DOI: 10.15625/vap.2021.0060

TOWARDS NÔM HISTORICAL DOCUMENT OPTICAL CHARACTER RECOGNITION

Tran Thi Anh Thu, Le Pham Ngoc Yen, Tran Thai Son, Dinh Dien

University of Science, VNU-HCM

ttathu@apcs.vn, lpnyen@apcs.vn, ttson@fit.hcmus.edu.vn, ddien@fit.hcmus.edu.vn

ABSTRACT: *Nôm script is the ancient ideographic writing system of the Vietnamese for hundreds of years in the past (from the 10th century to the 20th century), hence, it embodies invaluable heritage of Vietnamese historical cultures, politics, economics, medicine practices and many more. To contribute to the preservation of Nôm script, in this paper, we focus on the task of building Nôm Optical Character Recognition model - one of the important and prerequisite components that supports vast digitizing Nôm documents. Our OCR baseline achieves a recognition rate of 71.80% mAP on top-1 and 80.54% mAP on top-5 for our testing dataset Truyện Kiều (1902); 72.96% mAP on top-1 and 82.36% on top-5 for the testing dataset Truyện Lục Vân Tiên.*

Keywords: *Nôm OCR, Nôm text recognition, Nôm historical document text recognition.*

I. INTRODUCTION

Nôm script had been used for centuries throughout our history. During this time, many literary works were created, some of them remained impactful and still being discussed amongst students, teachers, writers, historians, linguists, etc. Some primary candidates are *Truyện Kiều* by the great poet Nguyễn Du, *Truyện Lục Vân Tiên* by Nguyễn Đình Chiểu, *Quốc Âm thi tập* by Nguyễn Trãi, *Cư Trần Lạc Đạo Phú* by Trần Nhân Tông. Since the invasion of France at the end of 19th century, Nôm script has slowly disappeared as Quốc Ngữ script is created based on Latin character system and is widely taught in our education system. Now, not so many people can read and understand Nôm script while there are still many documents have yet to be studied and properly recorded. From a historical and cultural perspective, Nôm script holds a rich and invaluable heritage of the Vietnamese people. Studying and researching Nôm materials is the key to have a better understand about our past, hence, these materials must be preserved at all costs. According to *Di sản Hán Nôm Việt Nam - thư mục đề yếu*, in 1993 [1], Institute of Sino-Nôm studies owns 5,038 books and around 30,000 documents. These books are of topics like Literature, History, Religion, Education, etc. and these documents includes woodblock prints, films, microfilms, stone stelae, bronze bells, stone gongs, wood plaques, etc. Since then, more documents have been collected and currently the institute owns approximately 20,000 books, 48,000 stone stelae, and 20,000 antique woodblock prints [2]. However, there can be more Nôm materials scattered across country that have yet to be reported. Unfortunately, these materials have been severely damaged due to natural disasters like floods and wildfire. Consequently, there have been many efforts dedicated to Nôm digitization such as standardizing fonts, registering Nôm code points in the Unicode encoding system, computer scanning Nôm historical documents, retyping Nôm text in digital forms, etc. but the process is still slow. In this work, we aim to build a Nôm historical document optical character recognition model that should help facilitate Nôm digitization, in which we consider two approaches - a finetuned Tesseract-OCR model and a CNN-based model. Our Nôm datasets include a training set of *Truyện Kiều* (1871), two testing sets of *Truyện Kiều* (1902) and *Truyện Lục Vân Tiên*.

The paper is organized as follows: section II briefly overviews the published work on Nôm recognition, section III describes our Nôm datasets and discusses general problems on collecting Nôm historical documents, section IV presents our proposed methods, section V documents our experiments and results, lastly section VI concludes the paper.

II. RELATED WORKS

There is an extensive study on Nôm recognition conducted at the Tokyo University of Agriculture and Technology over the past years from 2011 to 2017 ([3], [4] and [5]). These works target the domain of Nôm historical documents, in specific, Nôm woodblock-printed books and ancient books. Their essential test set consists of 47 True Nôm pages with 2,539 classes of characters. These materials were collected from ten Nôm literary works, provided by Nom Preservation Foundation's digital library. Regarding the first stage of Nôm recognition which is the Nôm character segmentation, they use the area Voronoi diagram to localize the analysed connected components, then apply recursive X-Y cut method to separate out character bounding boxes. The method achieved a prominent result of 86.97% recall and 84.60% precision. The method is further employed into their Nôm recognition system proposed in 2016. The system's classification stage uses a common approach to recognize Chinese and Japanese characters including a series of nonlinear normalization, feature extraction, coarse classification, and fine classification. While the

coarse classifier intermediately suggests clusters of similar characters for an input, the fine classifier helps determine a final classification among the candidate classes. The total number of classes used in the system reaches 32,695, of which 7,660 classes are more frequent than the others. The recognition rate of the system on the said Nôm test set averages to 66.92% for top-1 and 78.34% for top-10. However, this is computed to the exclusion of 865 unknown characters. In 2017, the research continues with new methods using Convolution Neural Network (CNN) to replace the previous classifiers. In the coarse classification step, they use VGG-11 architecture as the feature extractor and an output layer of size 304 (the number of Nôm radicals). The fine classification step uses a stack of three Inception blocks and an output layer of size 32.695. The reported result, although of different testing set other than the said True Nôm pages, shows a significant improvement in top-1 recognition rate (from 69.08% to 81.73%).

It is also notable that most historical documents are woodblock-printed, which are copies of carved handwriting on wooden plaques [6]. Therefore, it is essential to consult the methods of Sinoxenic handwriting text recognition. Zhong et al. [7] modify the architecture of GoogLeNet with less Inception modules and use three types of conventional feature maps (Gabor, gradient, and Histogram of Gradient (HoG)). They manage to achieve a leading result of only 3.26% error rate. Although the handwritings in their CASIA dataset [8] has some variations in character size, there are no extremely big or small characters like in Nôm documents. Those large characters are Nôm document's title and author name, publisher's name printed in the cover page while the small ones are usually handwritten notes. Furthermore, their background is white while Nôm documents usually have a non-uniform yellow-brown background. Therefore, we decide to only adopt their proposed HCCR-AlexNet as one of our baseline classifiers where the character images have already segmented.

III. DATASET

A. Dataset Overview

1. Collecting dataset

Not many Nôm datasets are available online for public usage, only two domains that allow public access for non-commercial use which are Nôm Foundation and Chữ Nôm Resource. After examining the two websites, we see that Nôm Foundation has a large collection of document images, but only some literary works have relatively full text like *Truyện Kiều*, *Truyện Lục Vân Tiên*, *Chinh Phụ Ngâm Khúc*, etc. while Chữ Nôm Resource only has text corpus contributed by multiple volunteers. Of five versions of *Truyện Kiều*, only version 1870, 1871, and 1902 that have full texts and images. Hence, we decide to use *Truyện Kiều* version 1871 [9] and 1902 [10], and *Truyện Lục Vân Tiên* [11] as our OCR datasets. The statistics of training and testing datasets for OCR are shown in table 1. From now on, we will refer data created from *Truyện Kiều* (1871), *Truyện Kiều* (1902), and *Truyện Lục Vân Tiên* as K1871, K1902, and LVT, respectively.

Table 1. Dataset statistics for OCR

Name	Type	Number of images	Number of sentences	Number of characters	Number of vocabularies
K1871	Training	136	3.194	22.808	2,841
K1902	Testing 1	163	3.665	23.757	2,917
LVT	Testing 2	105	2.060	14.450	2,222
Entire dataset		**404**	**8.919**	**61.015**	**4.294**

2. Building dataset

Nôm text data for each image does not come available as annotation file but only comes as text lines that resemble the literary work's poetic form where the text belongs. Therefore, we must build our own annotation files for the entire dataset. We used a labeling tool called LabelImg [12] to create bounding boxes which described by four numbers denoting left-top and right-bottom coordinates. For the next step, we crawled all literary works' text from the website, then parsed each character and merged it with previously achieved list of contours accordingly and finally hand-checked every single annotation line. Some problems we noted with the labels of the text dataset:

- Many characters in the dataset were unknown as they have not been registered to the Unicode system. Hence, we try to find a relatively correct morpheme or stroke combination for the character in the image, then assign a pseudo-code and a pseudo-character (if available) for each of them as shown in table 2. During the building process, we see that many of the unknown characters are only slightly different from the defined and correct character. Hence, assigning a pseudo-character for an unknown Nôm character will reduce the number of unknown tokens in a character-level language model and is open for future scalability.

Table 2. Top 5 morphologies appear most frequently in the dataset

Morpheme	□□□	□□□	□人氷	□□亟	□□青
Occurrence	616	431	135	117	100
Pseudo-unicode	100072	100505	100519	100032	100014
Pseudo-character	□	□	□	拯	情

- The number of vocabularies of frequency 1 is large, accounting for more than 1/4 of the dataset as shown in table 3.

Table 3. Number of vocabularies of frequency 1 and unique to a literary work

Dataset Name	Frequency of 1	Remainder (Unique)	Remainder (Not Unique)
K1871	367	152	2.471
K1902	432	194	2.482
LVT	413	285	1.805
Entire dataset	**1,212**	**631**	**3.082**

B. Dataset Augmentation

1. For two-stage models

Since we have limited source data, we decide to augment training images using the four followings methods: a) Adding noise to images using: Lee Filter, Possion, Salt and pepper, and Speckle; b) Applying affine transformation to images using: translation, rotation, scale up, scale down, and shear in x-, y- and xy-axis; c) Applying textures that resemble old, yellowed, and wrinkled paper since images in the dataset possess these kinds of properties; d) Choosing a subset of method from three methods above then applying all of them on one image.

- *For segmentation model:* For this model, we use augmented page-level K1871 combines with TKH-MTH dataset (version 2) [13] as training dataset.
- *For classification model:* For this model, we use augmented character-level HWDB1.1, K1871, K1902, LVT to create training and testing datasets. In particular, images in HWDB1.1 and K1871 are used as training sets, while K1902 and LVT are combined to create NomTest as a total testing set.

2. Text column image synthesizing

We decide to synthesize a text column dataset for Tesseract as Tesseract is very sensitive and will not work if text columns are not straight enough and if characters are too close to border. Tesseract requires input as follows: for images, they have to be a binary image and save in ".tif" format; for labels, they have to be a text file that records characters' value and save in ".gt.txt" format with name be the same as the name of the image it describes. Note that white space and punctuation must match between images and labels. To create a data point, we use previously mention augmented dataset which are HWDB1.1 and K1871. A character images will be randomly chosen and pasted in a straight line onto a canvas with white padding around four edges. The according text file records the labels. We create 150.000 synthesize images with each character label appears at an average rate of 285 times.

IV. METHODS

A. Tesseract OCR

The Tesseract OCR Project[1] started out as a dissertation of Raymond W. Smith at Hewlett-Packard (HP) Enterprise. This project is later adopted by Google LLC and has been developed as an open-source OCR system since 2006. There are three major versions of the Tesseract OCR which are 3, 4, and 5: version 3 uses conventional computer vision algorithms; version 4 is a superior version in which the text line recognizer is now using a learning neural network model; version 5 is in pre-release with extensive changes in code formats. We only use version 4 and 5 of Tesseract OCR for our investigation on different baselines. Based on the original architecture which is only designed for Latin scripts such as English, a proposal has been raised to extend its ability to handle multilingual OCR such as CJK scripts, Arabic script, and Devanagari script [14]. The core stages of Tesseract of said proposal is illustrated in figure 1.

[1] https://opensource.google/projects/tesseract/

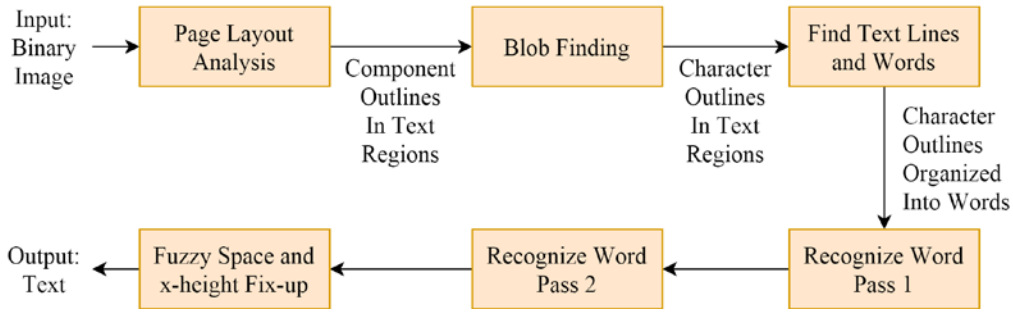


Figure 1. Tesseract top-level. [14]

The Page Layout Analysis algorithm is designed to be independent from characteristics of the input languages. The results of Connected Components Analysis (CCA) suggest page orientation (vertical or horizontal text lines) as well as determine boundaries between columns/lines of text. If the majority of the CCA results return vertical orientation, the whole page image is rotated +90º to achieve horizontal form. In later processing stage, these rotated characters are inversely rotated back to normal.

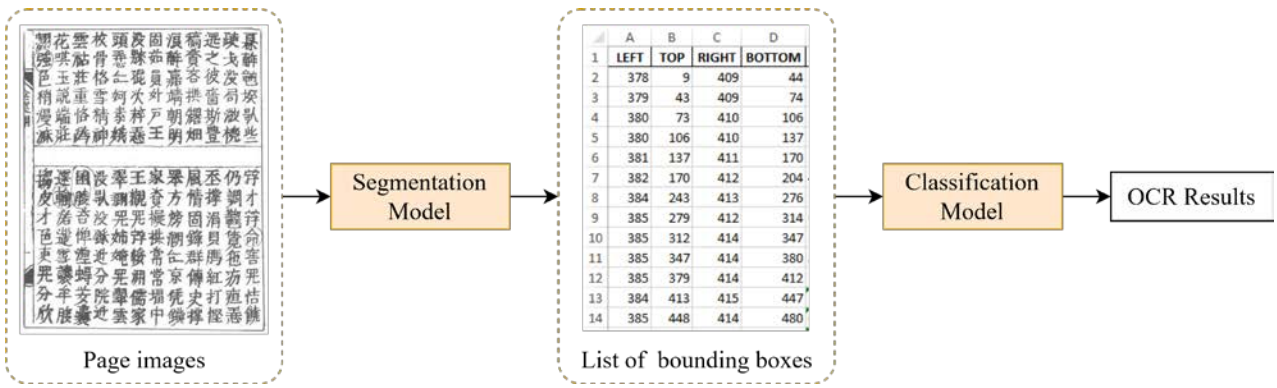
To separate blobs of potential characters, one useful sign in Latin scripts is the occurrence of white spaces on a text line. However, this is not the case for vertical CJKV scripts such as Nôm script. One of the solutions to this issue is to iteratively consult the results of the character classification. Therefore, the whole classification stage includes two main operations: Segmentation-Search and Shape Classification. In this proposed method regarding CJK scripts, they choose sentence punctuation to be character blob separators. To help limit the segments' length, they use a fixed-length constant of character width. In the shape classification operation, they adopt a mechanism called local voting of weak classifiers rather than using a global classifier with huge number of output classes.

Training task in Tesseract version 4 and 5 becomes easier: parameters as well as neural network structure can be declared directly in configuration file instead of recoding the entire model like in previous version. We decide to fine-tune Tesseract OCR for CJK scripts to train on Nôm data because:

- Nôm script is similar to Chinese since it was borrowed and modified based on it.
- Historical documents of both scripts were written Top-To-Bottom (TTB) Right-To-Left (RTL), which is also a mode supported by Tesseract OCR for CJK.
- According to Tesseract OCR documentation, to train a model from scratch requires a huge dataset and is extremely time-consuming. The result model is prone to have low recognition rate as it depends heavily on the dataset quality.

B. Convolutional Neural Network as an Approach to OCR

The OCR problem, at its core, is a special instance of the object detection problem where the target object class is a token (character, word, etc.) and the bounding box implies the token-order of a sequence. Therefore, object detection models can also be adapted for OCR purpose. Beside learning to predict label and bounding box of characters, the final returned sequences must also exhibit linguistic correctness (such as semantics and word-orders) for which sequence modeling can be applied. In our case, the order of writings follows historical Nôm documents' writings are RTL and TTB of a page.



	A	B	C	D
1	LEFT	TOP	RIGHT	BOTTOM
2	378	9	409	44
3	379	43	409	74
4	380	73	410	106
5	380	106	410	137
6	381	137	411	170
7	382	170	412	204
8	384	243	413	276
9	385	279	412	314
10	385	312	414	347
11	385	347	414	380
12	385	379	414	412
13	384	413	415	447
14	385	448	414	480

Figure 2. Two-stage OCR pipeline: segmentation and classification

In this Nôm OCR stage regarding using neural network baselines, we try with two basic types of detection pipelines: 1) End-to-end model using YOLO (version 5, referred to as YOLOv5); 2) Two-stage model using YOLOv5

for character image segmentation and a CNN-based classifier for character classification, where the classifiers in consideration are AlexNet (a simple architecture) and ResNet-101 (a more complex architecture), as shown in Figure 2. For the two-stage model, the data needed for segmentation only requires a single label for all Nôm characters.

YOLOv5: Although there is no published paper for YOLOv5 [15] during the time we conduct this research, certain key differences between it and previous YOLO versions are known. YOLOv5 also uses anchor boxes like previous version but instead of manually copy-paste hand-generated anchor boxes into configuration file, YOLOv5 has an auto anchor-box checking step that compares the anchor boxes against input data. If mismatch is detected, it will automatically train new anchor boxes and place them back into the model[2]. Furthermore, YOLOv5 is an easier to use model as it is implemented on a bigger and well-developed platform PyTorch instead of DarkNet like previous versions. The architecture of YOLOv5 implemented in PyTorch is shown in figure 3.

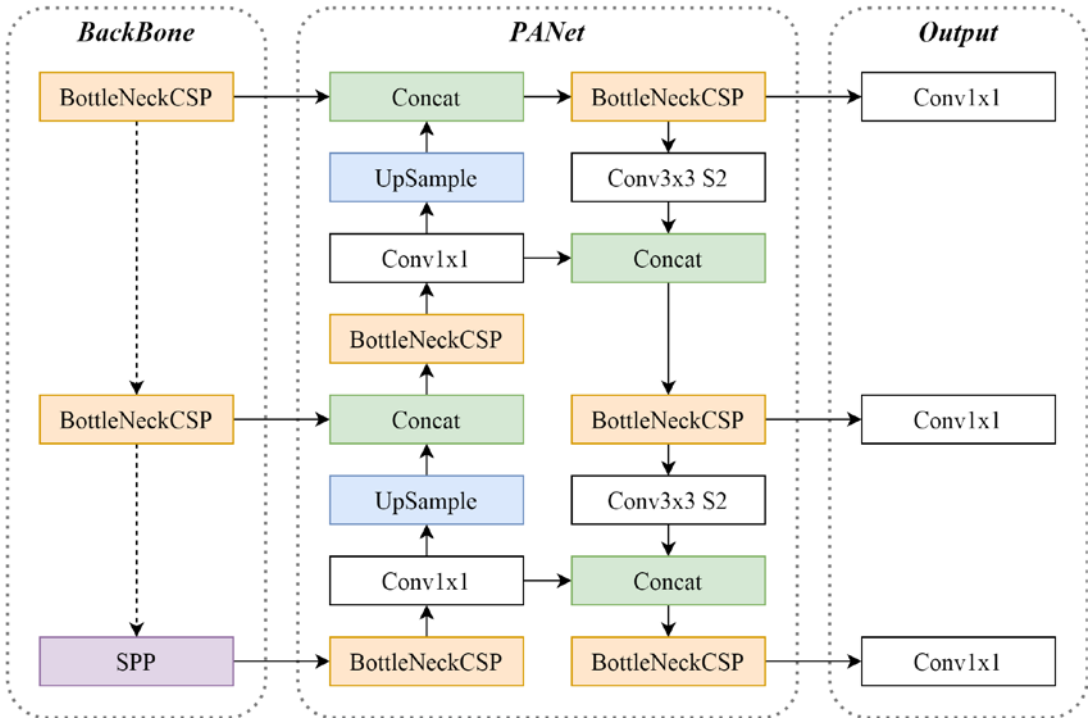


Figure 3. Overall architecture of YOLOv5[3]

AlexNet: AlexNet is the simplest architecture of CNN-based image classification model proposed by Krizhevsky et al. [16] in 2012. We modify the size of some hidden layers in refer to HCCR-AlexNet model proposed in [7], as illustrated in figure 4. We apply an additional Batch Normalization layer after each convolution step and each hidden dense layer.

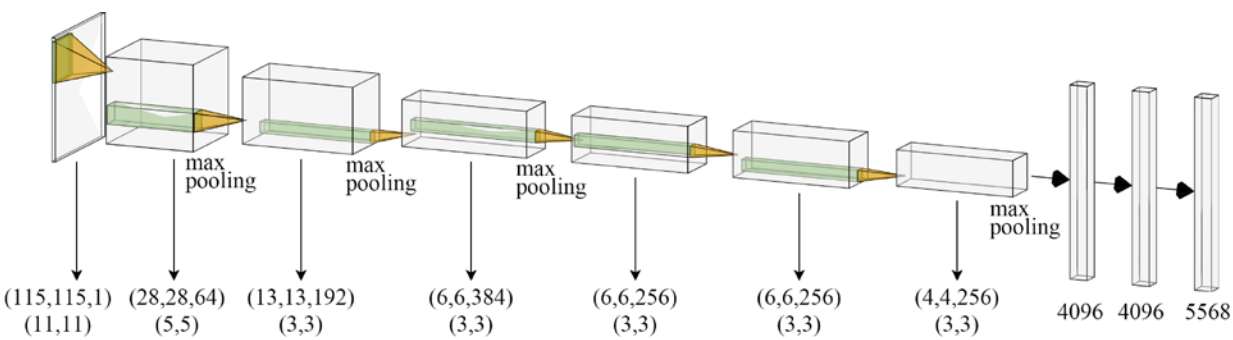


Figure 4. Overall architecture of AlexNet

[2] The author states this property here: https://www.youtube.com/watch?v=O4jOqVqyAo8

[3] This figure is not drawn in a published paper by the original author of YOLOv5, Glenn Jocher. It is drawn by a user in this github issue https://github.com/ultralytics/yolov5/issues/280 and is confirmed correct by the author.

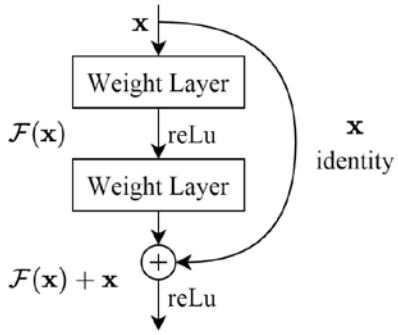


Figure 5.Residual learning block [19]

ResNet: When using CNN-based model to solve problem, one might ask if adding more layers, to make the model deeper and increase the receptive field, would result in a better model. This issue has been studied [17], [18], [19] and the answer is that this would instead make the model become worse and prone to higher training error as it becomes more difficult to propagate information through deeper layers and eventually the information will degrade and the gradient will vanish. To address this issue, He et al. [19] propose a Residual Neural Network (ResNet) that uses residual learning as building blocks instead of stacking convolutional layers like conventional CNN. As shown in figure 5, this unit block has a shortcut connection that bypasses intermediate layers and propagates information directly from shallow layer to deeper layer using identity function. This way, we provide an alternative path for the gradient and preserve some useful semantics information that capture in earlier layers. In this work, we only use ResNet-101 with conventional architecture [19] shown in figure 6.

To assess our models' performance, we use Mean Average Precision (mAP) metrics as commonly used by the ICDAR 2013 Chinese Handwriting Recognition Competition [20]. mAP computation involves calculating three values including true positive (TP), false positive (FP), and false negative (FN).

Precision and *Recall* are computed as follows: $p = \frac{TP}{TP + FP}$ and $r = \frac{TP}{TP+FN}$.

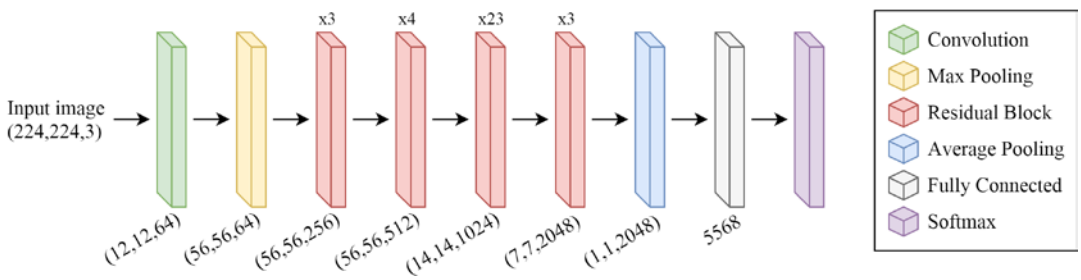


Figure 6. Overall architecture of ResNet-101

Then, the Average Precision (AP) is: $AP = \sum_{i=1}^{n-1}(r_{i+1} - r_i)p_{interpolated}(r_{i+1})$. The AP is a metrics producing a numerical value which is the area under the interpolated Precision-Recall curve. However, AP computation involves only one class while in OCR we are dealing with multiple classes. Therefore, mAP, which computes mean of AP across all K classes, would give better assessment for the model: $mAP = \frac{1}{K}\sum_{i=1}^{K} AP_i$.

We do not implement the metrics but use this application [21] that offers not just mAP metrics and absolute coordinate format, but many more metrics and formats required in different image contests.

C. Post-processing

1. Bounding box sorting

Given that the OCR results returned by an object detection model consists of a list of detected bounding boxes, and given that the writing order of our domain is RTL and TTB, we design an algorithm to sort the OCR results, as shown in Algorithm 1.

Algorithm 1 Nôm bounding box sorting algorithm

```
function SortBBoxRTLAndTTB(listOfBBoxes)
    firstBBox ← None
    maxCXCYDifference ← −∞
    for each bbox ∈ listOfBBoxes do
        bbox ← ConvertPixelToPercentageCoordinates(bbox)
```

```
        bbox.center ← FindBBoxCenter(bbox)
        bbox.CXCYDifference ← CXMinusCY(bbox.center)
        if bbox.CXCYDifference > maxCXCYDifference then
            firstBBox ← bbox
            maxCXCYDifference ← bbox.CXCYDifference
    columns ← Map⟨anchorBBox, memberBBoxes⟩
    columns.addNewColumn(firstBBox)
    listOfBBoxes ← PreSortDescendinglyByCX(listOfBBoxes)
    for each bbox ∈ listOfBBoxes do
        toAddNewColumn ← TRUE
        for each column ∈ columns and toAddNewColumn is TRUE do
            isInCurrentColumn ← checkCoverage(column.anchorBBox, bbox)
            if isInCurrentColumn is TRUE then
                column.memberBBoxes.addNewMember(bbox)
                toAddNewColumn ← FALSE
        if toAddNewColumn is TRUE then
            columns.addNewColumn(bbox)
    columns.sortColumnsDescendinglyByAnchorBBoxCX()
    columns.sortEachColumnMembersAscendinglyByCY()
    listOfSortedBBoxes ← columns.flatten()
    return listOfSortedBBoxes
```

The first step is to convert all bounding boxes' pixel coordinates to percentage coordinates by dividing each coordinate element by the image dimension. This step helps the sorting algorithm invariant to the input image's resolution. We then try to find the top-most and right-most bounding box which is the bounding box of the first Nôm character ($firstBBox$). Such bounding box has a difference between its x-coordinate center and y-coordinate center be a maximum. Next, we construct a data structure $Map\langle anchorBBox, memberBBoxes\rangle$ to keep track of a list of bounding boxes that belong to the same text column, identified by an anchor bounding box ($anchorBBox$). Next, we sort the list descendingly by x-coordinate centers in order for the loop to determine better anchors. In function "checkCoverage", we check whether the currently considered bounding box's x-coordinate center lies within any anchor's pair of x-coordinates: if it does then such bounding box belongs to the column identified by the anchor; otherwise, we construct a new column identified by it. The resulting $columns$ now contains the bounding boxes that are grouped into columns. To sort them into RTL order, we simply sort the data structure descendingly by the anchor boxes' x-coordinate center. And for each column's list of bounding boxes, we sort them ascendingly by y-coordinate centers to achieve TTB order. Finally, we convert the data structure to a regular list of bounding boxes for the end result by its internal function "flatten".

2. Nôm sentence splitting

Since these bounding boxes are already sorted from right to left and top to bottom like in Algorithm 1, the process of breaking the text sequence is very simple, as shown in Algorithm 2. As previously mentioned, a bounding box is represented by four values $left, top, right, bottom$. Using these values to calculate two more pairs of values, which is the center coordinate (x, y) and the bounding box size (w, h), using function "CalculateBBoxCenterAndSize". The function "CompareCenterAndSize" will then return true if any of the following conditions is met: a) distance between two x-coordinate centers is larger than a constant factor λ of average width of two bounding boxes; or b) distance between two y-coordinate centers is larger than a constant factor λ of average height of two bounding boxes. Else, a whitespace character is added instead to indicates that their values are on the same text line. Here, λ is a hyperparameter that we set to 1.5.

Algorithm 2 Page line splitting algorithm

```
function CalculateBBoxCenterAndSize(BBox)
    (left, top, right, bottom) ← (BBox_1, BBox_2, BBox_3, BBox_4)
    return ((right + left)/2, (bottom + top)/2, right − left, bottom − top)
function CompareCenterAndSize(x1, y1, x2, y2, w1, w2, h1, h2, λ)
    if (|x1 − x2| > λ · (w1 + w2)/2) or (|y1 − y2| > λ · (h1 + h2)/2) then
        return TRUE
```

```
    else
        return FALSE
function SplitLine (listOfBBoxes, λ)
    Initialize listOfSep as an empty list that will contain separator.
    B_page ← listOfBBoxes.length()
    for i = 1, 2, ..., B_page − 1 do
        (x1, y1, w1, h1) ← CalculateBBoxCenterAndSize(listOfBBoxes_i)
        (x2, y2, w2, h2) ← CalculateBBoxCenterAndSize(listOfBBoxes_{i+1})
        if CompareCenterAndSize(x1, y1, x2, y2, w1, w2, h1, h2, λ) is TRUE then
            listOfSep.append("\n")
        else
            listOfSep.append(" ")
    return listOfSep
```

V. EXPERIMENTS & RESULTS

A. Tesseract OCR

The network is configured for our Nôm data as follow: the LSTM hyper-parameter is [1,0,0,1 $Lbx256$ $O1c5568$]. The first four integers (batch, image height, image width, and image depth) indicate that our input is single batch, variable in size and is gray-scaled. The $Lbx256$ controls which network type used is Bidirectional-LSTM with 256 hidden outputs. The last value $O1c5568$ shows that the classification use Connectionist Temporal Classification loss, the output layer is of size 5,568 (the number of training labels). Our starting model is Tesseract's Chinese Traditional pretrained model on vertical script and language type chosen is RTL (Right-To-Left). Since we are dealing with vertical script, the selected page segmentation method is Single uniform block of vertically aligned text. The learning rate is left as default $1e⍰4$. Some examples training iterations and results are shown in Table 4 and figure 7. The current error rate of Tesseract OCR in our testing process is 62.46%, which is very high, and the model needs more time to converge.

B. Convolutional Neural Network

As mentioned previously, we experiment with three OCR baselines: End-to-end-YOLO, YOLO-AlexNet, and YOLO-ResNet101. All the three models are implemented in PyTorch - an open-source machine learning framework [22] and its research-accelerated side framework named PyTorch Lightning [23]. We train our models using Google Colaboratory [24] which is allocated 26GB of RAM, a Tesla V100-SXM2-16GB GPU and an Intel(R) Xeon(R) CPU @ 2.00GHz CPU. Details for each model as follows:

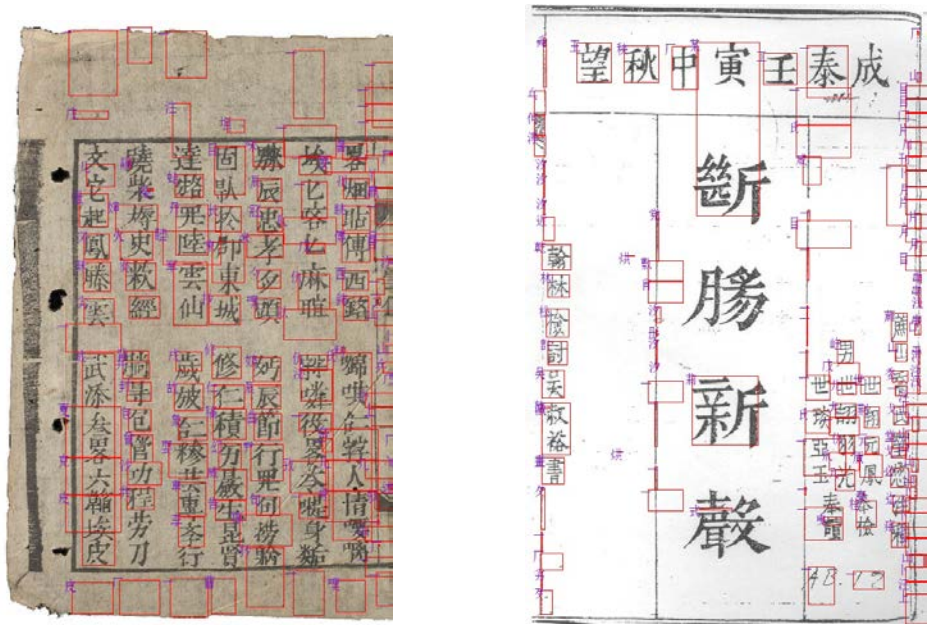
- End-to-end YOLO: We choose to use the version 5 of YOLO. This model is experimented with page-level dataset described in Section III.B.1 for both segmentation step and classification step.
- YOLO-AlexNet: the YOLO segmentation module uses the same datasets as the standalone end-to-end YOLO model where each Nôm character is label as one single class named nom_char. This model uses the page-level and character-level datasets described in Section III.B.1.
- YOLO-ResNet101: the model uses the same dataset as YOLO-AlexNet.

Table 4. Example predictions in some later iterations of Tesseract OCR

Iteration	Ground-truth	Best OCR text
251875	壙 戍□缔铀薄	壙 戍□缔钠藻
251894	摇辰悶 幍境漂拒午桩吧 沿梯□辣力再	描瓜悶 幍境漂姬吊吧 招揣□孰丿再
251902	籍造□□屯 甫	籍遗□鬚□店 甫
251909	枢□ □ 蚕锹绚按 □	柜□ □ 蚕琳岛护 □
251928	败联紫夾庞□差 □磕莊 抢咽呿增五	败院繁淡庞□芒 □磕莊 抡蛆呿增互

Table 5. Final top-1 and top-5 validation (mAP@0.5, %) on Nôm datasets of the three OCR baselines

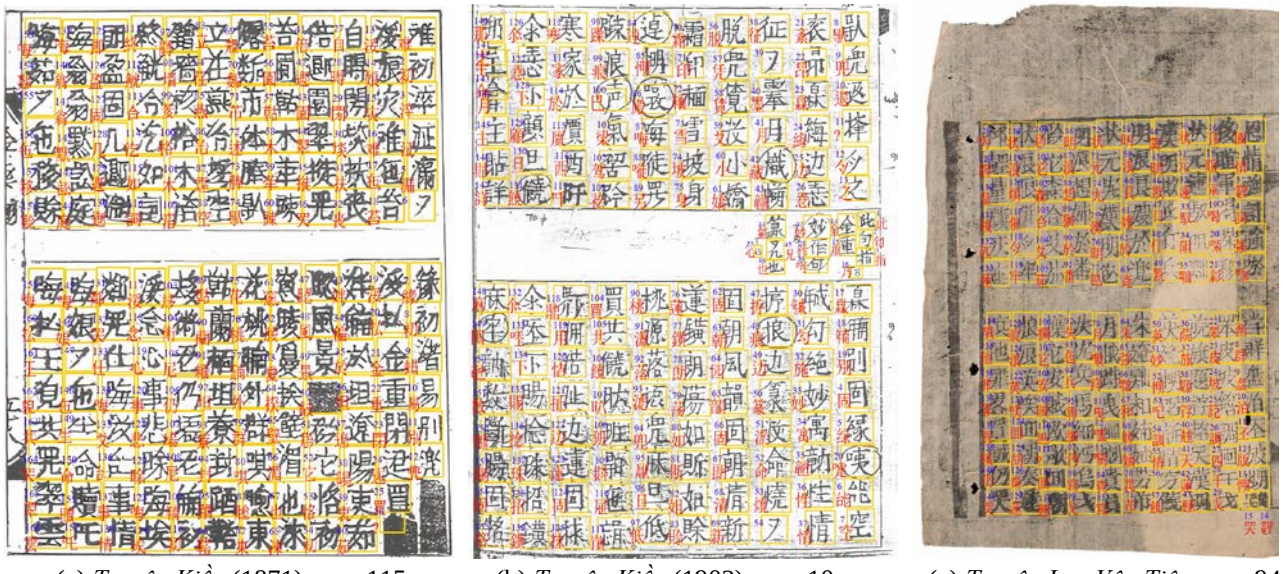
Model	NomTrain	NomTest	
	K1871	K1902	LVT
End-to-end YOLO	0.0618	0.062	
YOLO-AlexNet	98.47 - 99.14	59.02 - 68.76	59.78 - 70.60
YOLO-ResNet101	98.91 - 99.32	71.80 - 80.54	72.96 - 82.36



(a) Truyện Lục Vân Tiên page 2 (b) Truyện Kiều (1902) page 0

Figure 7. Some examples result outputs of Tesseract OCR

Since the target of baseline OCR models is to detect Nôm characters on page images, the final validation on Nôm datasets is listing in table 5. The end-to-end YOLO detector performs poorly since the number of labels is too large (5,568 labels) which is one of the limitations of YOLO itself [25]. When using as one-label segmentation module, YOLO performs exceptionally well. The classifier AlexNet uses Stochastic Gradient Descent (SGD) optimizer with a learning rate of $1e\text{-}3$ for the first 10 epochs and reduced to $1e\text{-}4$ for the latter epochs. The AlexNet fails to improve after 16 epochs. Meanwhile, although the ResNet101 classifier takes more epochs each with longer time to train, it converges better than AlexNet.



(a) *Truyện Kiều* (1871) page 115 (b) *Truyện Kiều* (1902) page 10 (c) *Truyện Lục Vân Tiên* page 94

Figure 8. Some example results of training and testing datasets (YOLO-ResNet101)

Some of the results of the model YOLO-ResNet101 are shown in Figure 8. The prediction shown in Figure 8a is an example for the final results on the training dataset, sorted RTL and TTB, in which there are some wrong captures at non-existed character areas such as bounding boxes numbered 26 and 50. The character class at the 6th bounding box is □, denotes a repetition of the character preceding it. The prediction is correct, however, the font Nom Na Tong we used to write on image is incapable of rendering it (along with some other unicodes that we notice). The prediction shown in Figure 8b and 8c are examples for the final results on the two testing datasets. The dataset *Truyện kiều* (1902) contains irregular comment texts in the middle area of the page which affects the sorting order of the OCR outcomes. For the testing dataset *Truyện Lục Vân Tiên*, we pick the page that is unevenly lit to illustrate. The result shows that the model manages to predict fairly correctly in such extreme conditions.

VI. CONCLUSION

In this paper, we give statistical information about Nôm page images and their labels that we use for our OCR model. Furthermore, we also address problems regarding the label along with the solutions we propose to partially resolve them. We also show each individual dataset that we generate or augment for different OCR models.

To construct our Nôm OCR model, two methods that we consider are open-source and commercially available OCR systems (Tesseract OCR) and established CNN-based models (YOLOv5, AlexNet, and ResNet). After the trial process, we see that there are many problems during the installation process of Tesseract OCR that cost us more than two weeks. It also has a very strict criteria on input data formatting, and demand a lot of resources to train locally. With respect to CNN-based models, the YOLOv5 end-to-end model performs badly as it cannot handle that many categories while the two-stage models produce good results and cost less time. In specifically, YOLO-ResNet101 has the better performance than YOLO-Alexnet, therefore, we choose to it for further experiments with language model integration.

As we adapt object detection model for Nôm OCR model, it is necessary that the final output also exhibits linguistic correctness for which sequence modeling can be applied. Therefore, we also give a sorting algorithm and splitting to handle this task.

VII. ACKNOWLEDGMENTS

This research is supported by research funding from Ho Chi Minh City Department of Science and Technology.

REFERENCES

[1] N. Tran, *Di sản Hán Nôm Việt Nam - thư mục đề yếu*. Ha Noi: Social Sciences, vol. I, pp. 15-47, 1993. Available: http://www.hannom.org.vn/default.asp?CatID=462.

[2] Institute of Sino-Nôm studies. Available: http://www.hannom.org.vn/default.asp?CatID=8.

[3] T. Van Phan, B. Zhu, and M. Nakagawa, "Development of nom character segmentation for collecting patterns from historical document pages", in *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*. New York, NY, USA: Association for Computing Machinery, pp. 133-139, 2011. Available: https://doi.org/10.1145/2037342.2037365.

[4] T. Van Phan, K. C. Nguyen, and M. Nakagawa, "A nom historical document recognition system for digital archiving", in *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 19, No. 1. Springer, pp. 49-64, 2016. Available: https://doi.org/10.1007/s10032-015-0257-8.

[5] C. K. Nguyen, C. T. Nguyen, and N. Masaki, "Tens of thousands of nom character recognition by deep convolution neural networks", in *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*. New York, NY, USA: Association for Computing Machinery, pp. 37-41, 2017. Available: https://doi.org/10.1145/3151509.3151517.

[6] D. H. Chau, "Woodblock printing, an essential medium of culture inheritance in chinese history", *Journal of the Hong Kong Branch of the Royal Asiatic Society*, vol. 18.

[7] Z. Zhong, L. Jin, and Z. Xie, "High performance offline handwritten Chinese character recognition using googlenet and directional feature maps", in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 846–850, 2015. Available: https://doi.org/10.1109/ICDAR.2015.7333881 .

[8] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang, "Casia online and offline chinese handwriting databases", in *2011 International Conference on Document Analysis and Recognition*, pp. 37-41, 2011. Available: http://www.nlpr.ia.ac.cn/databases/handwriting/Download.html.

[9] Nôm Foundation. Available: http://nomfoundation.org/nom-project/tale-of-kieu/tale-of-kieu-version-1871?uiLang=en.

[10] Nôm Foundation. Available: http://nomfoundation.org/nom-project/tale-of-kieu/tale-of-kieu-version-1902?uiLang=en.

[11] Nôm Foundation. Available: http://nomfoundation.org/nom-project/Luc-Van-Tien/Luc-Van-Tien-Text?uiLang=en.

[12] T. Lin, "labelimg", GitHub repository, 2015. Available: https://github.com/tzutalin/labelImg.git.

[13] L. J. S. W. J. W. Y. W. Weihong Ma, Hesuo Zhang, "Joint layout analysis, character detection and recognition for historical document digitization", *ICFHR 2020*, 2020.

[14] R. Smith, D. Antonova, and D.-S. Lee, "Adapting the tesseract open source ocr engine for multilingual ocr", in *Proceedings of the International Workshop on Multilingual OCR*. New York, NY, USA: Association for Computing Machinery, 2009. Available: https://doi.org/10.1145/1577802.1577804.

[15] G. Jocher, A. Stoken, J. Borovec, NanoCode012, A. Chaurasia, TaoXie, L. Changyu, A. V, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, J. Hajek, L. Diaconu, Marc, Y. Kwon, oleg, wanghaoyang0106, Y. Defretin, A. Lohia, ml5ah, B. Milanko, B. Fineran, D. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations", apr 2021. Available: https://doi.org/10.5281/zenodo.4679653.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Commun. ACM*, vol. 60, no. 6, p. 84-90, may 2017. Available: https://doi.org/10.1145/3065386.

[17] K. He and J. Sun, "Convolutional neural networks at constrained time cost", *CoRR*, 2014. Available: http://arxiv.org/abs/1412.1710.

[18] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks", *CoRR*, 2015. Available: http://arxiv.org/abs/1505.00387.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *CoRR*, 2015. Available: http://arxiv.org/abs/1512.03385.

[20] F. Yin, Q.-F. Wang, X.-Y. Zhang, and C.-L. Liu, "Icdar 2013 chinese handwriting recognition competition", in *2013 12th International Conference on Document Analysis and Recognition*, pp. 1464-1470, 2013,. Available: https://doi.org/10.1109/ICDAR.2013.218.

[21] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit", *Electronics*, vol. 10, No. 3, 2021. Available: https://www.mdpi.com/2079-9292/10/3/279.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., pp. 8024–8035, 2019. Available: http://papers.neurips.cc/paper/9015-pytorch- an-imperative-style-high-performance-deep-learning-library.pdf.
[23] e. a. Falcon, WA, "Pytorch lightning", *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.
[24] E. Bisong, Google Colaboratory. Berkeley, CA: Apress, pp. 59–64, 2019. Available: https://doi.org/10.1007/978-1-4842-4470-8_7.
[25] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", CoRR, 2015. Available: http://arxiv.org/abs/1506.02640.

TIẾP CẬN NHẬN DẠNG KÝ TỰ QUANG HỌC TRÊN ẢNH CHỤP TÀI LIỆU CỔ CHỮ NÔM

Trần Thị Anh Thư, Lê Phạm Ngọc Yến, Trần Thái Sơn, Đinh Điền

TÓM TẮT**:** *Chữ Nôm là hệ thống chữ viết cổ của Việt Nam từ khoảng thế kỷ thứ X đến thế kỷ thứ XX. Chính vì được sử dụng trong khoảng thời gian rất dài của lịch sử, chữ Nôm chứa đựng kho tàng quý báu của người Việt xưa về văn hóa, chính trị, kinh tế, y học,... và nhiều hơn thế nữa. Nhằm góp phần vào công việc bảo tồn chữ Nôm, trong bài nghiên cứu này, nhóm tác giả bước đầu đề xuất giải pháp nhận dạng ký tự quang học chữ Nôm cho loại tài liệu bản khắc gỗ Nôm cổ. Tỉ lệ nhận dạng giải pháp đạt được, theo hệ mAP, như sau: 71.80% (top-1) và 80.54% (top-5) trên tập kiểm tra Truyện Kiều (1902); 72.96% (top-1) và 82.36% (top-5) trên tập kiểm tra Truyện Lục Vân Tiên.*