MEF UNIVERSITY

PRE-OCR IMAGE OPTIMIZATION BY REINFORCEMENT LEARNING

Capstone Project

Cihan Tektunalı

İSTANBUL, 2018

MEF UNIVERSITY

PRE-OCR IMAGE OPTIMIZATION BY REINFORCEMENT LEARNING

Capstone Project

Cihan Tektunalı

Advisor: Prof. Dr. Muhittin Gökmen

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Pre-OCR Image Optimization By Reinforcement Learning
Name/Last Name of the Student: Cihan Tektunalı
Date of Thesis Defense: 10/9/2018

I hereby state that the graduation project prepared by Cihan Tektunalı has been completed under my supervision. I accept this work as a "Graduation Project".

10/9/2018
Prof. Dr. Muhittin Gökmen

I hereby state that I have examined this graduation project by Cihan Tektunalı which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

10/9/2018

Prof. Dr. Özgür Özlük
Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of _____ and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Prof. Dr. Muhittin Gökmen	……
2. Prof. Dr. Özgür Özlük	……

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name Date Signature

EXECUTIVE SUMMARY

PRE-OCR IMAGE OPTIMIZATION
BY REINFORCEMENT LEARNING

Cihan Tektunalı

Advisor: Prof. Dr. Muhittin Gökmen

SEPTEMBER, 2018, 25 pages

Optical Character Recognition technology usage in digital transformation of documents is steadily growing by the help of new hardware and software technologies. However digital image optimization for more accurate OCR results continues to be a problem. In this study, we propose a reinforcement learning based model that learns optimal set of actions to increase OCR accuracy in computer screenshot images. Model input images are identified by their grayscale histogram distributions. An unprocessed base image having 100% OCR accuracy is taken initially. The correlation between the grayscale histograms of base image and input image is used for comparison. We implemented reinforcement learning's random (or optimal) action and reward approach for creating a Q-table. For measuring image to text conversion success, Tesseract OCR software is used. The introduced approach can improve OCR accuracy especially in bulk image to document conversion jobs. By using optimal actions for single image or bulk images, it can also decrease computational load and time-consumption in image processing.

Key Words: OCR accuracy optimization, reinforcement learning, q-table learning, increasing bulk image OCR accuracy

ÖZET

TAKVİYELİ ÖĞRENME İLE OPTİK KARAKTER
TANIMA ÖNCESİ GÖRÜNTÜ OPTİMİZASYONU

Cihan Tektunalı

Tez Danışmanı: Prof. Dr. Muhittin Gökmen

EYLÜL, 2018, 25 sayfa

Metinsel dokümanların sayısal ortama aktarılmasında optik karakter tanıma teknolojisinin kullanımı donanım ve yazılım alanındaki gelişmelerin yardımıyla giderek artmaktadır. Bununla birlikte karakter tanımanın daha yüksek başarıyla yapılabilmesi için sayısal görüntü optimizasyonu bir problem olmaya devam etmektedir. Bu çalışmada bilgisayar ekran görüntülerinden karakter tanıma başarısının arttırılması için sayısal görüntü optimizasyonu yapan ve takviyeli öğrenme yöntemini kullanan bir model öne sürülmüştür. Modele girdi olarak verilen sayısal görüntülerin gri ton dağılımları görüntü durumlarını tanımlamak için kullanıldı. Ham haliyle tam başarılı karakter tanıma yapılabilen bir görüntü baz alındı. Verilen yeni görüntüler ile baz alınan görüntünün gri ton dağılımı arasındaki korelasyon değeri görüntüleri karşılaştırmak için kullanıldı. Takviyeli öğrenme ile uygulanan rastgele veya optimal aksiyon dizileri ve sonuç olarak elde edilen ödül değerleri kullanılarak Q-tablosu oluşturuldu. Görüntünün metne çevrilme başarısının ölçümü için Tesseract OCR yazılımı kullanıldı. Oluşturulan bu model ile özellikle sayısal ortama toplu aktarım işlemlerinde karakter tanıma verimi arttırılabilir. Ayrıca görüntü bazında veya tüm görüntü kümesinde optik karakter tanıma iyileştirmesi sağlayacak optimal aksiyonlar kullanılarak toplamdaki hesaplama yükünün ve görüntü işlemede kaybedilen zamanın azaltılması sağlanabilir.

Anahtar Kelimeler: Optik karakter tanıma başarım optimizasyonu, takviyeli öğrenme, Q-tablosu ile öğrenme, çoklu görüntülerin optik karakter tanıma başarımının arttırılması

TABLE OF CONTENTS

.

1. INTRODUCTION

Identification of machine printed text which is named as Optical Character Recognition (OCR) technology and its conversion process to digital medium has been increasing its significance by the help of improvements in the computer hardware and software industry, even it has been around for more than 10 years. In order to acquire successful results in digital conversion of printed texts by OCR, image parameter optimizations are generally required. Because character recognition in the digital medium requires image processing knowledge and expertise (such as defining scope, order and parameters of image filters being applied), this process still needs an expert's intervention at least in some stages, making it demanding and time consuming especially if there is a bulk conversion scenario with tens of thousands of documents. At this point, reinforcement learning techniques that is a branch of machine learning, come to rescue for creating a self-learning model to optimize image processing steps for conversion before the application of OCR software. In this study, we used reinforcement learning's Q table approach to create a software that tries to find the best image processing options for the given images to increase the character recognition accuracy of OCR software. To measure image to text conversion accuracy, we used the recognized outputs of Tesseract Open Source OCR software (version: 3.05.02).

As data inputs to our self-learning model, we created arbitrary cropped samples of computer screenshot images. For simplification purposes, these images included one to seven words that needed to be recognized by OCR software. The content language of words is in either English or Turkish.

1.1. What is Reinforcement Learning?

Reinforcement learning can be defined as learning to reach a specific target by interacting with the medium that the learner (named agent) is in and gathering a reward (or punishment) back as a result of this interaction. By this way, the learner finds out which actions to take to maximize the reward and which actions not to take to minimize the punishment. Apart from other machine learning branches, named as supervised learning (function estimation using previously labeled data) and unsupervised learning (estimating

groups using unlabeled data), reinforcement learning differs in terms of learning method, which aims at maximizing the long-term reward, by taking various actions. To explore the action space and deal with local minimum problem in machine learning, the action selection is made either randomly or based on the maximum reward, adjusted by a threshold value. As Sutton and Barto (2017) described in a general sense, six members can be defined as part of a reinforcement learning system:

- Agent (the learner)
- Environment
- Policy
- Reward signal
- Value function
- Model of the environment (optional)

In this study, a specialized method called Q-learning is selected for our optimization problem.

Figure 1: Diagram for basic Q learning procedure

Q-learning is an algorithm used in reinforcement learning space which provides a solution to Markov Decision Process model. It uses an action – value function to define the value (reward) of an action applied to each state of the agent (Figure 1). It is simply an off-policy algorithm, which means that the optimal action-value function estimation is unrestricted by the policy in use. As Maini (2017) mentioned, the action-value function is defined as:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot \left(\overbrace{\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{estimate\ of\ optimal\ future\ value}}^{learned\ value} - \underbrace{Q(s_t, a_t)}_{old\ value} \right). \quad (1)$$

Here Q is the calculated value after taking action *a* at state *s* and time *t*,

α is the learning rate,

r_t is the reward at time *t*,

and γ is the discount factor.

We try to find the optimal action that maximizes Q value for current state. Before Q-table is updated, the weight of new reward value is adjusted by its previous value, discount factor, optimal future value and learning rate.

1.2. Literature Review

To enable more accurate text conversion results in OCR, different approaches are used. As El Harraj and Raissouni (2015) stated, some of these techniques depend on correcting character or word based errors by comparing with a dictionary or using the scope of corresponding text to find the true version of the inaccurate part. In the case of digital camera-based OCR, Ezaki *et al.* (2004) created a system that tries to find the regions with tiny characters in image and then zooms to the selected region for character recognition. They propose that the success rate of various techniques is connected with the dimension of characters. They also listed the order of the best performing operations as Sobel edge detection, Otsu binarization, connected component extraction and rule-based connected component filtering with the emphasis on the bigger characters. Some of the further procedures used in image optimization includes image processing techniques such as greyscaling, scaling, blurring, denoising, erosion, dilation, histogram equalization, adaptive thresholding, edge detection etc. as Mordvintsev and Abid (2017) listed in the documentation of OpenCV library. The list of actions we used in this study is generated from a subset of essential processes applied in pre-processing step of reviewed projects about Optical Character Recognition (Alginahi, 2010; Chiatti *et al.*, 2018; El Harraj & Raissouni, 2015; Hamad & Kaya, 2016; Huang *et al.*, 2014; Janvalkar *et al.*, 2014; Taylor & Wolf, 2004).

2. EMPIRICAL DATA

As our dataset, we created 211 small sized color images in Portable Network Graphics (png) format, cropped from computer screenshots. The screenshots were taken from Windows and Linux operating systems. Image sizes differ from 29 to 334 pixels in width and 20 to 216 pixels in height. Training set is created with roughly half of the set (106 images) by random sampling. The rest of the set is used for testing. We decided to represent each image by its histogram values. Histograms represented the features in our design. In order to simplify the feature set and also as an OCR best practice, we preferred to use grayscale histograms. No pre-processing is applied to images before they are fed into our Q-learning algorithm. Some of the samples that we used are displayed in Figure 2.

Figure 2: Sample Input Images

3. PROJECT DEFINITION

3.1. Project Objectives

Image to text conversion by OCR is a time-consuming process. Furthermore, OCR software accuracy is not stable and can change according to image parameters like quality, resolution, character size, or scale. This may lead to human intervention for problematic images at multiple stages for correction.

The main objective of this study was to create a self-learning system that optimizes cropped screenshot images before sending to OCR software for acquiring more successful results. By using Q-learning approach, we created a model that calculates OCR accuracy using Tesseract software and tried to increase the accuracy by applying different processes selected from predefined actions list. The second objective was to fine-tune processing actions or parameters to further increase OCR accuracy of the whole test set.

3.2. Project Scope

Because computer printed text existing in screenshot images have no background noise compared to other digital image sources like scanner or digital camera, our optimization scenario does not consider background noise reduction operation. Also there were no document orientation problem which may require skewness correction as in the scanned document cases.

During data preparation step, since image processing demands high computational power, sample image sizes were limited to maximum 300 pixels in width and 50 pixels in height. In addition to that, preparing cropped images from computer screenshots and listing image file names along with their content strings was a time-consuming process. This led to a limited training data size. Our source data consisted of 106 images (out of 211 images in total). Thus the resulting trained Q-table model has a limited variability. That can be considered as a negative effect in the way of gaining more accurate results with this model.

At implementation stage, we decided to use grayscale histogram correlations with the base image as state values in Q-table approach. Only the initial correlation value is used for each image in updating Q-table. As a result, number of states are limited to unique correlation values of input images with the base image.

4. METHODOLOGY

For a self-learning system, we adapted Q-learning algorithm from the group of reinforcement learning techniques. Python code is written for implementing Q-table approach, applying image processing functions, training and testing image data (see Appendices A, B and C). We used a base image as a reference point at model (Q-table) training. The basic members required by Q-learning are mapped as follows:

- Environment: Base image and Input images
- Agent: Python code
- States: Correlation coefficient between the base image and the input image that ranges between -1 and 1 with 0.01 intervals
- Actions: Image processing operations (like scaling, black and white conversion, denoising etc.) defined as a list

The set of actions in our model with their weighted selection probability p in probability distribution is as follows:

- Scale image width and height by 2 times, p = 4/14
- Scale image width and height by 4 times, p = 2/14
- Black and white conversion, p = 2/14
- Denoising, p = 2/14
- Erosion, p = 1/14
- Dilation, p = 1/14
- Histogram equalization, p = 2/14

The probabilities we defined above were the initial values during model training and test. After test results observed, as a fine-tuning attempt the action set was narrowed down to 4 actions that rewards the most.

General strategy of the model is as follows: The agent first reads base image which represents default state value (the correlation of the base image with itself calculated from grayscale histogram equals 1). An empty Q-table is created and updated with default state

value. The agent then starts iterating through the input image set and calculates the correlation between base image and input image. At the next step, it tries random actions (or the optimal action set if exists) from the list of image processing options. At each iteration it updates the table with state, action set and adjusted reward value using OCR accuracy from Tesseract software's image to text conversion result. As the original text content is known, OCR accuracy A for the current state of the image is calculated by:

$$A = N / T \tag{2}$$

Here, N is the number of correctly recognized characters and T is the total number of characters in the image.

At the end of all iterations, we acquire a table of image states (rounded correlation values relative to base image) and corresponding ordered actions list applied so far, along with the reward values calculated by action-value formula.

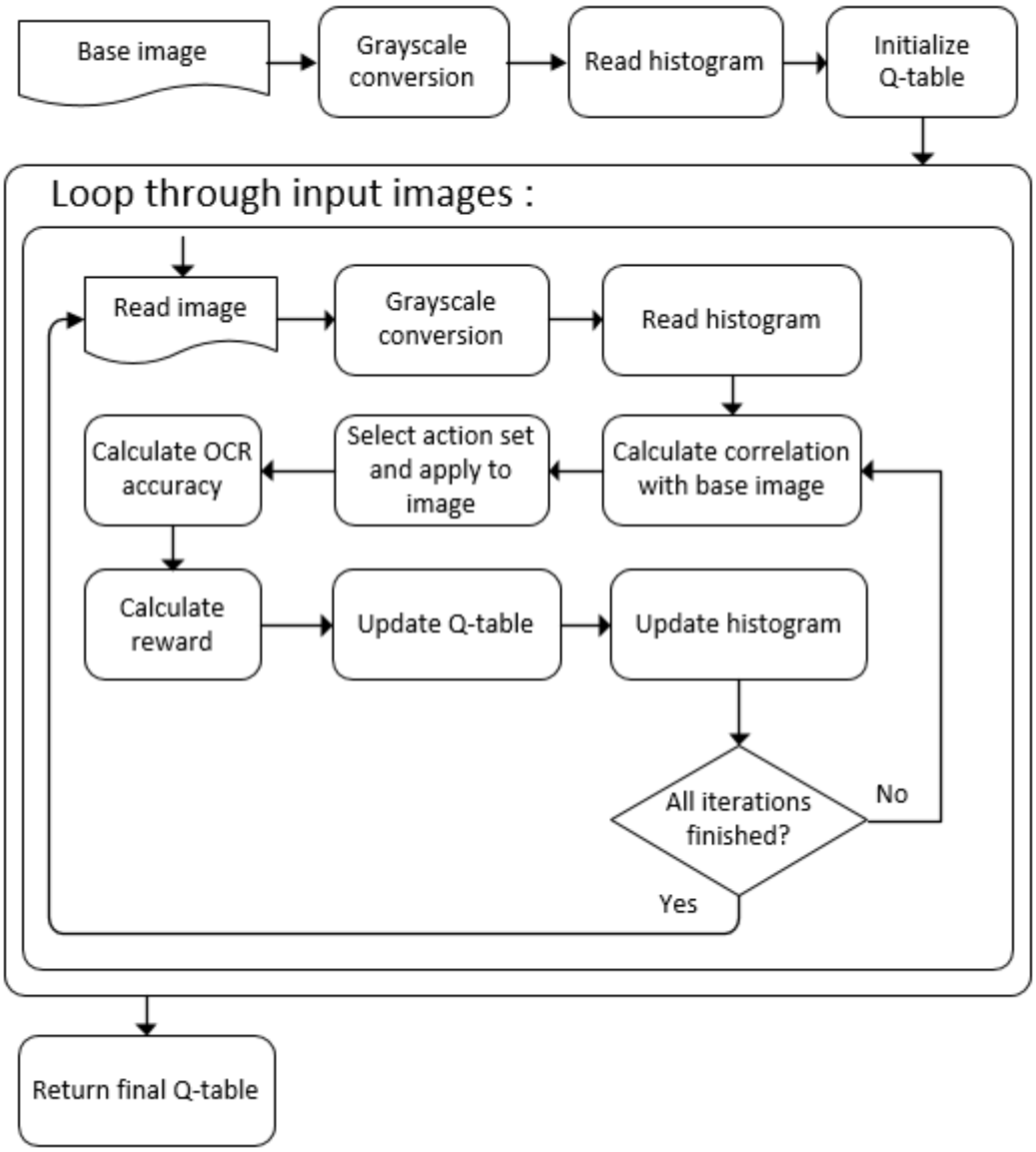

Figure 3: Diagram for Q learning procedure adapted to image optimization

We trained our Q-table by the adapted procedure (Figure 3) using 100, 500, 1000, 2500 iterations per image with training image set (see Appendix B). At next step, we exposed each image in test set to trained Q-tables and collected accuracy values (see Appendix C). Since the algorithm is able to choose random actions even if there is an optimal action set having the maximum reward, we preferred to average the accuracy scores after 100 iterations per image to reach more stable results.

5. RESULTS

In the final part of this study, we compared average OCR accuracy values of images in test set (Table 1). Raw accuracy values were obtained by direct exposure to Tesseract software without any image processing. Other results were taken after making use of Q-tables trained with different number of iterations. The same Q-table algorithm was employed for acquiring accuracy value.

Table 1

Average OCR Accuracies of Images in Test Set

Raw	Q-table 100 iterations	Q-table 500 iterations	Q-table 1000 iterations	Q-table 2500 iterations
0.296	0.345	0.348	0.352	0.351

The results show that the average OCR accuracy of all images in test set has risen approximately 5% with 100 times iterated Q-table model. The iterations over 100 brings slight additional improvements to the model. The model results seem to converge between 500 to 1000 iterations. Comparison with raw accuracies is shown in Figure 4.

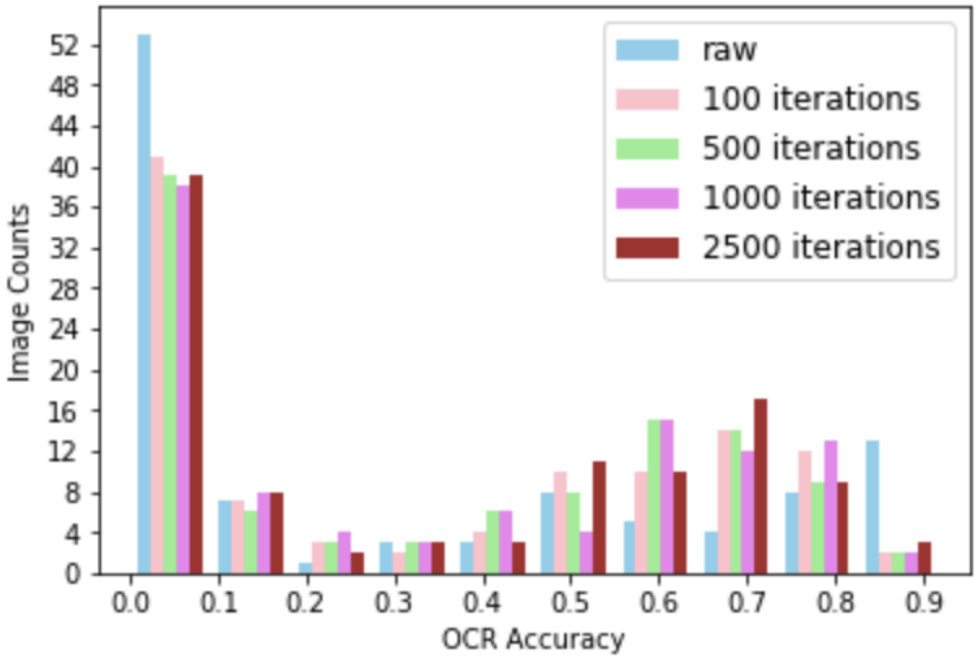

Figure 4: Average OCR Accuracy Distribution of Test Images Before and After Q-table Exposure

The bar plot in Figure 4 displays the effect of iteration groups on accuracy values grouped in 0.1 intervals. Trained Q-table exposure brings significant increases in image groups with 0.6, 0.7 and 0.8 accuracy values whereas there is a steep fall in all iteration groups compared with raw accuracy value in 0.9 interval. This implies that some images which already have high OCR accuracies in unprocessed states need to be treated differently. To provide with an additional improvement to current algorithm, all raw image accuracies can be checked and treated accordingly before Q-table exposure.

As a tuning attempt to this model, we reviewed top 5 highest rewarded actions from each Q-table and selected the most significant 4 actions. We just applied these single processes to all test set instead of random selection from a pool. The results are shown in Table 2.

Table 2

Average OCR Accuracies of Images in Test Set - After Single Process Applied

Action	Scale 2x	Black and White Conversion	Denoising	Histogram Equalization
Accuracy	0.378	0.296	0.285	0.143

As seen in Table 2, even a single action like scaling can boost OCR accuracy scores more than an entire trained Q-table. The strategy for further optimization of this model may consider searching the best single actions from a much broader set of actions and apply as few of them as possible. As the number of actions increase, the minimum number of iterations required for convergence will rise. So this should also be taken into consideration for optimization attempts.

6. FUTURE RESEARCH

In this project, we targeted to increase optical character recognition accuracy of a set of screenshot images by incorporating a self-learning model. For the sake of limited resources, we defined some constraints in our machine learning model. These can be summarized as the number of images used to train and test, source of images (computer screenshots or scanned images etc.), image width – height ranges, number of words in image texts, number of image processing actions, probability weights of actions, iteration counts in terms of training model and averaging test results, parameters of Q-learning model and reward calculation formula. In future improvements to this model, these constraints can be redesigned according to related scenario. Besides that, new approaches can be adapted to Q-learning algorithm for a more robust image accuracy control.

REFERENCES

Alginahi, Y. (2010). Preprocessing Techniques in Character Recognition, Character Recognition Minoru Mori, IntechOpen, DOI: 10.5772/9776. Available from: https://www.intechopen.com/books/character-recognition/preprocessing-techniques-in-character-recognition

Chiatti, A., Cho, M. J., Gagneja A., Yang X., Brinberg M., Roehrick K., … Giles C. L. (2018). Text Extraction and Retrieval from Smartphone Screenshots: Building a Repository for Life in Media. Retrieved from https://arxiv.org/pdf/1801.01316.pdf

El Harraj, A. and Raissouni, N. (2015). OCR Accuracy Improvement On Document Images Through a Novel Pre-processing Approach. *Signal & Image Processing: An International Journal (SIPIJ), 6(4).* DOI: 10.5121/sipij.2015.6401

Ezaki, N., Bulacu M., and Schomaker, L. Text Detection from Natural Scene Images: Towards a System for Visually Impaired Persons. *In Proc. of 17th. Int. Conf. on Pattern Recognition (ICPR 2004),* IEEE Computer Society, 2004, pp. 683-686, vol. II, 23-26 August, Cambridge, UK

Hamad, K.A., Kaya, M. (2016). A Detailed Analysis of Optical Character Recognition Technology. *International Journal of Applied Mathematics, Electronics and Computers*, 4(Special Issue), 244–249. Retrieved from http://dergipark.gov.tr/download/article-file/236939

Huang C., Chuang Y., Chang R. & Chen Y. (2014). Enhance Text Recognition by Image Pre-Processing to Facilitate Library Services by Mobile Devices. *In Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART-2014), pp. 453-460.* DOI: 10.5220/0004818504530460

Janvalkar, S., Manjrekar, P., Pawar, S., & Naik, L. (2014). Text Recognition from an Image. *International Journal of Engineering Research and Applications*, Vol. 4, Issue 4 (Version 5), April 2014, pp.149-151. Retrieved from http://ijera.com/papers/Vol4_issue4/Version%205/Z04405149151.pdf

Maini, V. (2017). Machine Learning for Humans, Part 5: Reinforcement Learning. Retrieved from https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

Mordvintsev, A. & Abid, K. (2017). *OpenCV-Python Tutorials Documentation (Release 1)* [PDF file]. Retrieved from https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf

Sutton, R.S. & Barto, A.G. (2017). Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press.

Taylor, G.W. & Wolf, C. (2004). Reinforcement learning for parameter control of text detection in images from video sequences. 2004 *International Conference on Information and Communication Technologies: From Theory to Applications*, 517-518. Retrieved from http://www.uoguelph.ca/~gwtaylor/publications/icict2004.pdf

APPENDIX A

Python Code for Creating Train and Test Sets

```
#Split train-test data by reading data information file
def TrainTestSplit():
    #Read data from file, write to lists and split train-test sets
    names=[]
    contents=[]
    langs=[]
    count=0
    with open('list.csv', encoding='ISO-8859-9') as f:
        data = csv.reader(f)
        for row in data:
            #print(row[2])
            count+=1
            if(count>1):
                names.append(row[1])
                contents.append(row[2])
                langs.append(row[3])

    langs=['tur' if l=='tr' else l for l in langs]

    #Create train and test sets
    train_indexes=random.sample(range(0, 211), 106)
    #Filename list
    train_names=[names[i] for i in train_indexes]
    test_names =[names[i] for i in range(0, 211) if i not in train_indexes]
    #Text content of files
    train_contents=[contents[i] for i in train_indexes]
    test_contents =[contents[i] for i in range(0, 211) if i not in train_indexes]
    #Text language of files
```

```
    train_langs=[langs[i] for i in train_indexes]
    test_langs =[langs[i] for i in range(0, 211) if i not in train_indexes]

    return train_names,train_contents,train_langs,test_names,test_contents,test_langs

# Split training and test data and write them to separate files
    #Get split data
    train_names,train_contents,train_langs,test_names,test_contents,test_langs =
        TrainTestSplit()

    #Write train data to file
    with open("train.txt","w",encoding='utf-8') as f:
        for i,n in enumerate(train_names):
            f.write(n+"\t"+train_contents[i]+"\t"+train_langs[i]+"\r\n")

    #Write test data to file
    with open("test.txt","w",encoding='utf-8') as f:
        for i,n in enumerate(test_names):
            f.write(n+"\t"+test_contents[i]+"\t"+test_langs[i]+"\r\n")
```

APPENDIX B

Python Code for Training Q-Table Model

```
#import required libraries
import random
import numpy as np
import pandas as pd
import cv2
import pytesseract
import glob
import PIL
import time
import csv
import statistics as st

#Set initial values
iterations=100
#Greedy Threshold
GRDT=0.5
#Discount Factor
DF=0.9
#Learning Rate
LR=0.1

#Calculate Tesseract accuracy using iamge, groundtruth string and language code
def getTesseractAccuracy(image, groundtruth_string, lang_code):
    tesser_out = pytesseract.image_to_string(image, lang=lang_code)
    true_prediction_count=0
    n=0
    if len(groundtruth_string)< len(tesser_out):
        n=len(groundtruth_string)
```

```
    else:
        n=len(tesser_out)

    for i in range(n):
        if tesser_out[i]==groundtruth_string[i]:
            true_prediction_count+=1

    if n>0:
        return true_prediction_count/n
    else:
        return 0
#Get similarity value(correlation) between two image histograms
def getSimilarity(source_hist, target_hist):
    similarity = cv2.compareHist(target_hist, source_hist, cv2.HISTCMP_CORREL)
    return similarity;

#Generate actionset with randomized actions and weighted probabilities
def getRandomActions(min,max,n_min,n_max):
    #select size of actions sample size from distribution with given probabilities
    a= np.random.choice(list(range(min, max)), np.random.randint(n_min,n_max,1),
replace=False, p=[4/14, 2/14, 2/14, 2/14, 1/14,1/14,2/14])
    a2=''.join(map(str, a))
    return a2

#Generate random or best reward action set based on random value and greedy threshold
value
def getActionList(state, table):
    actionset_maxreward=0
    maxreward=0
    #Get max reward actionset for given row's correlation value
    for index, row in table.iterrows():
        if table['Correlation'][index]==state:
```

```
            actionset_maxreward = row[1:].idxmax()
            maxreward= row[1:].max()

    #Random actionset or greedy
    if (np.random.uniform() > GRDT) or (maxreward==0):
        actionset=getRandomActions(1,8,0,6)
    else:
        actionset = actionset_maxreward


    #if both action 1 (scale 2x) and 2 (scale 4x) exists, remove one of them randomly
    if '1' in actionset and '2' in actionset:
        actionset=actionset[actionset!=np.random.randint(1,3,1)[0]]


    return str(0) if len(actionset)==0 else ''.join(str(actionset))


#Apply selected processes to image, return OCR accuracy of processed image
def getFeedback(im, Actions, groundtruth_string,lang_code):
    im_current=im
    img_proc=im
    accuracy=0

    try:
        for i in Actions:

            if i=='1':#scale 2x
                img_proc = cv2.resize(im_current,None,fx=2, fy=2, interpolation =
cv2.INTER_CUBIC)
            if i=='2':#scale 4x
```

```
            img_proc = cv2.resize(im_current,None,fx=4, fy=4, interpolation = cv2.INTER_CUBIC)
        if i=='3':#bw
            (thresh, img_proc) = cv2.threshold(im_current, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        if i=='4':#denoising
            img_proc = cv2.fastNlMeansDenoising(im_current,None,10,7,21)
        if i=='5':#erosion
            kernel = np.ones((2,2),np.uint8)
            img_proc = cv2.erode(im_current,kernel,iterations = 1)
        if i=='6':#dilation
            kernel = np.ones((2,2),np.uint8)
            img_proc = cv2.dilate(im_current,kernel,iterations = 1)
        if i=='7':#histogram eq
            img_proc = cv2.equalizeHist(im_current)
        im_current=img_proc
      #Get OCR accuracy
      accuracy = getTesseractAccuracy(im_current,groundtruth_string,lang_code)
   except (RuntimeError, TypeError, NameError):
      print("err: "+str(Actions))

   return accuracy




def main():

   #Read base image
   img_base = cv2.imread('test.png',0)
   hist_base=cv2.calcHist([img_base],[0],None,[256],[0,256])
   groundtruth='Image Processor'
```

```
reward=getTesseractAccuracy(img_base,groundtruth,'eng')/len(groundtruth)

#Create q-table, initialize Correlation and 0 (no action) columns with their values
q_table= pd.DataFrame([[1,.0]],columns=['Correlation','0'])
#Copy table for loop operations
table=q_table.copy()

#Read image data from file
with open('train.txt', 'r',encoding='utf-8') as f:
    train_data = f.readlines()
#read image name, ground truth string in image and language code of ground truth string
train_names=[r.replace("\r\n","").split('\t')[0] for r in train_data]
train_contents=[r.replace("\r\n","").split('\t')[1] for r in train_data]
train_langs=[r.replace("\r\n","").replace("\n","").split('\t')[2] for r in train_data]

#loop through all images in training list
for i,n in enumerate(train_names):
    #Read current image in training list as grayscale
    im = cv2.imread(n,0)
    #Calculate histogram
    hist=cv2.calcHist([im],[0],None,[256],[0,256])

    #Compare each image with the base image to calculate current state (correlation)
    state=round(getSimilarity(hist_base, hist),2)
    #
    for it in range(iterations):
        Actions=getActionList(state,q_table)

        #apply actions on current image and calculate reward
        Feedback=getFeedback(im,Actions,train_contents[i],train_langs[i])

        #Add new row for new correlation value if not exists
```

```
        if len(q_table[q_table.Correlation == state])==0:
            table = table.append(pd.Series([state], index=['Correlation']), ignore_index=True)
        #Add new column for new actionset if not exists
        if ~pd.Series([str(Actions)]).isin(q_table.columns).all():
            try:
                table.insert(loc=len(table.columns),
                             column=str(Actions),
                             value=[0 if x != len(table.index)-1 else 0.0 for x in
range(len(table.index)) ])
            except (RuntimeError, TypeError, NameError):
                print("err: "+str(Actions))

        #Replace all NA values with 0
        table=table.fillna(0)
        #Calculate previous value of selected state-actionset pair
        prev_val=table[str(Actions)][(table["Correlation"]== state)]
        #Update q-table with new value
        table[str(Actions)][(table["Correlation"]== state) ] = prev_val+((Feedback+DF-
prev_val)*LR)
        #Update original table
        q_table=table

    return q_table

#Start program
q=main()
q.to_csv("q_100.txt", sep='\t', encoding='utf-8')
```

APPENDIX C

Python Code for Testing Q-Table Model

```
#import required libraries
import random
import numpy as np
import pandas as pd
import cv2
import pytesseract
import glob
import PIL
import time
import csv
import statistics as st

#Set initial values
iterations=1

#Greedy Threshold
GRDT=0.5

#Discount Factor
DF=0.9

#Learning Rate
LR=0.1

def test_main():

    #Define lists for storing Accuracies
    acc=[]
    aclist=[]
```

```
#Read base image, calculate OCR accuracy
img_base = cv2.imread('test.png',0)
hist_base=cv2.calcHist([img_base],[0],None,[256],[0,256])
groundtruth='Image Processor'
reward=getTesseractAccuracy(img_base,groundtruth,'eng')/len(groundtruth)

#Read selected q-table file
q_table = pd.read_csv('q_100.txt',delimiter='\t',encoding='utf-8',index_col=0)

#Take a copy of table for comparison
table=q_table.copy()

#Read test image set data from file
with open('test.txt', 'r',encoding='utf-8') as f:
    test_data = f.readlines()
 #Read test file names, content strings and content language codes for Tesseract
test_names=[r.replace("\r\n","").split('\t')[0] for r in test_data]
test_contents=[r.replace("\r\n","").split('\t')[1] for r in test_data]
test_langs=[r.replace("\r\n","").replace("\n","").split('\t')[2] for r in test_data]

#loop through test set
for i,n in enumerate(test_names):
    #Iteration each image for averaging accuracy
    for x in range(100):
        im = cv2.imread(n,0)
        hist=cv2.calcHist([im],[0],None,[256],[0,256])

        #Compare each image with the base image to calculate current state (correlation)
        state=round(getSimilarity(hist_base, hist),2)

        Actions=getActionList(state,q_table)
```

```
        #apply actions on current image and calculate reward (and next state (new histogram))
        Feedback=getFeedback(im,Actions,test_contents[i],test_langs[i])
        #Add row for new state if not exists
        if len(q_table[q_table.Correlation == state])==0:
            table = table.append(pd.Series([state], index=['Correlation']), ignore_index=True)
        #Add column for new actionset if not exists
        if ~pd.Series([str(Actions)]).isin(q_table.columns).all():
            try:
                table.insert(loc=len(table.columns),
                             column=str(Actions),
                             value=[0 if x != len(table.index)-1 else 0.0 for x in range(len(table.index)) ])
            except (RuntimeError, TypeError, NameError):
                print("err: "+str(Actions))

        #Replace all NA values with 0
        table=table.fillna(0)
        #Read previous value
        prev_val=table[str(Actions)][(table["Correlation"]== state)]

        #Update q-table
        table[str(Actions)][(table["Correlation"]== state) ] = prev_val+((Feedback+DF-prev_val)*LR)

        #Update original table
        q_table=table

        #Add current OCR accuracy to list
        acc.append(Feedback)
    #Average accuracy value for current image after all iterations
    aclist.append(sum(acc) / float(len(acc)))
```

```
            #Clear list
            acc=[]
        #Return list of average accuracy values
        print(aclist)
        #return q-table
        return q_table

#Start test program
q=test_main()
```