eYSIP-2018

Character Recognition Tutorial

Authors:
Swapnil Masurekar
Abhishek Sharma

Duration of Internship: 21/05/2018 – 06/07/2018

2018, e-Yantra Publication

0.1 Overview

This tutorial is the entire description of Optical Character Recognition Application using logistic regression and convolutional neural network. Entire procedure along with code is explained in this document.

0.2 Installation Instructions:

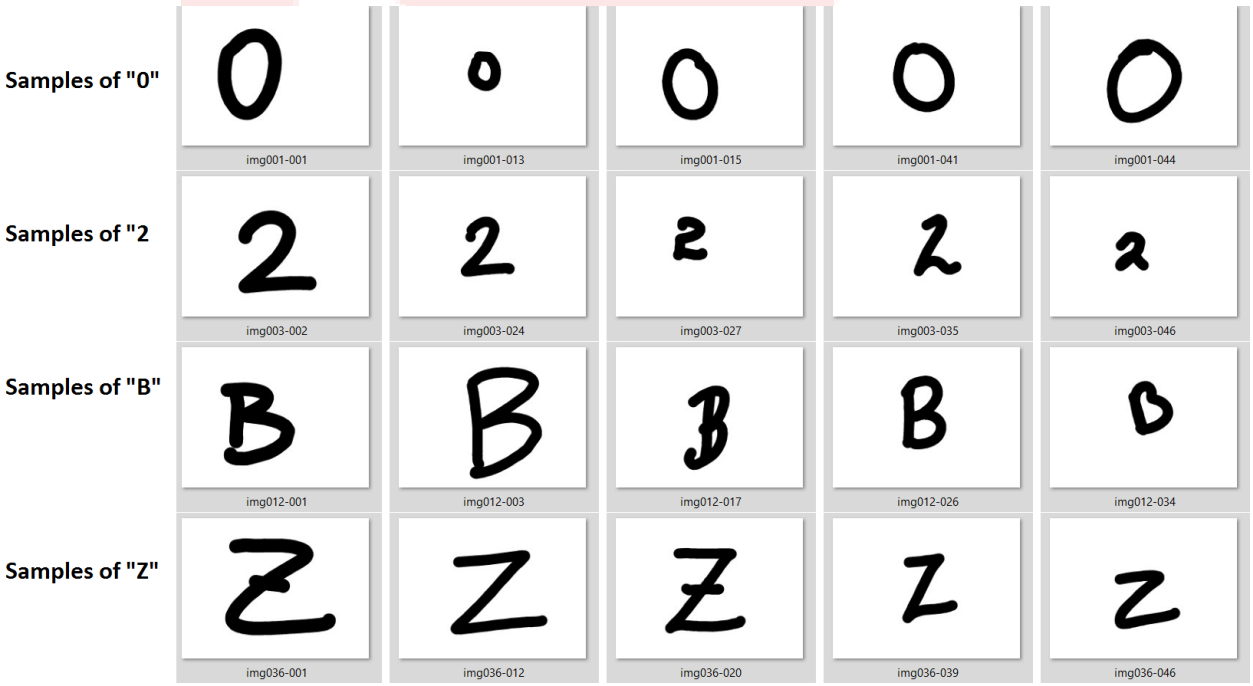
The list of libraries required for each application are: Optical Character Recognition:

- Scikit-Learn v0.19.1
- Keras v2.1.5
- Tensorflow-CPU v1.8.0
- Pandas v0.22.0
- OpenCV v3.4.1

For Installation Instructions refer this page: -

0.3 Dataset Description

Dataset of 3410 samples (55 samples of each category:



0.4 Image Preprocessing:

1. Acquiring the image bitmap in gray-scale colors.

```
imgGray=cv2.cvtColor(imgTestingNumbers, cv2.COLOR_BGR2GRAY)
```

2. Applying a Gaussian filter to the bitmap. (blur)

```
imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0)
```

3. Threshold the image irrespective of the contrast, this is done by passing "+cv2.THRESH_OTSU" as a parameter while thresholding. It automatically calculates a threshold value from image histogram for a bimodal image.

```
ret,imgThresh = cv2.threshold(imgBlurred,0,255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

4. Create a copy of the image obtained from the above method and detect contours in the image. This is done by

```
imgContours,npaContours,npaHierarchy=cv2.findContours(
    imgThresh,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
```

5. Finding the bounding boxes of external contours in the bitmap. A class named "ContourWithData()", whose object will store the information of the contours bounding rectangle. Initialize the object "cwd" of this class find the bounding rectangle for each contour holding information

```
for npaContour in npaContours:
  cwd.boundingRect = cv2.boundingRect(cwd.npaContour)
```

6. Now, in case of CNN invert the threshold image and take its gaussian blur, further the contours are cropped from this image
   Python code snippet:

```
imgThresh = self.invert(imgThresh)
imgGray = cv2.GaussianBlur(imgThresh, (5,5), 0)
```

7. Cropping the detected contours with the reference of bounding rectangle,
   CNN:

```
imgROI = imgGray[cwd.intRectY : cwd.intRectY + cwd.
    intRectHeight,
     cwd.intRectX : cwd.intRectX + cwd.intRectWidth]
```

   Logistic Regression:
8. Resizing the sub-bitmaps to 20X30 or 32x32 pixels. Here "RESIZED_IMAGE_WIDTH, RESIZED_IMAGE_HEIGHT" are global variables.

```
imgROIResized = cv2.resize(imgROI,
        (RESIZED_IMAGE_WIDTH, RESIZED_IMAGE_HEIGHT))
```

9. Unrolling the sub-bitmap matrices to feature vectors per 600 or 1024 elements.

```
npaROIResized = imgROIResized.reshape((1,
        RESIZED_IMAGE_WIDTH * RESIZED_IMAGE_HEIGHT))
```

0.5 Caching Characters Dataset:

Refer Caching_English_dataset file for caching. Download dataset from here: The Dataset with 5975 samples and "0-9,A-Z" labeled categories are preprocessed by above steps and cached into the text file which makes it computationally inexpensive while running code each time. This is done by reading each and every image from dataset, applying image-preprocessing steps and appending it to the array of flattened images.

Python code snippet:

```
npaFlattenedImage = imgROIResized.reshape((1,
    RESIZED_IMAGE_WIDTH * RESIZED_IMAGE_HEIGHT))  # flatten image
     to 1d numpy array so we can write to file later
npaFlattenedImages = np.append(npaFlattenedImages,
    npaFlattenedImage, 0)                         # add current
    flattened impage numpy array to list of flattened image numpy
     arrays
```

The classifications are appended as follow,

Python code snippet:

```
intClassifications.append(ord(decode_character(
    training_folder_number)))
```

The above steps are done for all the images together, later when "intClassifications" are obtained do these operations:

Python code snippet:

```
fltClassifications = np.array(intClassifications, np.float32)
                  # convert classifications list of ints to
    numpy array of floats

npaClassifications = fltClassifications.reshape((
    fltClassifications.size, 1))   # flatten numpy array of
    floats to 1d so we can write to file later
```

For the saving the flattened images and classifications in text file:

Python code snippet:

```
npaClassifications_og = np.loadtxt("classifications_gray.txt",
    np.float32)
npaFlattenedImages_og = np.loadtxt("flattened_images_gray.txt",
    np.float32)# read in training images
npaFlattenedImages_loadable= np.append(npaFlattenedImages_og,
    npaFlattenedImages,axis=0) # adding new images to load in
    text file
npaClassifications_loadable = np.append(npaClassifications_og,
    npaClassifications)# adding new classifications to load in
    text file
print(len(npaClassifications_loadable),len(
    npaFlattenedImages_loadable))
np.savetxt("classifications_gray.txt",
    npaClassifications_loadable)# write flattened images to text
np.savetxt("flattened_images_gray.txt",
    npaFlattenedImages_loadable)# write classifications to text
```

0.6 Logistic Regression:

Encoding the data is not required for logistic regression in scikit learn. But if you find need for encoding the dataset, you may refer "Encoding Classifications in CNN Approach".
For fitting the dataset to a mulitinomial we use the following code snippet:

```
from sklearn.linear_model import LogisticRegression
print("Training the classifier....")
classifier_local = LogisticRegression(solver='saga',multi_class=
    'multinomial',random_state = 0)# saga                    #
    instantiate LogisticRegression object to train and save model
classifier_local.fit(npaFlattenedImages, npaClassifications)
```

For testing which parameters works best you perform grid search over all the parameters. For understanding grid-search refer: Finding best model The code snippet for grid search is:

```
#     Applying Grid Search to find the best model and the best
    parameters
    from sklearn.model_selection import GridSearchCV
    parameters = [{'solver':['sag','saga','lbfgs','newton-cg'],'
    multi_class':['multinomial']},
                  {'solver':['liblinear'],'multi_class':['ovr'
    ]}]

    grid_search = GridSearchCV(estimator = classifier_local,
                               param_grid = parameters,
                               scoring = 'accuracy',
                               cv = 4,
```

```
                                    n_jobs = -1)
    grid_search = grid_search.fit(npaFlattenedImages[740:2000],
    npaClassifications[740:2000])
    print("best_accuracy = ",grid_search.best_score_)
    print("best_parameters = ",grid_search.best_params_)
    print(score_matrix)
    joblib.dump(classifier_local, filename)
```

Grid Search results of the optimizer are as shown below:

Solver	Validation Accuracy(%)
newton-cg	84.8516%
lbfgs	85.1402%
liblinear	83.6721%
sag	85.2901%
saga	85.5486%

As seen the saga optimizer has the highest cross-validation accuracy. Hence, this approach involves multinomial class classification using saga optimizer.

0.7 CNN Approach:

The required libraries for this approach are as shown below, some more packages would be included later during the course of tutorial:

```
print("Importing Libraries....")
import cv2
import numpy as np
import operator
import os
import keras
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd # to load the dataset
```

Load the data:

Python code snippet:

```
npaClassifications = np.loadtxt("classifications_gray.txt", np.
    float32)                  # read in training classifications
npaFlattenedImages = np.loadtxt("flattened_images_gray.txt", np.
    float32)                 # read in training images
```

The data need to be shuffled for the validation purposes, which would be explained later,

```
from sklearn.utils import shuffle
npaFlattenedImages,npaClassifications=shuffle(npaFlattenedImages
    ,npaClassifications)
```

0.7.1 Encoding Classifications:

Feature scaling is needed if Real time data Augmentation is **not** used. This would be better understood while creating the ImageGenerator Object. To understand the need of feature scaling refer "ML_Coding_Tutorial".

```
# Feature Scaling
# Required when no image generator is used
from sklearn.preprocessing import StandardScaler
sc_npaFlattenedImages = StandardScaler()
npaFlattenedImages = sc_npaFlattenedImages.fit_transform(
    npaFlattenedImages)
```

As classification label is an categorical variable, label encoding is required, and as the output layer of CNN consists of one neuron for each classification. Vectorization is needed. Hence we use "Onehotencoder" for converting labels into one-hot-encoded vector. Refer "ML_coding_Tutorial.pdf" for understanding the need for encoding categorical data. This is done by following code snippet:

```
# Encoding done for Classifications
npaClassifications = npaClassifications.reshape((
    npaClassifications.size, 1))       # reshape numpy array to 1
    d, necessary to pass to call to train
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_npaClassifications = LabelEncoder()
                        # Label encoding not needed in this case
    as classifications are ASCII numeric values, But here we use
    label encoder to later use inverse transform method for
    decoding purposes
npaClassifications[:, 0] = labelencoder_npaClassifications.
    fit_transform(npaClassifications[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])
npaClassifications = onehotencoder.fit_transform(
    npaClassifications).toarray()
```

0.7.2 Real Time Data Augmentation:

For inputing the data into CNN some the shape of the input needs to to manipulated, it is done be,

```
images2D=[]
# Reshape flattened gray images to 30x20 2D array and store in
    list
for npaFlattenedImage in npaFlattenedImages:
    images2D.append(npaFlattenedImage.reshape(
    RESIZED_IMAGE_HEIGHT,RESIZED_IMAGE_WIDTH,1))
images2D=np.array(images2D)# converting list to 3D numpy array
```

Now, for real-time data augmentation, ImageGenerator object needs to be created and our data must be fitted to this object, for more information refer Keras Image Preprocessing

```
datagen = ImageDataGenerator(
            featurewise_center=True,
            featurewise_std_normalization=True,
            rotation_range=40, # can be 45, 60, 90
                              )
datagen.fit(images2D)
```

0.7.3 Creating the CNN:

The convolutional and fully connected layer's perceptrons are associated with Rectifier activation function while as the output layer is associated with softmax activation which gives probabilistic distribution in output layer. This sequential CNN model can be build by the following steps:

```
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten

classifier_cnn = Sequential()

    # Step 1 - Convolution
    classifier_cnn.add(Convolution2D(32,# number of filters
                                     3, 3,# shape of filter
                                     input_shape = (
   RESIZED_IMAGE_HEIGHT,RESIZED_IMAGE_WIDTH,1) ,# shape of input
                                     activation = 'relu'
                    # Rectifiler activation for convolution layer
                                     ))
    # Step 2 - Pooling
    classifier_cnn.add(MaxPooling2D(pool_size = (2, 2)))
                    # Max-pooling

    # Adding a second convolutional layer
    classifier_cnn.add(Convolution2D(64, 3, 3, activation = '
   relu'))
    classifier_cnn.add(MaxPooling2D(pool_size = (2, 2)))

    # Step 3 -  Re-Flattening
    classifier_cnn.add(Flatten())

    # Step 4 - Full connection
    classifier_cnn.add(Dense(output_dim = 1000, activation = '
   relu'))

```

```
    # Step 5 - Adding output layer
    classifier_cnn.add(Dense(output_dim = len(npaClassifications
    [0]), activation = 'softmax'))

    # Compiling the CNN
    classifier_cnn.compile(optimizer = 'adam', loss = '
    categorical_crossentropy', metrics = ['accuracy'])
```

The model Summary is as shown,

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_27 (Conv2D)           (None, 30, 30, 32)        320
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
flatten_16 (Flatten)         (None, 2304)              0
_________________________________________________________________
dense_35 (Dense)             (None, 1000)              2305000
_________________________________________________________________
dense_36 (Dense)             (None, 36)                36036
=================================================================
Total params: 2,359,852
Trainable params: 2,359,852
Non-trainable params: 0
```

For understanding which loss function to use watch Deep Learning Section is this youtube playlist Machine Learning A-Z (Complete Course on ML)

0.7.4 Training the model:

Refer Keras Sequential model API for more details.
If Real time data augmentation is not used:

```
    # Fitting CNN
    classifier_cnn.fit(images2D, npaClassifications,
                       validation_split = 0.08, # here in
    validation_split the split is not random it always take last
    10% of the data, hence shuffle before fitting
                       batch_size = 50,
                       nb_epoch = 25)
```

If Real time data augmentation is used:

```
    classifier_cnn.fit_generator(datagen.flow(images2D,
    npaClassifications, batch_size=32),
                        steps_per_epoch=len(images2D) / 32,
    epochs=50)
```

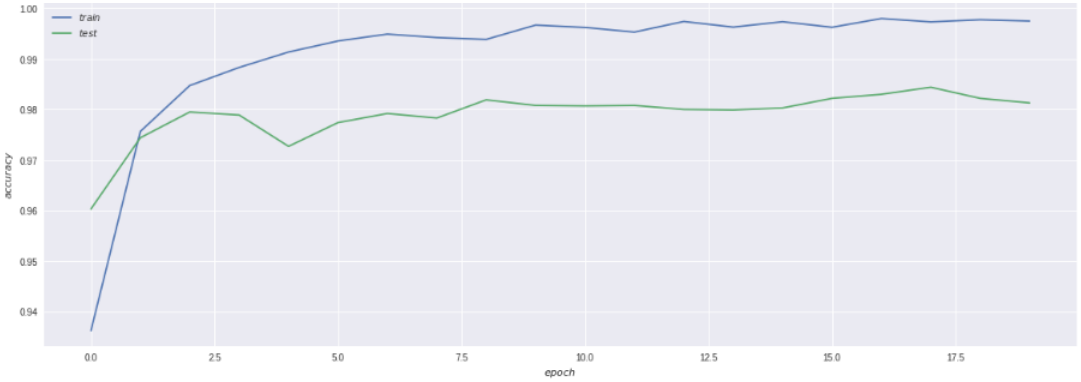
For **saving** the model:

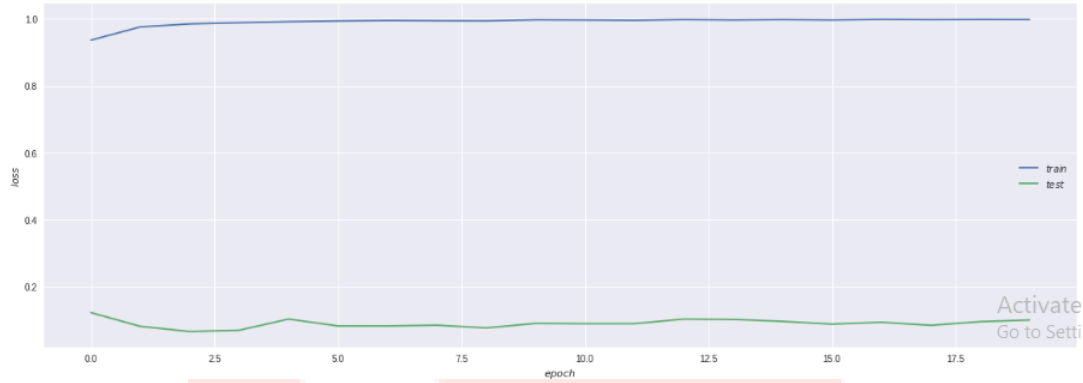
```
filename='Convolutional_Neural_network_model.h5'
classifier_cnn.save(filename)
```

For plotting the history of training use this function (use your model.fit method)

```
def fit(model, Xtrain, ytrain, Xtest, ytest, *args, **kwargs):
    history = model.fit(Xtrain, ytrain, validation_data = (Xtest
    , ytest), *args, **kwargs)# Use your fit method here

    fig, axes = pplt.subplots(2)
    fig.set_size_inches(20, 15)

    axes[0].plot(history.history['acc'])
    axes[0].plot(history.history['val_acc'])
    axes[0].set_xlabel('$epoch$')
    axes[0].set_ylabel('$accuracy$')
    axes[0].legend(['$train$', '$test$'])

    axes[1].plot(history.history['acc'])
    axes[1].plot(history.history['val_loss'])
    axes[1].set_xlabel('$epoch$')
    axes[1].set_ylabel('$loss$')
    axes[1].legend(['$train$', '$test$'])

    return axes, model
```

This is an example plot, your plot may differ from this,





0.8 Predicting the Results:

0.8.1 CNN:

For the test image perform all the image preprocessing step and you need to predict output for each cropped ROI. This done by,
In case of Real-time Data augmentation:

```
npaROIResized=npaROIResized.reshape(1,RESIZED_IMAGE_HEIGHT,
    RESIZED_IMAGE_WIDTH,1)
datagen.fit(npaROIResized)
y_pred = classifier_cnn.predict_generator(datagen.flow(
    npaROIResized))
```

Otherwise,

```
npaROIResized=sc_npaFlattenedImages.transform(npaROIResized)
npaROIResized=npaROIResized.reshape(1,RESIZED_IMAGE_HEIGHT,
    RESIZED_IMAGE_WIDTH,1)
y_pred = classifier_cnn.predict(npaROIResized)
```

Now "y_pred" is a 1D vector which holds the probabilities for each category, the category with highest probability is the actual predicted classification. To find the index of highest probability we use "np.argmax" function and labelEncoder object which we created earlier to inverse transform the value obtained from "argmax". Thus we get the actual prediction.

```
y_pred_inverted = labelencoder_npaClassifications.
    inverse_transform([np.argmax(y_pred[0, :])])
```

Above is the prediction for single contour, this needs to be done for all the detected contours.

Useful ML Resources:

- Text Documents
    1. Machine Learning and Algorithms Here you will get the brief idea about ML and Its basic learning algorithm
    2. Basic Terminology related to ML Here you will learn about the basic terminologies and jargons used in ML and Neural Network
    3. Artificial Neural network A Quick introduction to ANN
    4. Convolution Neural network Introduction to Architecture of CNN
- For video tutorial over machine learning refer to following links
    1. Machine learning course by Andrew ng - Click here (enroll for free)
    2. Machine learning A-Z Hands on python in data science - click here(refer tutorials for Python only)
    3. Recurrent Neural Network by Andrew ng - Click here(refer video till LSTM only)
    4. Deep learning by Andrew ng - Click here
    5. Convolution Neural Network by Andrew ng - Click here