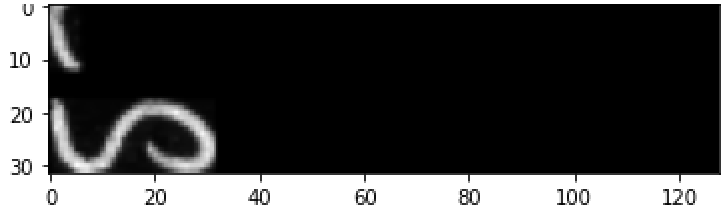
```
original_text =   United
predicted text = United
```



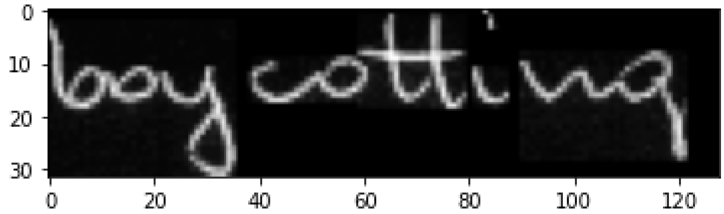
```
original_text =   Party
predicted text = Party
```



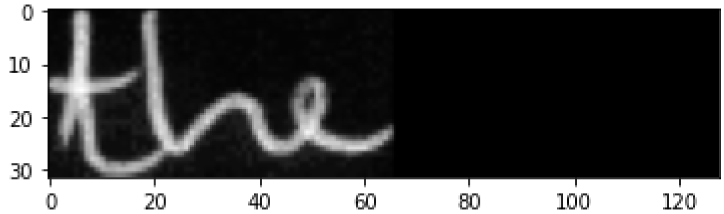
```
original_text =   is
predicted text = is
```



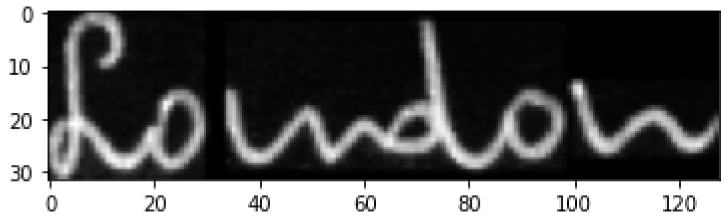
```
original_text =   boycotting
predicted text = boycotting
```



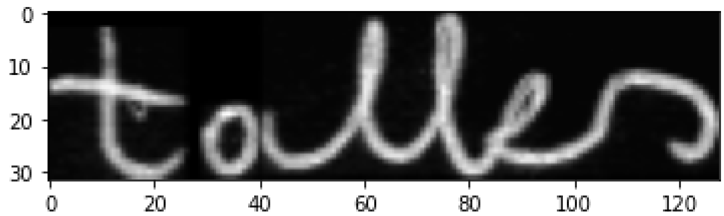
```
original_text =   the
predicted text = the
```



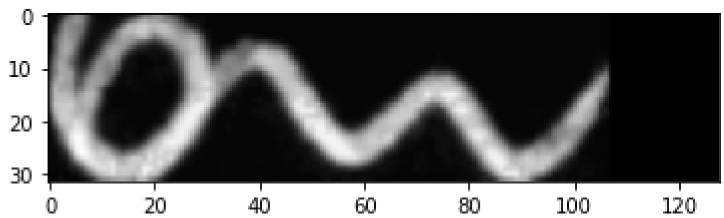
```
original_text =   London
predicted text = London
```



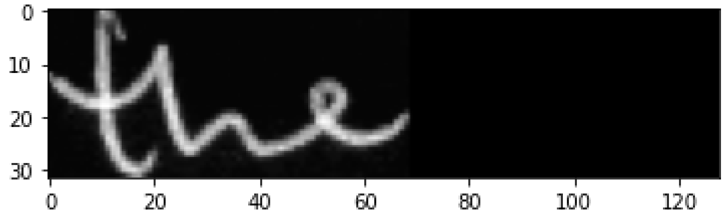
```
original_text =   talks
predicted text = talks
```



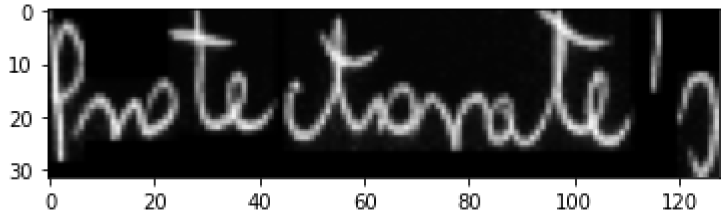
```
original_text =   on
predicted text = on
```



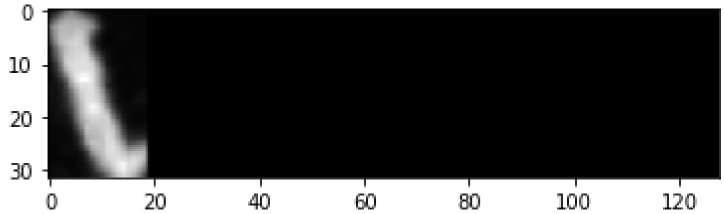
```
original_text =   the
predicted text = the
```



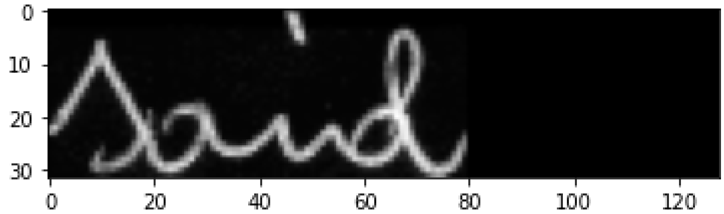
```
original_text =   Protectorate's
predicted text = Protectorate's
```



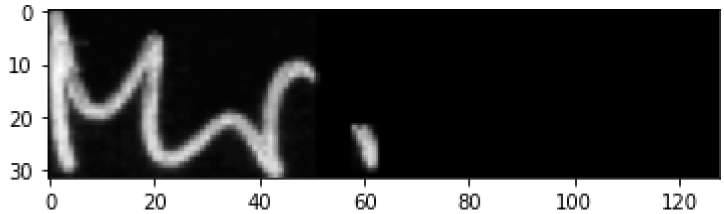
```
original_text =   .
predicted text = .
```



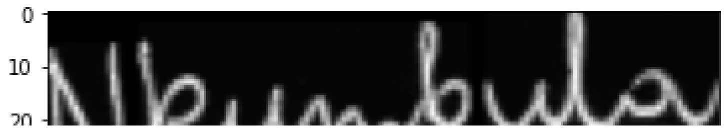
```
original_text =   Said
predicted text = Said
```



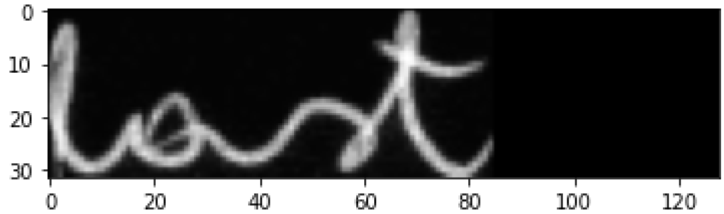
```
original_text =   Mr.
predicted text = Mr.
```



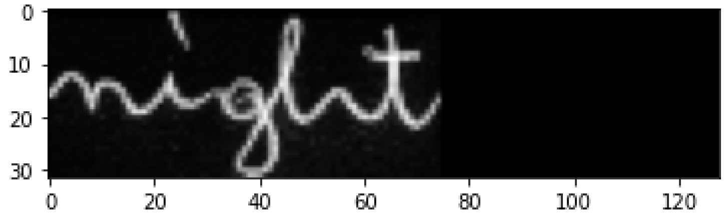
```
original_text =   Nkumbula
predicted text = Nkumbula
```



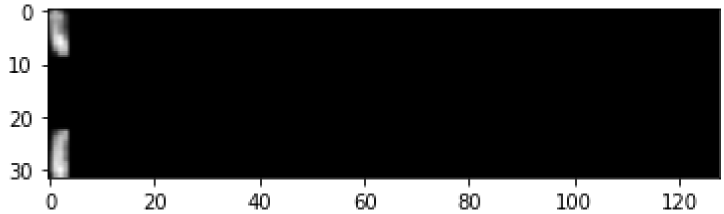

```
original_text =   last
predicted text = last
```



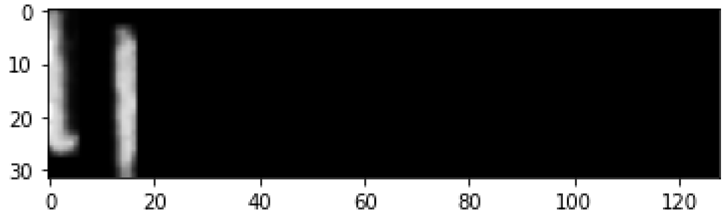

```
original_text =   night
predicted text = night
```



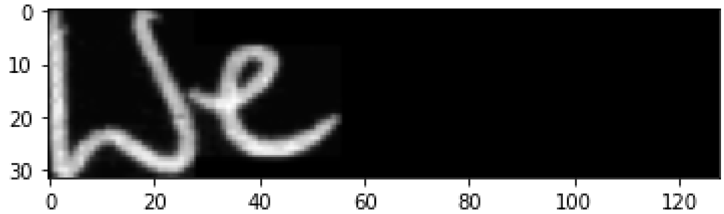

```
original_text =   :
predicted text = :
```



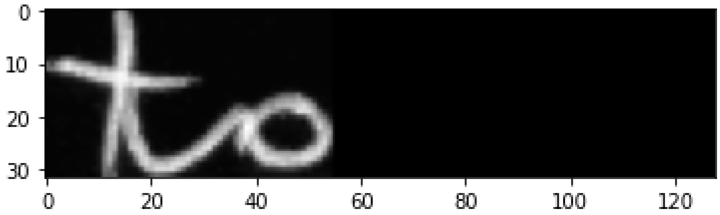

```
original_text =   "
predicted text = "
```



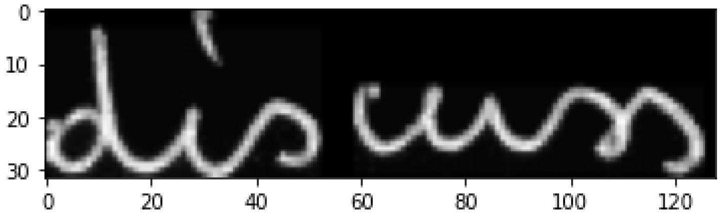

```
original_text =   We
predicted text = We
```



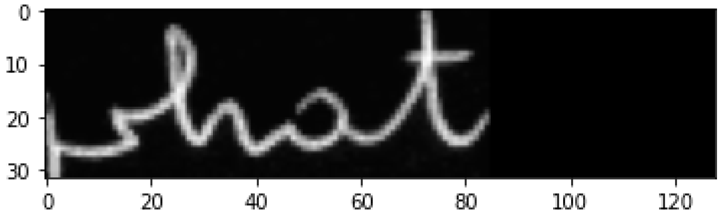

```
original_text =   to
predicted text = to
```



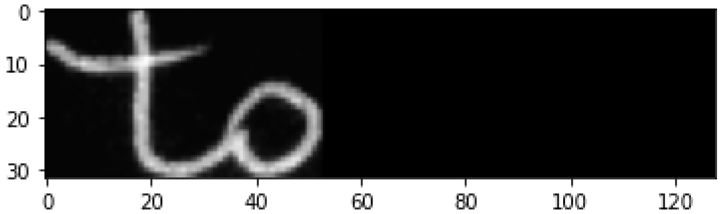
```
original_text =   discuss
predicted text = discuss
```



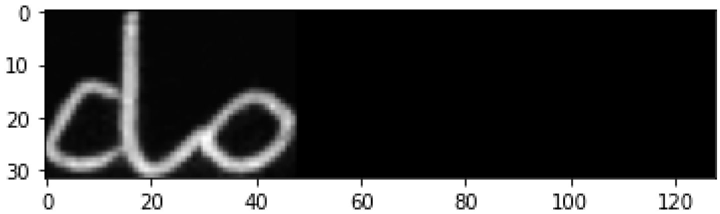
```
original_text =   what
predicted text = what
```



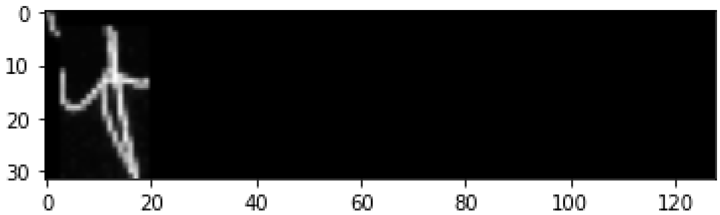
```
original_text =   to
predicted text = to
```



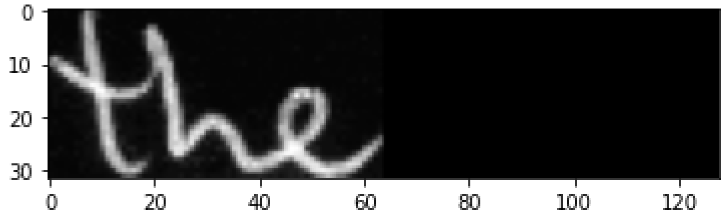
```
original_text =   do
predicted text = do
```



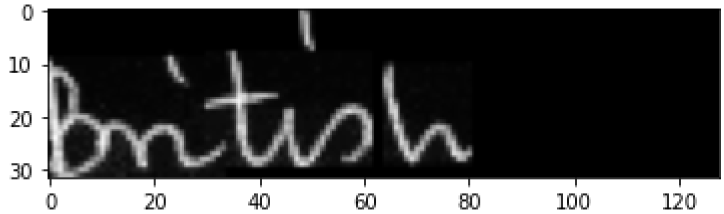
```
original_text =   if
predicted text = if
```



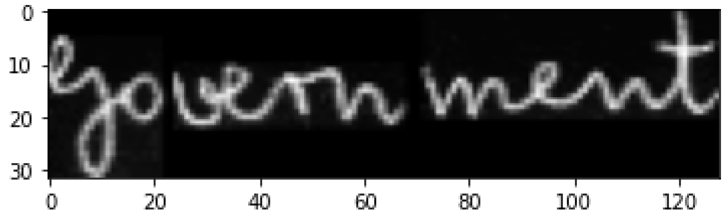
```
original_text =   the
predicted text = the
```



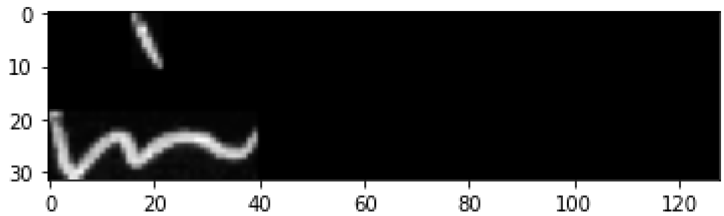
```
original_text =   British
predicted text = British
```



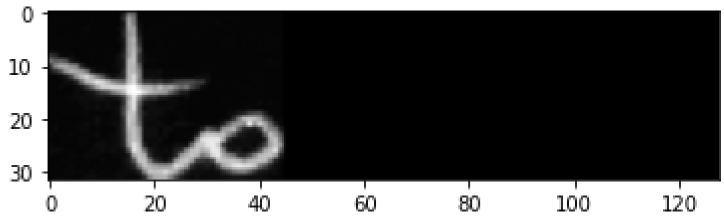
```
original_text =   Government
predicted text = Government
```



```
original_text =   in
predicted text = in
```

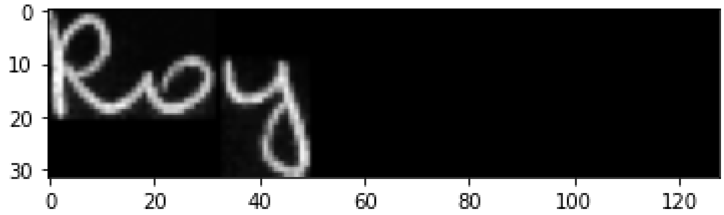


```
original_text =   to
predicted text = to
```

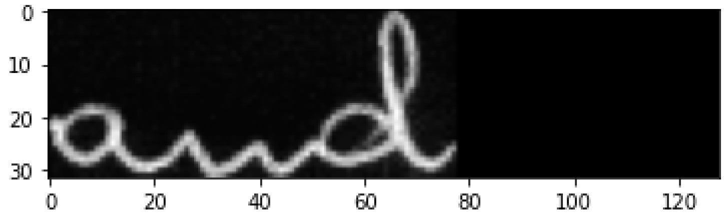


```
original_text =   Sir
predicted text = Sir
```

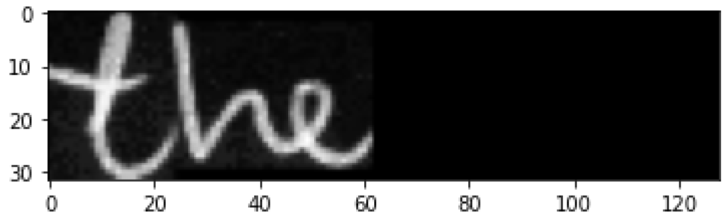
```
original_text =   Roy
predicted text = Roy
```



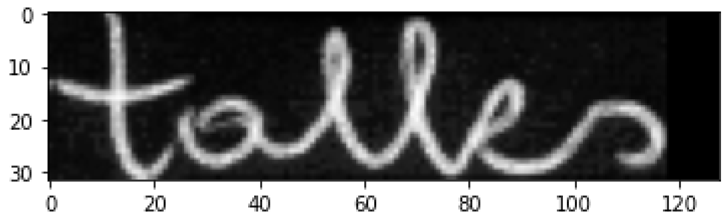
```
original_text =   and
predicted text = and
```



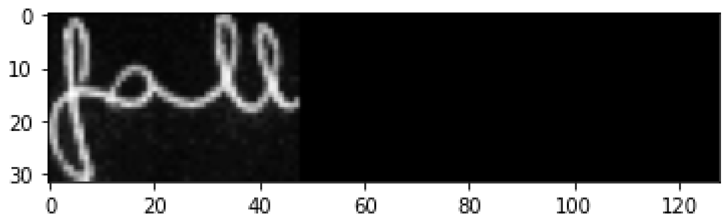
```
original_text =   the
predicted text = the
```



```
original_text =   talks
predicted text = talks
```

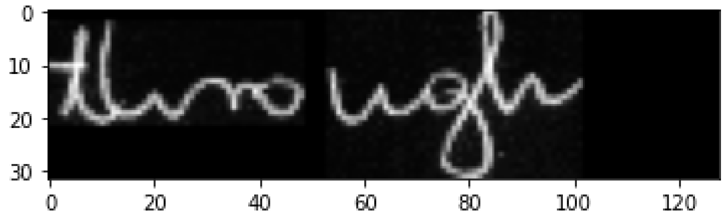


```
original_text =   fall
predicted text = fall
```

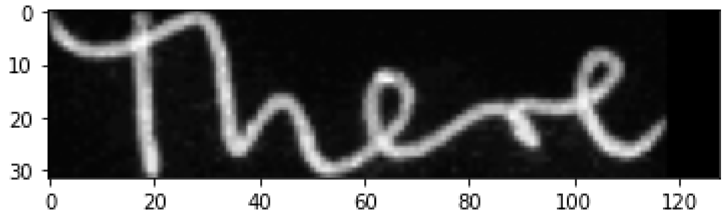


```
original_text =   through
predicted text = through
```

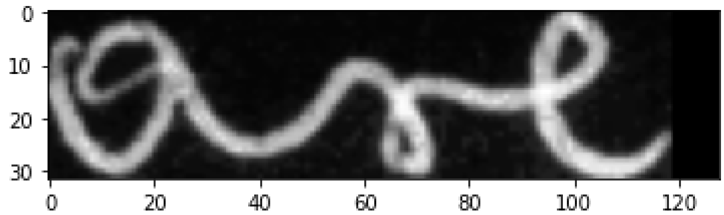
predicted text = through



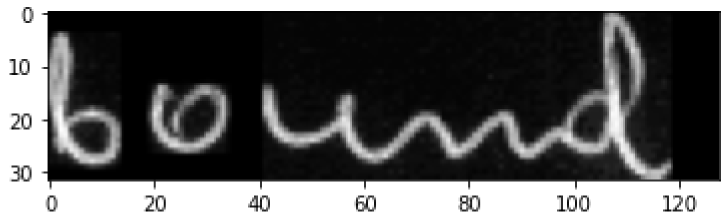
original_text =   There
predicted text = There



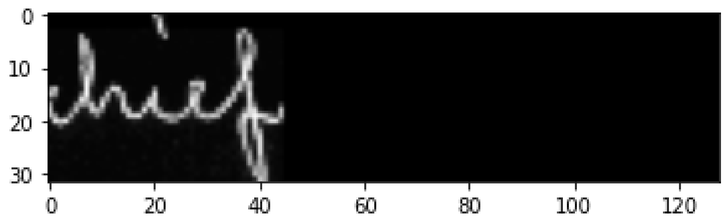
original_text =   are
predicted text = are



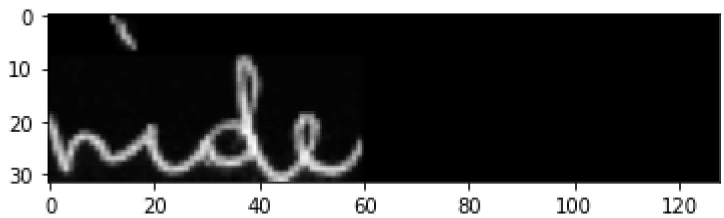
original_text =   bound
predicted text = bound



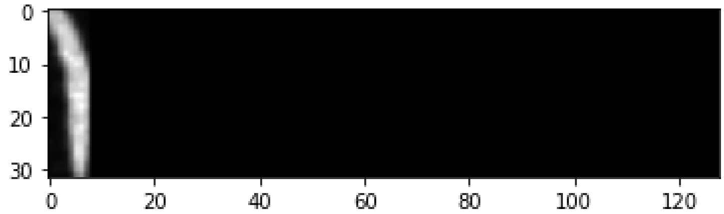
original_text =   chief
predicted text = chief



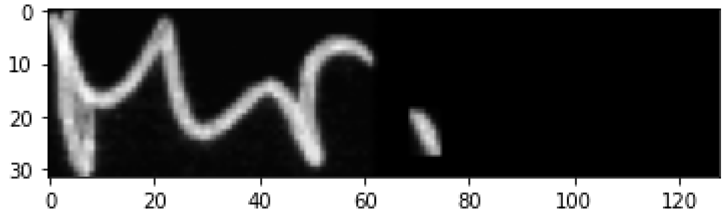
original_text =   aide
predicted text = aide



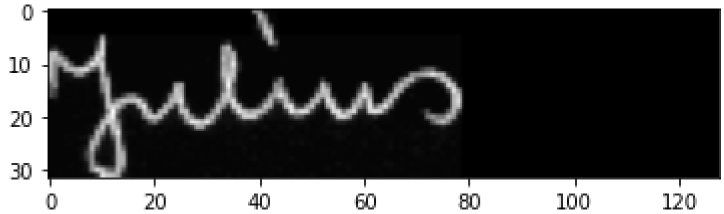
```
original_text =   ,
predicted text = ,
```



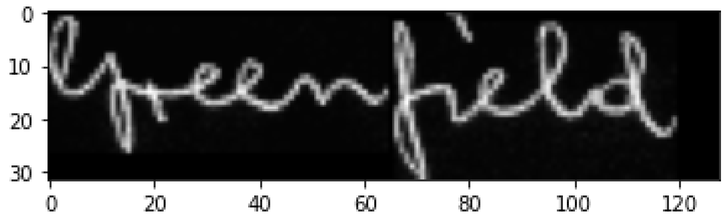
```
original_text =   Mr.
predicted text = Mr.
```



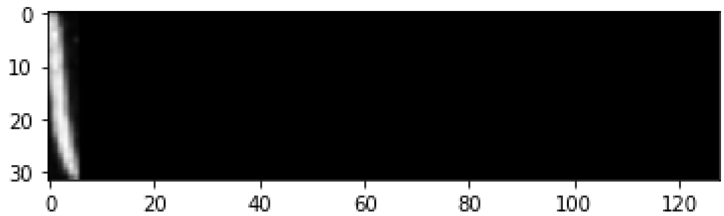
```
original_text =   Julius
predicted text = Julius
```



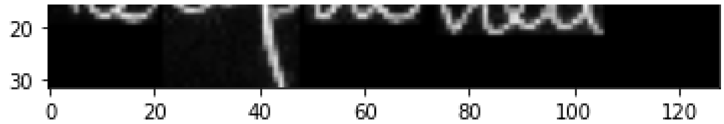
```
original_text =   Greenfield
predicted text = Geenfield
```



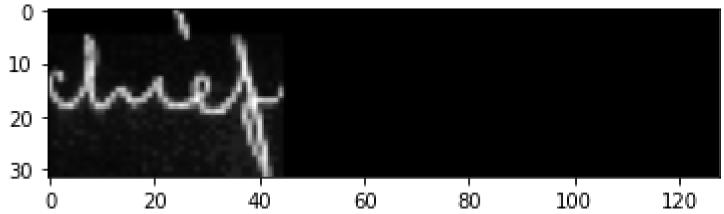
```
original_text =   ,
predicted text = ,
```



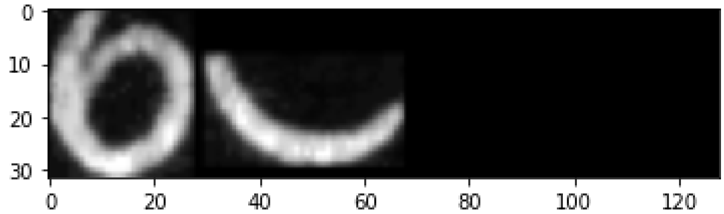
```
original_text =   telephoned
predicted text = telephoned
```



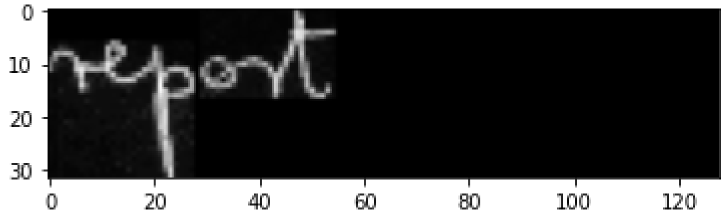
```
original_text =   chief
predicted text = chief
```



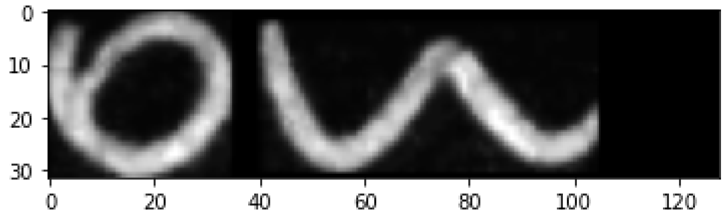
```
original_text =   a
predicted text = a
```



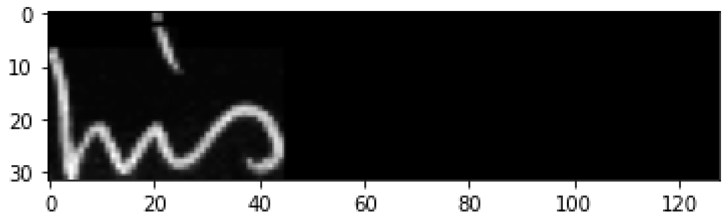
```
original_text =   report
predicted text = report
```



```
original_text =   on
predicted text = on
```

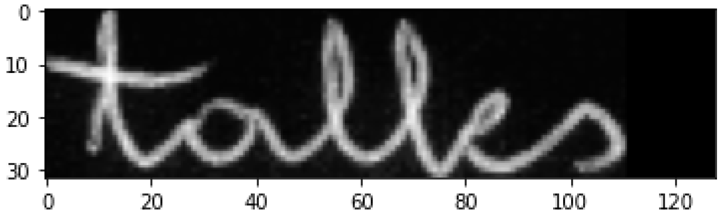


```
original_text =   his
predicted text = his
```

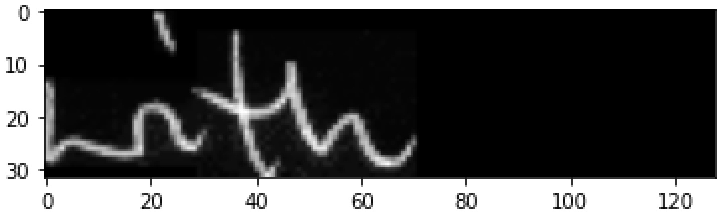


```
original_text =   talks
```

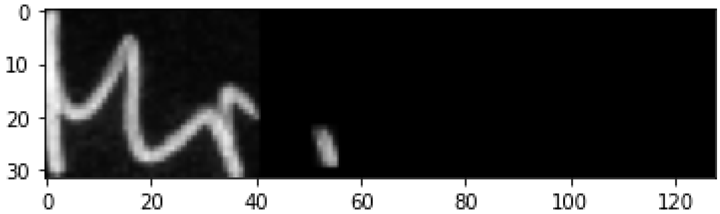
```
predicted text = tals
```



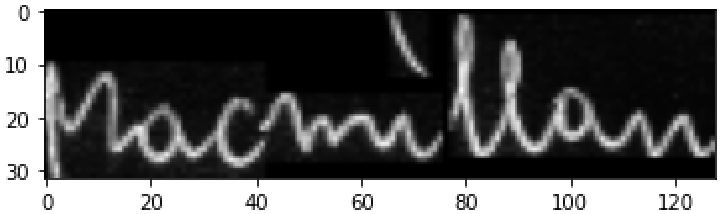
```
original_text =   with
predicted text = with
```



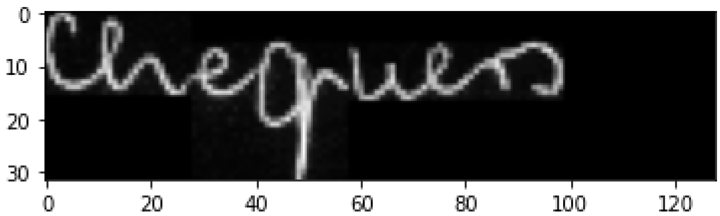
```
original_text =   Mr.
predicted text = Mr.
```



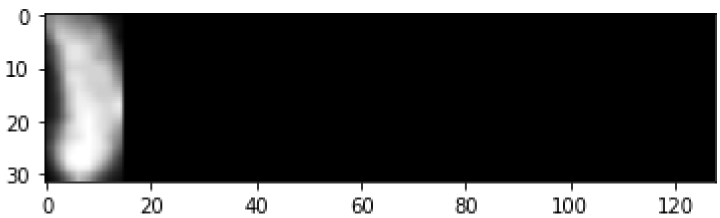
```
original_text =   Macmillan
predicted text = Macmillan
```



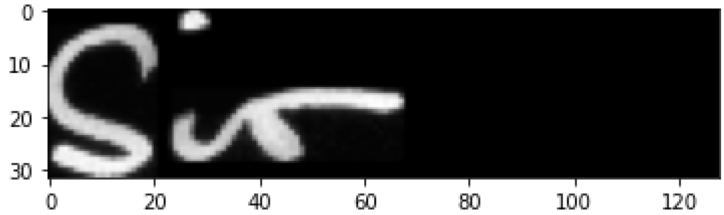
```
original_text =   Chequers
predicted text = Cheqhuers
```



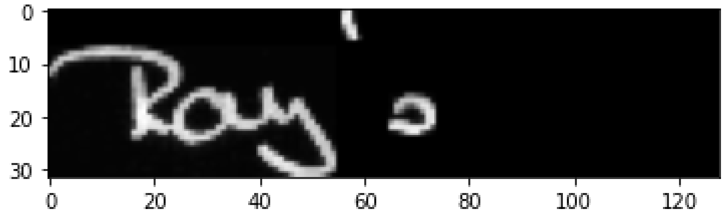
```
original_text =   .
predicted text = .
```



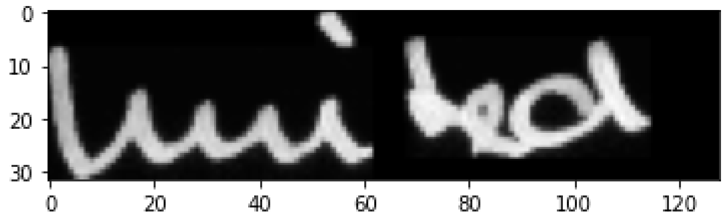
```
original_text =   Sir
predicted text = Sir
```



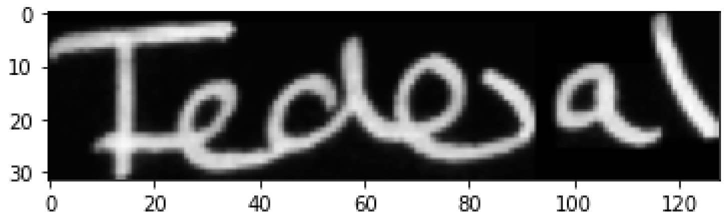
```
original_text =   Roy's
predicted text = Roy's
```



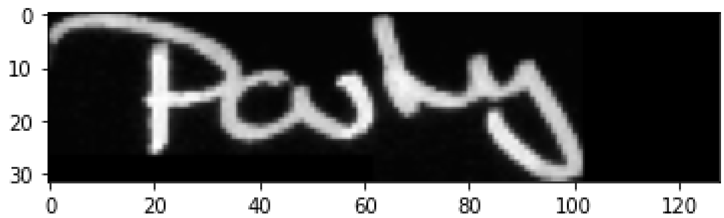
```
original_text =   United
predicted text = United
```



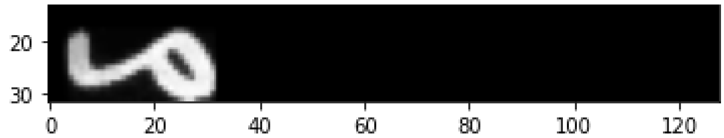
```
original_text =   Federal
predicted text = Federal
```



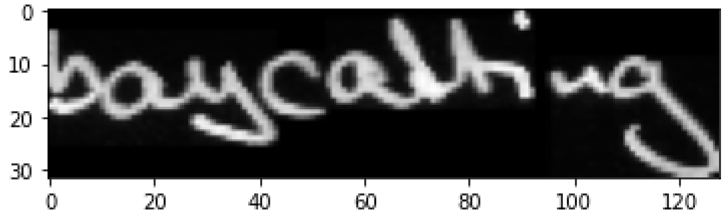
```
original_text =   Party
predicted text = Party
```



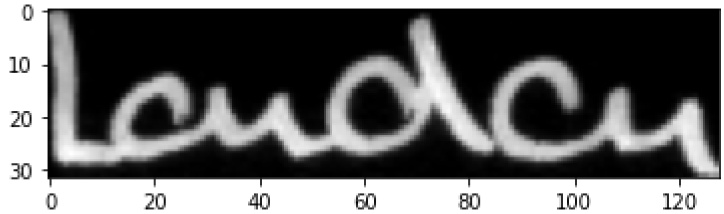
```
original_text =   is
predicted text = is
```



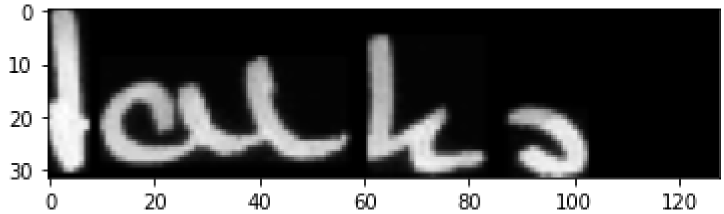
```
original_text =   boycotting
predicted text = boycotting
```



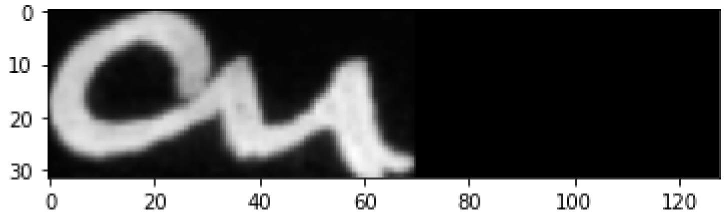
```
original_text =   London
predicted text = London
```



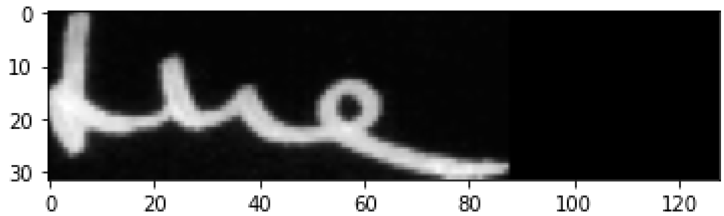
```
original_text =   talks
predicted text = talks
```



```
original_text =   on
predicted text = on
```

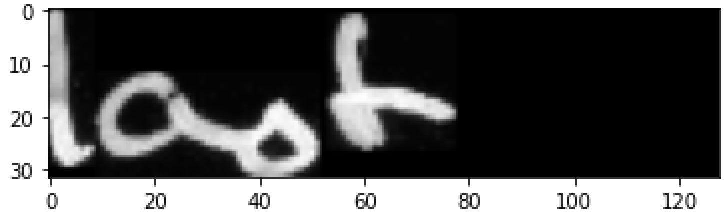


```
original_text =   the
predicted text = the
```

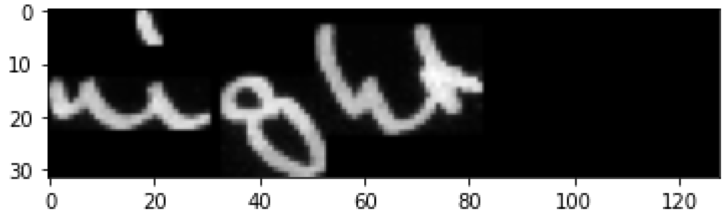


```
original_text =   last
```

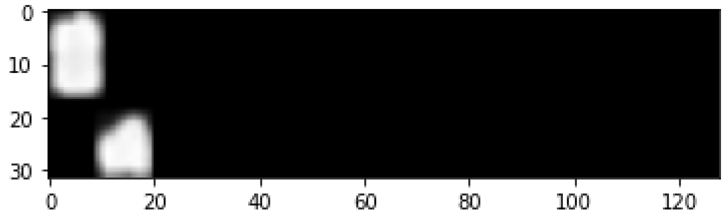
```
predicted text = last
```



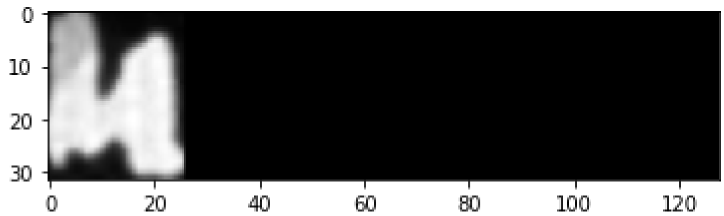
```
original_text =   night
predicted text = night
```



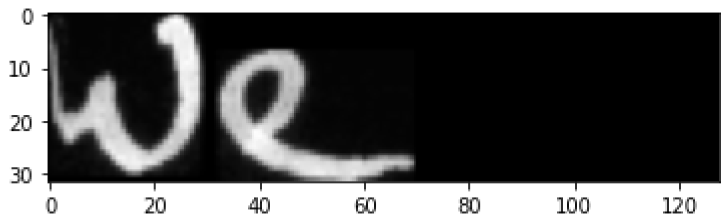
```
original_text =   :
predicted text = :
```



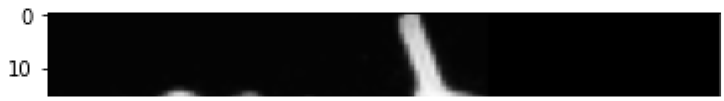
```
original_text =   "
predicted text = "
```



```
original_text =   We
predicted text = We
```



```
original_text =   want
predicted text = want
```

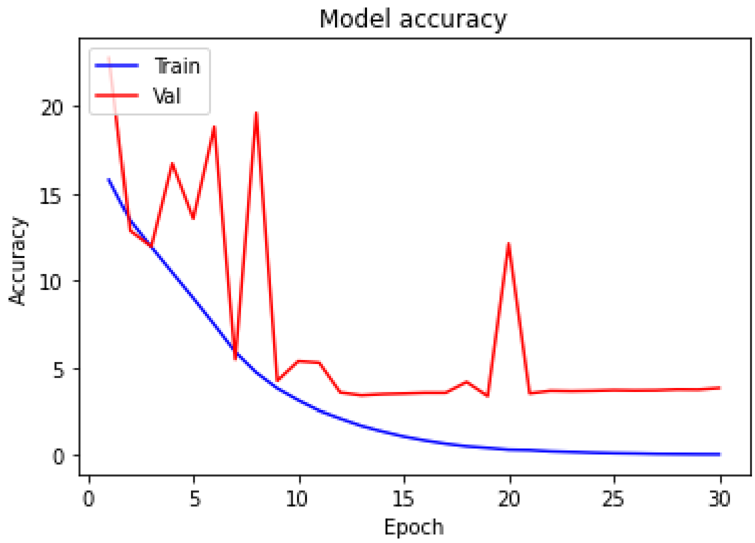


Plot Accuracy and Loss

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

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)
```

```
plotgraph(epochs, loss, val_loss)
```




```
plotgraph(epochs, acc, val_acc)
```

Get Best Model Index

```
minimum_val_loss = np.min(history.history['val_loss'])
best_model_index = np.where(history.history['val_loss'] == minimum_val_loss)[0][0]

best_loss = str(history.history['loss'][best_model_index])
best_acc = str(history.history['accuracy'][best_model_index])
best_val_loss = str(history.history['val_loss'][best_model_index])
best_val_acc = str(history.history['val_accuracy'][best_model_index])

with open('gdrive/My Drive/history.txt', 'a') as f:
    new_data = '{},{},{},{},{},{},{},{},{},{}\n'.format(filepath,
                                                         optimizer_name,
                                                         str(RECORDS_COUNT),
                                                         e,
                                                         str(train_images.shape[0]),
                                                         str(valid_images.shape[0]),
                                                         best_loss,
                                                         best_acc,
                                                         best_val_loss,
                                                         best_val_acc)
    f.write(new_data)
```

Save the Model.

```
model.save('gdrive/My Drive/Text_recognizer_Using_CRNN.h5')
```