VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Pham Truong Giang

Final Thesis

LEARNING EMBEDDINGS FOR RECOGNIZING HAN-NOM CHARACTERS IN VIETNAMESE HISTORICAL BOOK

HA NOI - 2022

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Pham Truong Giang

Final Thesis

Learning Embeddings for Recognizing Han-Nom Characters in Vietnamese Historical Books

Supervisor: Dr. Ta Viet Cuong, MsC. Kieu Hai Dang

HA NOI - 2022

ABSTRACT

The historical Han-Nom books which were written during pre-modern era of Vietnamese contains are important sources of cultural and historical values. One of the main challenge of analysis task of Han-Nom books is to detect and recognize the character. There are several open sources which can be used for working with Han-Nom books. However, these tools are usually designed to work with Chinese character rather than Han-Nom character which makes them not suitable for analyzing the historical Vietnamese Han-Nom books. Moreover, because of the evolving and mixing of language, the historical Han-Nom characters change overtime and it is difficult to design a universal detector which can work with a wide range of books.

In order to resolve the above issues, our works employ a few shot learning approach which could be trained based on a small amount available data. We propose a way to recognize the detected characters given only a few samples. For dealing with the small amount of available data, we employ an representation learning approach. Instead of training a classifier, our network learn an efficiency embedding which can be used to retrieve the candidate characters from the list of labeled characters.

We evaluate our method on a wide range of historical Han-Nom books which include the printing-style, Han character, handwriting-style character and Nom character. For classifying characters, our learning represenation approach out-performs the traditional classifier training on the domain of few-shot learning. Our proposed method could improve the top 1 and top 5 accuracy substantially in comparison to train a standard classifier. Moreover, we build a small web-based demo which can be used for detecting and classifying Han-Nom characters in books automatically.

Contents

List of figures

List of tables

CHAPTER 1. INTRODUCTION

1.1 Motivation

According to wikipedia, Sino-Vietnamese characters (Vietnamese: Hán Nôm) are Chinese-style characters read as either Vietnamese or as Sino-Vietnamese. When they are used to write Vietnamese, they are called Nôm. The same characters may used to write Chinese.

Han-Nom comprises of two writing systems: Hán scripts and Nôm scripts, where Hán script is the Chinese characters, Nôm script is the (basically) Chinese characters used to represent Sino-Vietnamese vocabulary and some native Vietnamese words. The meaning of the name "Nôm" is that this is the language used to record the voice of the Southern people (Vietnamese people), as opposed to the Chinese characters used by Northern people, ie Chinese people. As a result, Nôm script inherits many existing words from Chinese scripts and other words represented by new characters created using a variety of methods.

Han Nom characters are a cultural heritage of Vietnam, it plays a particularly important role in literature throughout the history. Institute for the Study of Han Nom Vietnam is currently storing hundreds of valuable Han-Nom documents in the study of the ancient Vietnamese in many fields: literature, ideas, philosophy, art, language, law, history, morality, etc. However, there are only a few people who can read and write Han-Nom nowadays, making the research of Sino difficult. Therefore, applying technology to automatically recognize and analyze Han-Nom characters is of great importance and has wide practical impacts on the research of ancient Vietnam.

1.2 Approach

There are a number of challenges in building a system that can detect Han Nom characters. The fact that the Institute for the Study of Han Nom not only stores but also publishes Han Nom documents on the website bring huge potentials for research. However, most of the documents are currently unlabeled, and most of supervised machine learning model requires large amount of labeled data in order to operate efficiently. Moreover, Han Nom characters has been continuously updated throughout the history so when working with a new document, there is a good probability that we encounter one or more new

characters. For all of the above reasons, my approach focuses on building a system that is able to observe a small amount of Han Nom data (in our case the data is one or a few images of a Han Nom character) but can already generalize on a bigger scale. In our method, I take characters that can be labelled by EasyOCR (a tool for detecting and recognizing text) as raw labels and split them into support and query set, the size of the query set is bigger than the support set, and then train the model to learn the embeddings or the main features of the characters so when a new character is updated into the database, the model can learn to recognize it faster. Then, I deploy the models on the web, which takes pages of Han Nom, localizes characters and outputs top 1 and top 5 closest labels for a character.

For the text recognition part, we designed our solution to be flexible and can adapt to new unforeseen characters easily. We approached the problem in few shot learning directions: we limited ourselves to use only a small amount of training data and trained them to be able to generalize to a bigger scale. In order to aid the training process, we used additional artificial data generated automatically from fonts and used them to pretrain the model. The model can be then fine-tuned on the book dataset. In the inference step, we use the trained model to embed images in the query set to extract compact representations of the characters and search the support set for top possible labels by using k nearest neighbor on the embedded space.

As stated, most of the data we have at hand are unlabeled. To obtain labels for any meaningful learning processes, we employed an off-the-shelf OCR tool for Chinese characters recognition. The resulting labels acquired by this way are messy and can be detrimental to the training process if used directly: they include gibberish texts, undetected characters and false predictions. We alleviated this issue by applying some data cleaning techniques to labels.

1.3 Data

Our data includes seven Han Nom books captured in jpeg form from the Institute for the Study of Han Nom Vietnam. The name of the seven books are: Phap Hoa De Cuong, Truy Mon Canh Huan, Khoa Hu Luc, Dai Nam Quoc Su Dien Ca, Dai Nam Thuc Luc Tien Bien, Dai Viet Su Ky Toan Thu (typed version and hand written version). Some of the different images for different pages in different books are shown in figure 1.1. Descriptions of the seven books are presented below:

- Phap Hoa De Cuong is a Buddhism book written in the twentieth century by a Vietnamese monk. The images from Phap Hoa De Cuong are captured from a printed book.
- Truy Mon Canh Huan is a Buddhism book, written in Tong Dynasty in China. It contains discipline for the monks to follow. It encourages people to follow a religious life. The images from Truy Mon Canh Huan are captured in a printed Chinese book.

- Khoa Hu Luc is the work of King Tran Thai Tong in the thirteen century. The book can be describe as King Tran Thai Tong's comtemplation on the nothingness. The images are captured from a Chinese printed book.
- Dai Nam Quoc Su Dien Ca is a creation followed the order of King Tu Duc in the era of the Nguyen Dynasty. It is a history book written in Luc Bat, presents the history of Vietnam from King Duong Vuong till the end of the Le Dynasty. The images from Dai Nam Quoc Su Dien Ca are captured from a printed book written in Nom.
- Dai Nam Thuc Luc Tien Bien is published in 1844, and is known as the official records of the Nguyen Dynasty. Dai Nam Thuc Luc Tien Bien is written in classical Chinese. The images from Dai Nam Thuc Luc Tien Bien in the data are captured from a printed book.
- Dai Viet Su Ky Toan Thu is another historical records written in classical Chinese. It is written by historian Ngo Si Lien and finished in 1479, in the period of the Le Dynasty. Our data consists images of Dai Viet Su Ky Toan Thu in printed version and hand-written version.

Figure 1.1: Han Nom books samples

1.4 Outlines

Our report is structured as followed:

- Chapter 1: Introduction. This chapter presents motivation for applying technology in recognize characters in Han Nom documents and shows the importance of applying few-shot in document analysis.
- Chapter 2: Related works. This chapter briefly describes the related work.

- Chapter 3: Methods. This chapter describes in detail our method for text recognition in few-shot settings.
- Chapter 4: Experiments and Results. This chapter shows training details, illustrates the result and discusses it.
- Chapter 5: Models Deployment. This chapter presents technology and results on deploying the models which is built upon the results of our experiments.
- Chapter 6: Conclusion and future works.

CHAPTER 2. RELATED WORKS

As mentioned in the Introduction chapter, our work involves Han Nom characters classification, in which we acquire the use of some few-shot technique to classify the characters' images that has been localized. Chapter 2 presents background knowledge on Deep Network, Few-shot learning.

2.1 Deep Network

This section will introduce to the concepts of Deep Neural Network, different components of a Deep Neural Network, a variation of Neural Network called Convolutional Neural Network (CNN), and explain the structure of the deep models I have used in the project.

2.1.1 Deep Neural Network

Deep Neural Network [7] is often seen in supervise learning scenarios. In supervise learning, we have datasets with (x, y) input, output pairs. The goal of machine learning is to find a function f, so that for each input x, $f(x) = y$. Since finding the exact function f is often impossible in reality, all machine learning models try to find another function $f^* \sim f$ so for each input x, we have $y \sim f^*(x)$. In fact, each function f^* not only takes input x as the input to the function, deep learning models often also have their own set of parameters θ, when the correct set of θ is applied, we can find our optimal function f^*. And the goal of most supervise learning models as well as deep learning models is to figure out the best θ through different model architectures and techniques. Overall, we can formulate the problem of supervise learning for deep learning as given input, output pairs (x, y), find a function f^* and a set of parameters θ so that $y \sim f^*(x; \theta)$.

The reason why it is called Network is because in Neural Network, function f^* is a compilation of different functions $f^*(x) = f^1(f^2(f^3(...f^n(x)))$. Each function $f^k; k \in 1, 2, 3, ..., n$ is called a hidden layer. A function f^k is normally a linear transformation combined with an activation function. An activation function are used to increase the deep learning model non-linearity, since (x, y) pairs in every dataset do not always have linear relationship. General structure of a simple Neural Network is found in figure 2.1 while different activation functions are shown in figure 2.2

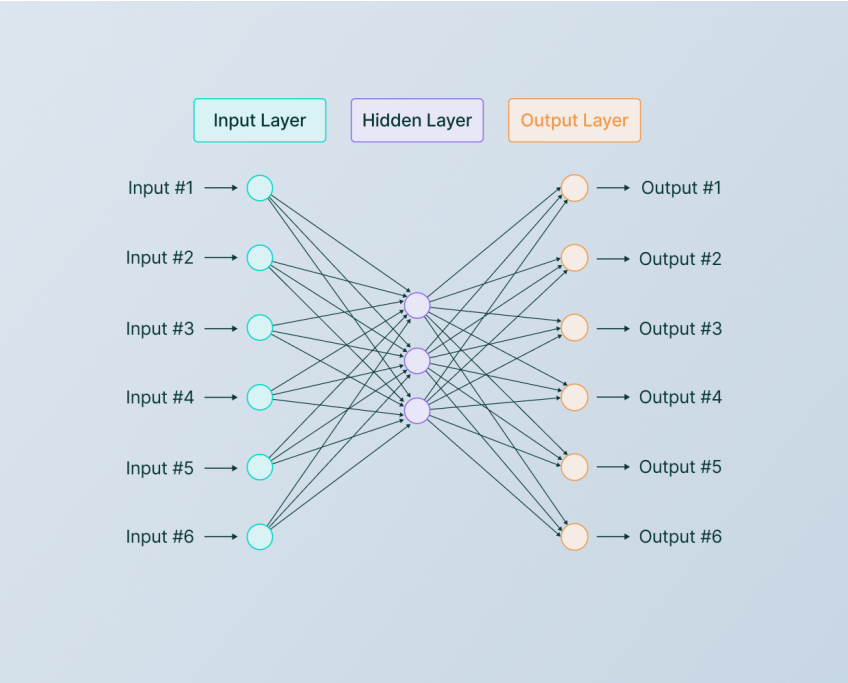


Figure 2.1: Simple Neural Network Architecture, taken from [1]

As explained above, the task of a typical deep neural network is to find the set of θ so that $y \sim f^*(x;\theta)$. In order to measure the similarity between the output and our prediction function, a Loss function is used. The loss function is defined as $L = Loss(f(x), y)$. The smaller the Loss, the better our learned function is. The method to step by step modify the set of θ so as to minimize the Loss function is called Gradient Descent. For example, assuming that we have of θ_0, which produce loss L, then Gradient Descent will update θ_0 to θ_1 according to formula 2.1. The formula will run iteratively in the hope that after some iterations, θ will converge and loss L will be at its local or global minimum. Of course, in order to apply the formula 2.1, the loss function should be differentiable. And the process of updating θ is called the training process.

$$\theta_1 = \theta_0 - \frac{\nabla L}{\theta_0} \tag{2.1}$$

In traditional machine learning, we often divide the dataset into training set and test set (sometimes we also have the validation set). The training set will be used for the process of updating the θ parameters. The test set is used to validate the function learned on the training set. Then when training and testing the model, a few patterns will emerge. In the first scenario, the model achieves high result on both training set and test set, then we might have trained a good model since it can generalize well even on unseen data in the test set. In the second case, our model gets low results on both training set and test set, which is called under-fitting, it happens when the model cannot adapt to both the training set and test set. The third scenario is the model gets high result on the training set but low result on the test set, we call the pattern to be over-fitting. In case there is too little labelled data in the training set, we can fit the training set well, however

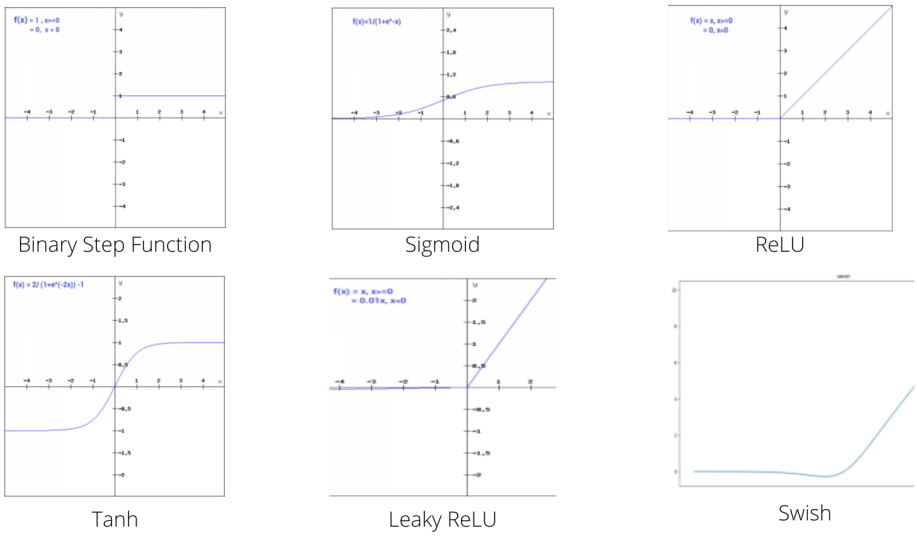


Figure 2.2: Different activation functions [20]

the training set is not enough to represent the distribution of the dataset, thus get low result on the test set. This over-fitting case mentioned above happens to be the problem that few-shot learning wants to tackle, the subject of which I will discuss deeper in section 2.2. Under-fitting, Over-fitting and Appropriate fitting are visualized in classification is visualized in figure 2.3.

However, there are some problems when applying simple neural network model as described in figure 2.1 to the image dataset since the input to simple neural network architecture is a flat vector. However, in image dataset, the area surrounding a pixel tells something about that pixel as well, so flattening the image will cause losses to the images' information. So for the image dataset, Convolutional Neural Network architectures are leveraged.

2.1.2 Convolutional Neural Network

Convolutional Neural Network [16] is very similar to traditional deep neural network architecture except they do not flatten the input immediately, but instead make use of the convolution operation to extract information not from individual pixel (in the case of image dataset) but from their neighborhood as well. The following passages will clarify the concept of convolution operation and pooling operation, two typical operations in Convolutional Neural Network. I will also present about the basic architecture of Residual Neural Network, the backbone I use in all of my models.

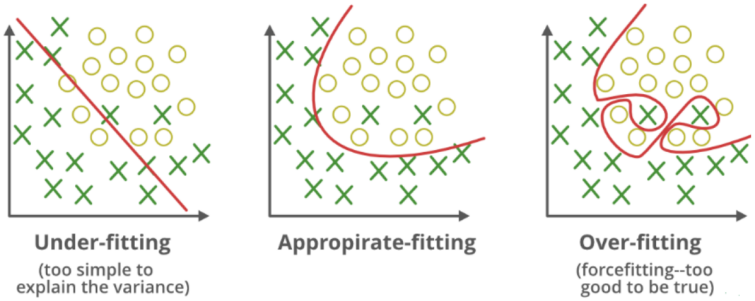


Figure 2.3: Different model fitting scenario [5]

Convolutional Operation

Convolutional Operation involves the use of a small kernel (normally of size $(2k+1)\times(2k+1)$). The kernel slides across the input images or feature maps to output another feature map as shown in figure 2.4. The output feature map then goes through activation function to increase the non-linearity of the model as explained in section 2.1.1.

Pooling Operation

In a typical Convolutional Neural Network Architecture, the input goes through several convolutional block with activation function to extract different features of the image. Then the pooling layer is applied. The output feature maps after pooling can be said to contain statistical properties of the previous layer. Pooling also involves a kernel moving through the image, but different from convolutional operation, the output feature map of pooling only depends on the value of the input of pooling layer. Some of Pooling techniques often used are Max Pooling or Average Pooling.

Residual Neural Network

Residual Neural Network architecture is first introduced in [8] after the author realizes the limitation of traditional convolutional neural network. One of them is that after training for some time for model with many layers, gradient explode/vanish will appear, harming the training process. So, the authors add some changes to the architecture. Normally, a layer in traditional convolutional neural network only uses the layer right before

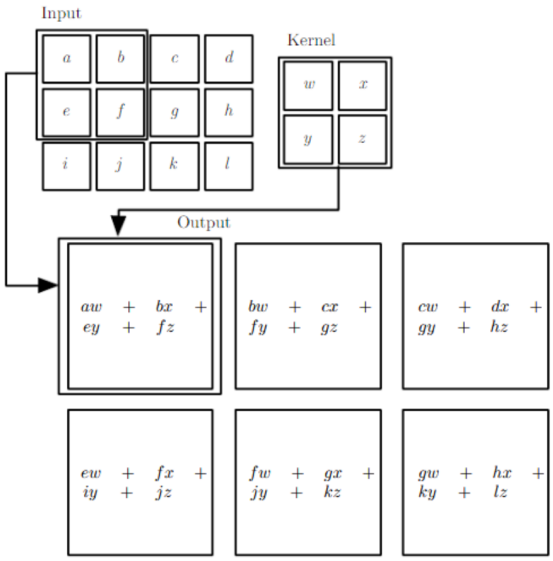


Figure 2.4: Different model fitting scenario [7]

it, thus if the layer before it appears to have problem, the latter layers will be affected as well. As a result, a slightly modified structure is introduced as described in figure 2.5. A feature map in layer n will be added to the feature map in layer $n + k$. The intuition is that if feature map in layer n does not have any problem but layer $n + k$ appears to have gradient explode or vanish. The next layer can still be fine since it also uses the layer n. From this basic building block, different architecture was born and used effectively in different settings. My experiment leverages Resnet 50 and Wide Residual Neural Network [27] as backbone to classify between different characters in the Han Nom dataset.

2.2 Few-shot learning

The fact that we want to generalize our model to characters in Han Nom books after training with a few examples makes our problem become a few-shot problem. We as humans can learn to recognize a character, whether normal text or Han Nom text only through a few samples. However, most modern deep learning models can only generalize well when given a good amount of data to learn. To bridge that gap between humans and machines, the idea of few-shot learning has emerged. Formally, Few-shot learning is defined as a type of machine learning problem. The Machine Learning model is defined as a model to improve accuracy P on classification task T, given the experience E. The only difference between few-shot learning and other machine learning problem is that in few-shot learning, E has only a few samples [25]. In a normal set-up of few-shot learning, there is a base class (or base dataset) where abundance of data is collected so that the model can learn meaningful features. Apart from the base dataset, there also exists the novel dataset, which contains completely new labels (although not necessary). The novel

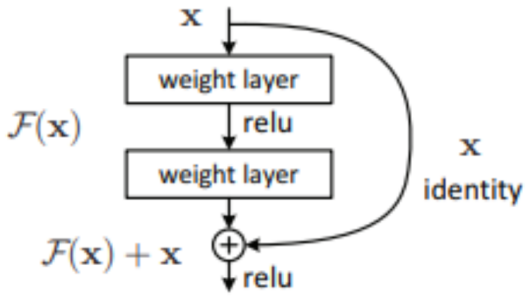


Figure 2.5: Residual Block [8]

dataset is also splitted further into support set and query set. Support set contains k labelled data points for each of n classes where k is small and normally 1,5, or 20, we call this kind of dataset n ways k shots. Query set contains data of the same classes as support set and is used for testing the model.

Different ideas have emerged to tackle the problem such as augmenting the data, using different model architectures, or changing the algorithm used with a model. In this part, I will briefly introduce the background on the strategies I have used for the project as well as the previous works done on Han Nom documents analysis.

2.2.1 Transfer learning

Traditional machine learning techniques work under the assumption that the distribution as well as the feature space of the training and test set are the same. When meeting a distribution shift or feature space shift, it is likely that a new dataset with large amount of data should be re-assembled. In the Han Nom project's however, there are only a few training data for each characters and it is not possible to sample more data of the same class with the same image's characteristic. Therefore, I make use of another artificial dataset to pre-train the model as will be explained in later sections. The problem with this approach is that I have to find ways to adapt the model pre-trained on the artificial data to the book dataset, which is called Transfer Learning.

Transfer learning is a technique where a model pre-trained on one task is applied to train another task. In situations where there is a lack of data, the model is pre-trained on another dataset with similar characteristics, so that when applied to the current task, the model can achieve decent results with less data and training.

Machine Learning takes longer and more resources for the model to converge since it only trains with 1 dataset only. On the other hand, the pre-trained model in transfer learning already learns some specific features when training with the previous dataset. As a result, it converges faster with less data. However, the dataset for pre-train (called the source dataset) and the current dataset (called the target dataset) should have some common characteristics. For example, in my experiment, I generated a source dataset of Chinese characters image and with image size of 64×64, closely similar to the book dataset.

Different types of Transfer learning are classified based on 3 main criteria: whether the source and target data is labelled or unlabelled, in the same or different domains, perform the same task or different tasks. Two datasets have the same domain if they have the same feature space or marginal distribution, for example my artificial dataset and book dataset both have the image size of 64x64. And two datasets are used for the same task when they have the same label space, even if their domains are different. Based on the aforementioned criteria, according to [17], 3 categories of transfer learning are listed as below:

- Inductive Transfer learning refers to cases when the target tasks and the source tasks are different but the domains are the same. Inductive Transfer Learning can also be classified further. If the source and target tasks both contain features and labels, then it is similar to multi-task learning, else if the source task does not contain label, it is similar to self-taught learning.
- Transductive Transfer learning is quite the opposite of Inductive Transfer learning. While inductive transfer learning datasets have the same domain but different task, transductive transfer learning datasets have the same task but different domain (different in feature space or feature distribution). One more characteristic of Transductive Transfer Learning is that the source dataset contains abundant of data. On the opposite, the target dataset does not contain data with label.
- Unsupervised transfer learning refers to cases when both the source and dataset tasks do not contain labels.

Our approach belongs to the class of Inductive Transfer Learning since our source dataset and target dataset have the same domain but different set of classes. There are different solutions to tackle Inductive Transfer Learning problem.

A solution called Instance Transfer uses the source dataset along with the target dataset in order to boost the performance of the model. Example of Instance Transfer is Adaboost [26].

A solution called Parameters Transfer uses the same settings as Multi-task learning, in which different tasks and dataset share some parameters. Typical examples of Parameters Transfer can be found in [11].

Another solution, which is my approach, is feature representations transfer. From the source dataset, I apply models with different kinds of losses to learn the representa-

tions. The models are then fine-tuned to produce best performance on the limited book dataset.

2.2.2 Learning Image Representations

One of the most popular and effective techniques in transfer learning is learning image representations. Traditional deep network contains hidden layers and output layer. The last layer serves the predictive role and the hidden layers learn different features of the input. Learning Image Representation is only different from traditional learning method that instead of using the output layer, it uses different techniques the match the feature vector or feature map of the previous layers. However, of course, we would want our representation to have certain different characteristics. Different characteristics of representation vector requires different types of training. The ways I apply different image representations technique on Han Nom dataset is further explained in chapter 3.

CHAPTER 3. Methods

For general task of character recognition, a reasonable approach is to do a standard supervised learning for classification problem, where each class corresponds to a distinct type of characters. However, the exact total number of characters in Chinese, the most spoken language, is unknown, but they can generally be around 50000 or even more (though most of them are obsolete and rarely used), and as Han-Nom has a deep link to Chinese, we can take it as a rough estimate of the upper bound of the number of Characters that the model should be able to detect. As a result, to train such a large classifier requires a lot of labeled data, which is hard to achieve with Nom as there are only few Nom scholars nowadays. On the other hand, with a fixed number of characters, the model needs to be retrained whenever a new type of characters is recorded, as new Nom documents are recovered. For these reasons, we need a better method that is more flexible and can adapt to new labels with limited data and little changes to the model. Therefore, we decided to approach the problem by learning an embedding space of Han-Nom characters. Such space should encode high-level important information of characters like orientation of strokes and thickness of lines to be able to distinguish between different characters and also to generalize well to unknown characters. With that in mind, we use a model with Triplet loss and Resnet50 backbone, and a model with Manifold Mix-up loss and Wide Resnet backbone to train on our data. Our data consists of two parts, the first part is the data we generate by using Chinese characters combined with a set of fonts, the second part is characters localized from the Han-Nom books shown in the introduction and labeled by EasyOCR. Chapter 3 aims to explain in details how we use models with Triplet and Manifold Mix-up loss to train the embeddings as well as techniques to match embeddings of the same classes. Different methods to learn the embeddings and match the embeddings are synthesized in figure 3.1.

3.1 Learning Embeddings

As explained in related works, learning embeddings is one of the most popular approach in tackling few-shot learning problems and often achieve high results on different benchmark. In representation learning, we might want our representation to capture the most meaningful and rich features of the image, or we might want the representations of similar class to be close to each other in the embedding space. For the first purpose, my model uses Resnet 50 and Wide Resnet backbone pretrained on different popular dataset and apply to further fine-tune on fonts and Han Nom book dataset, so the model already

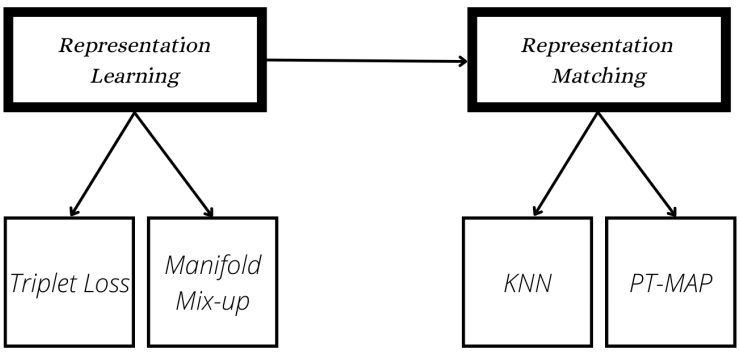


Figure 3.1: Different methods of learning and matching representations

contains rich features in the beginning. For the second purpose, I find and apply two different methods to train on the dataset namely Triplet Loss and Manifold Mix-up. This section will serve as an introduction to the concepts of Triplet Loss and Manifold Mix-up and how I apply these methods on training and validating on the Han Nom documents.

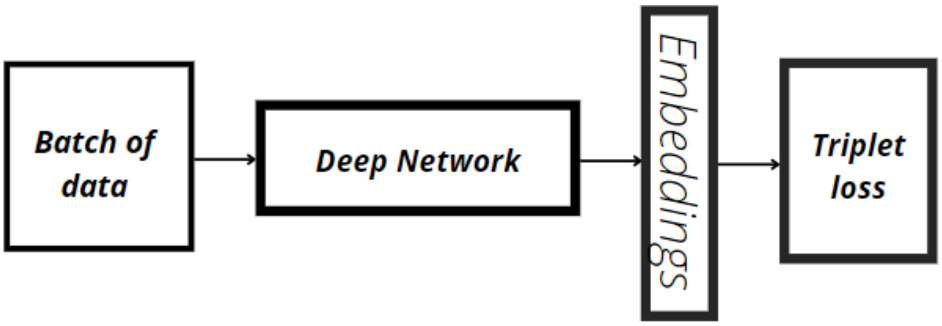


Figure 3.2: Triplet structure

3.1.1 Learning Embeddings with Triplet loss

Triplet learning is the learning method that was first introduced in [19] in the context of face recognition and verification problems, it learns a Euclidean embedding that

abstracts the geometry structure of high level features of the original data. The L2 distances in this learned space directly corresponds to the similarity between original data points. In our problem, similar characters should be close to each other in this embedded space, thus character classification can be done by some clustering algorithms, which in our case is k nearest neighbors.

Triplet method begins with the idea of utilizing an embedding function f, which is commonly implemented as a neural net, to embed an image into feature space, such that the squared distance between characters of the same type, regardless of their writing style and imaging conditions, whereas the distance between pair of different type of characters is large. To make the method an end-to-end learning task, Triplet learning translates this idea into an optimization problem by a special loss function called Triplet loss.

Triplet loss [19] works by choosing a triplet of samples of two similar classes, called anchor and positive, and a sample from another class called negative. Since we want images from the same class to be close in this embedding space and far away to images from different classes, triplet loss directly computes the distance between anchor-positive and anchor-negative pairs and compare the two distances, a margin a is enforced in the loss function to further distinguish between positive and negative pairs. This can be written formally as

$$\|f(x^a) - f(x^p)\|_2^2 + a < \|f(x^a) - f(x^n)\|_2^2$$

The Triplet objective is then the minimum optimization problem of the loss

$$L = \sum_i \max\left(\|f(x_i^a) - f(x_i^p)\|_2^2 + a - \|f(x_i^a) - f(x_i^n)\|_2^2, 0\right)$$

Where x_i^a, x_i^n and x_i^p belong to the set of all possible triplets. This loss function can be easily satisfied by many easy triplets as some negative pairs are easy to tell apart. As a result, these triplets contribute little to the training and can lead to slow convergence and additional mining techniques must be applied to ensure faster convergence. Triplet does this by actively selecting hard positives and negatives, hard positives and hard negatives are pairs that violate the loss function the most, i.e. they make positive distances maximum and negative distances minimum. However, using too many hard examples would lead to slow training as mislabeled and poor condition images dominate the training examples. Therefore triplet uses hard positive pairs and semi-hard negative examples, semi-hard examples are negative pairs that are still further to the anchors than positives but they stay inside the margin. Besides, since positives and negatives are chosen from mini-batches, there should be enough positive examples present in each batch to provide useful anchor positive distances.

3.1.2 Learning Embeddings with Manifold Mix-up loss

In normal deep learning settings, we process inputs and their labels sequentially, however, with the manifold mix up, when training up to a specific layer, the layer and labels will mix up together in a linear fashion. For example, we have input x and x',

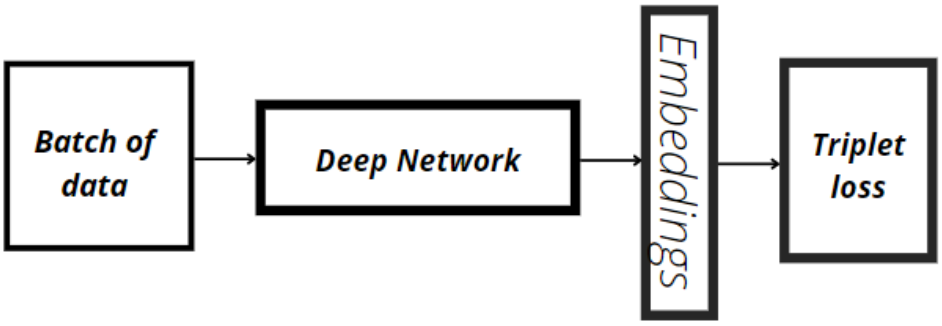


Figure 3.3: Triplet structure

feature vectors at a specific layer are $f(x)$ and $f(x')$, and output y and y'. Then manifold mix-up mixes the feature vectors and their labels together by formula 3.2, 3.3, where y and y' are one-hot vectors. After the mix-up, new batch of data is generated and evaluated.

$$f(X) = \lambda f(x) + (1 - \lambda) f(x') \tag{3.2}$$

$$Y = \lambda y + (1 - \lambda) y' \tag{3.3}$$

After mix-up the data, I use Manifold Mix-up loss as defined in formula 3.4, using symbols defined from formula 3.2 and 3.3.

$$L_{mm} = \mathbb{E}_{(x,y)}(f(X), Y) \tag{3.4}$$

Manifold Mix-up technique is first introduced in [22] as an regularizer to tackle contemporary deep learning problems. For example, they realize that in regularizers using weight-decay [14], dropout [23] and batchnorm [10], the decision boundary is often not smooth enough and thus most predictions are made with high confidence, even with samples that are close to the decision boundary, however, intuitively, we want the samples being close to decision boundary should be predicted with less confident. This weakness makes normal deep learning settings with aforementioned regularizers susceptible to adversarial attacks, even a small change which cannot be captured by the human's eyes can cause the model to predict totally different class with high confidence. Manifold mix-up technique, by smoothing the decision boundary, partly restrains the effect of adversarial attacks. The difference in decision boundaries when compared between Manifold Mix-up and other regularization techniques can be visualized in figure 3.4.

There are different reasons that I decide to experiment with Manifold Mix-up in addition to Triplet loss model. First, in [22], the author has plotted out the decision

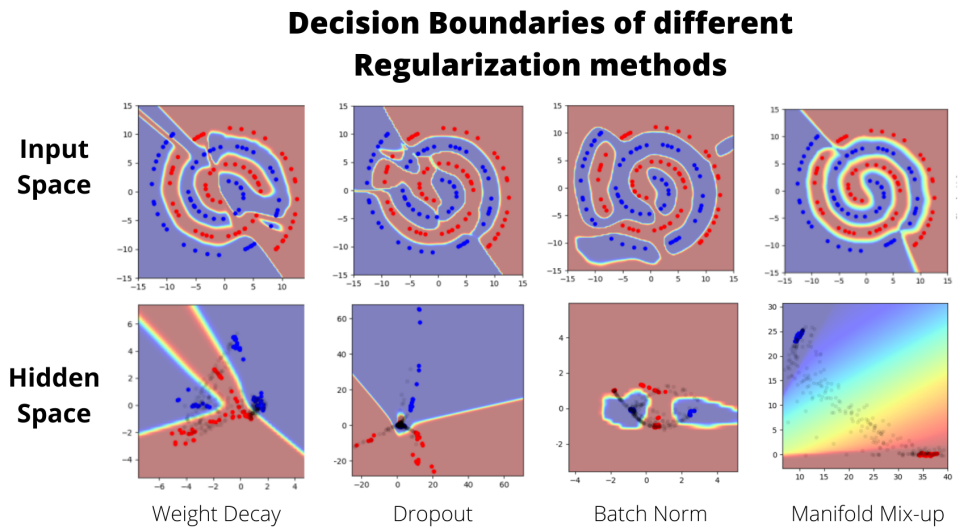


Figure 3.4: Comparison of resulting decision boundaries after using different types of regularization techniques. There are 2 types of decision boundaries visualized here: the first one is from the input space, the second one is from the hidden space (decision boundaries from one layer of the model). The visualizations are taken and edited from [22]

boundary in the hidden space of the model using manifold mix-up as seen in figure 3.4. From observation and proof from the paper, I can conclude that Manifold Mix-up not only improves decision boundaries in the input space, but it also does well on the hidden space and from its representation, we can confidently separate between classes with even simple embeddings matching techniques like KNN, which is partly how Manifold Mix-up is applied in my experiment. Second, it is also stated that model trained with Manifold Mix-up techniques perform well when there is unexpected transformations of the input images, which is suitable since different Han Nom books have different types of page styles and the bounding boxes of the characters are not always nicely cropped.

Because of the ability to learn good representations as explained above, manifold mix-up is also leveraged in few-shot learning settings [15]. Few-shot learning set-up often includes Base Classes and Novel Classes. The Base classes contains sufficient data and labels so the model, or feature extractor can learn meaningful representations. The Novel Classes contain only a few training data with labels belonging to classes that are different from Base Classes, and used to test the representation learned by the model having been trained with Base Classes. My experiment uses the same settings, I use an artificial dataset from Chinese characters and fonts with abundant data as Base Classes to train Manifold Mix-up on. In the training phase, I train on Base Classes in the same way as training traditional machine learning models, with the last layer as a one-hot vector representing probabilities for each class. In the testing phase, I fine-tuned on the characters' images like before, then removed the last layer and used the penultimate layer as embeddings to test the model's accuracy.

By definition, the mix-up happens after the input has been trained through some

layers (the number of layers can be 0). However, I do not know if mixing-up at which layer gives the best result. So I used the charting the right manifold method proposed in [15], a method that randomly chooses which layer to mix-up the data for each iteration. The structure of Manifold Mix-up experiment is shown in figure 3.5.

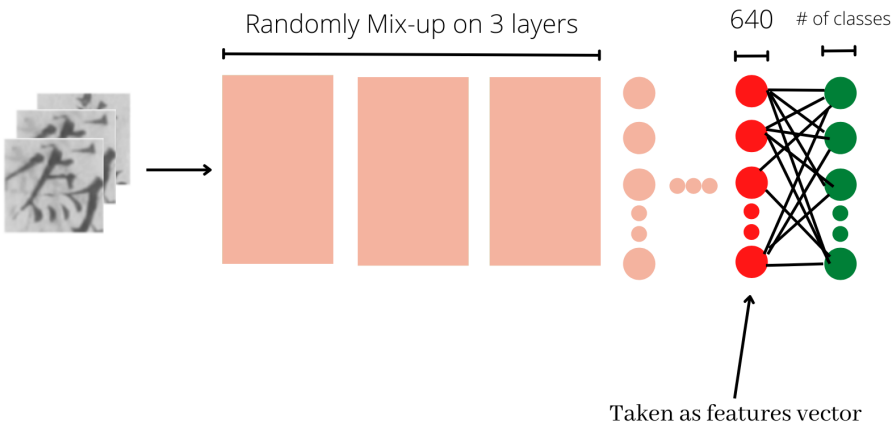


Figure 3.5: Manifold Mix-up model architecture. The first three layers represented by pink rectangles are layers where mix-up between data happen. In the training phase, I use the last layer to classify between classes. In the testing phase, I use the penultimate layer of size 640 as representation vector for the image.

3.2 Matching Embeddings

After getting all the embeddings for the support as well as query images from Han Nom dataset. We need a way to infer the correct class of the query embeddings obtained from the feature extractor. We can look at this problem from multiple perspectives, each perspective gives us different ways to infer the result from embedding space. In one perspective, we consider these embeddings as points in the embedding space, the closest point to the target might be the class the target belongs to. In statistical perspective, each element in the features vector of an image has its own distribution, and if we can have some insights on each element of the features vector, we might be able to infer the correct class, since we assume that each class should have different features vector. We use K Nearest Neighbors (KNN) with L2 distance to implement the first perspective, and utilize PT-MAP to implement the second perspective. This section will be dedicated to explaining the concepts and usage of KNN and PT-MAP in my attempt to classify the Han Nom dataset.

3.2.1 K Nearest Neighbor (KNN)

K Nearest Neighbors [18] is one of the most simple and popular method for classification. With K Nearest Neighbors, we make a prediction by selecting the K closest labelled data points to the target based on some metric. My experiment leverages L2 loss for K Nearest Neighbors algorithm.

K Nearest Neighbors method is based on an assumption that the feature vectors of objects in the same class are close to each other. This assumption holds true for our representation learning method. In triplet loss model, I try to train the embeddings by keeping embeddings of the same class close together while pushing embeddings of different classes further away. Manifold Mix-up is no difference since as visualized in Manifold Mix-up session, even in its hidden layers, Manifold Mix-up still outputs good decision boundary that clearly separates different classes. KNN decisions on top K Nearest Neighbors by L2 loss are visualized in figure 3.6

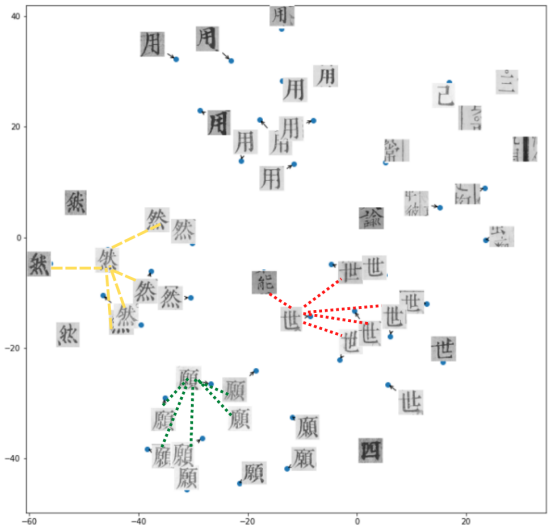


Figure 3.6: Top k nearest neighbors of different characters trained by Manifold Mix-up model. The colored dotted lines are top 5 selections by the specified characters.

3.2.2 PT-MAP

As explained in the previous sections, embeddings training is one of the most appropriate and popular ways to deal with few-shot learning problems. It is also leveraged by many few-shot state-of-the-art models. Many works train the feature extractor on Base Classes to make the model easily adapt to Novel Classes. Since it is not easy to always have qualified Base Dataset, many approaches use pretrained backbone on different dataset like Cifar-FS, Image Net, CUB and apply directly to the Novel Dataset. The problem with adapting a pre-trained backbone to the target task is that the data distribution of different tasks are different, which can cause failure to transfer-based few-shot

learning method. As a result, in [9], the authors propose a way to normalize the features vector distribution called Power Transform (PT), and leverage an effective optimal transport algorithm based on Maximum A Posteriori (MAP) for features vector matching. This section will serve as an introduction to PT-MAP (Power Transform - Maximum A Posteriori) and its effect on my transfer-learning model.

Power Transform (PT)

As mentioned in [9], many works on representation learning assume that the distribution of feature vectors to be Gaussian with little to no proof on it. And in reality, the features extractor's output is often skewd rather than Gaussian-like. For example, in my experiment, after going through Wide Resnet backbone pretrained by Manifold Mix-up, the output feature vectors of size 640 are all left-skewd. Different works have already tackled this distribution problem by some statistical methods [24], [13]. However, the authors of [9] argue that for feature vector of unexpected distribution, these statistical methods can only make the training process worse, and they propose to normalize the features vector to Gaussian-like distribution before using the representations for inference.

Inspired by [9], I use the Tukey Transformation Ladder [21] to "Gaussianize" my left-skewd data. The Tukey Transformation Formula is defined in equation 3.5

$$f(v) = \begin{cases} \frac{(v+\epsilon)^\beta}{||(v+\epsilon)^\beta||_2} & \beta \neq 0, \\ \frac{(v+\epsilon)}{||(v+\epsilon)||_2} & \beta = 0 \end{cases} \tag{3.5}$$

In the 3.5 equation, if we assume that the original vector v is left skewed or right skewed, the Tukey Transformation Formula helps making the distribution more Gaussian-like by the value β. By experiment, I find that setting up $\beta = 0.5$ like in [9] transform my data into Gaussian-like distribution. The distribution of a random element in the features vector before and after Tukey Transformation Ladder are shown in figure 3.7.

I apply Power Transform to the representations learned by Manifold Mix-up of the dataset as a whole. As viewed in figure 3.7, the representations of the whole dataset have become Gaussian-like. And we assume that the feature vectors of each class has Gaussian distribution with their own means and standard deviations. The transformed feature vectors are then used as input to MAP algorithm. The distribution of a random element in features vector of three different classes are shown in figure 3.8

Maximum A Posteriori (MAP)

After applying Power Transform to the representation of the whole dataset, each class's features will be assumed to have Gaussian distribution. Since the features distribution is Gaussian-like and we assume the same for every class, we can assume that there exists a center representation of every class. Finding these center embeddings will be beneficial in classification between different classes. If there is abundant data in each

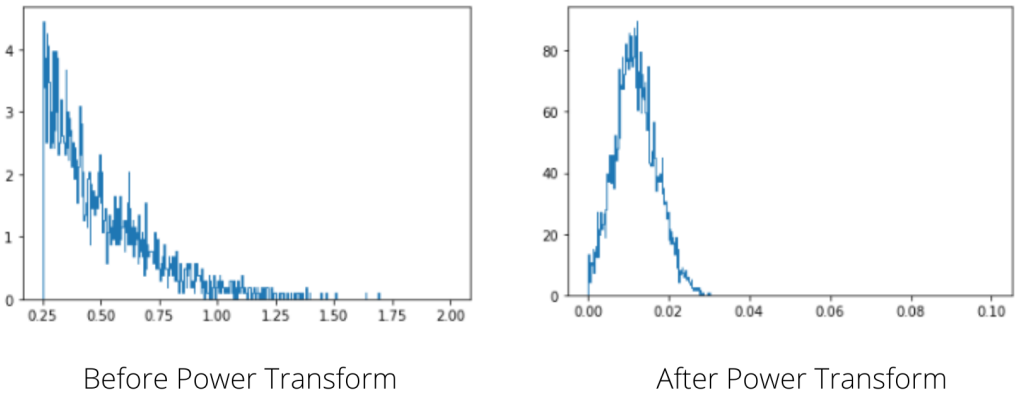


Figure 3.7: Effect of Power Transform on the distribution of a random element from Manifold Mix-up feature vector of size 640 on Han Nom Dataset

class, we can easily calculate the center feature representation by averaging all feature embeddings of that particular class. However, in my situation, I only have 20 labelled embeddings per class (each of the Han Nom dataset's classes have only 20 images with label for training, and 60 unlabelled images for testing), so calculating the center with so few data in one class might not return the center embedding that represent the whole class. The PT-MAP paper proposes an algorithm to iteratively find the center feature vectors of each class as followed:

- First, we set up a matrix of $wq \times w$ where each row sums up to 1 and each column sums up to q (w is the number of classes while q is the number of query images for each class). Intuitively, the rows are the probability distribution of each query image to the classes. The columns sum up to q because each class has q query images.
- Then, we iteratively recalculate the center feature vector of each class, then recalculate the matrix in the previous step, then re-update the center vector. In the end, the matrix represents the probability for each class of each query image. The matrix is re-calculated and updated using Sinkhorn distance [6]

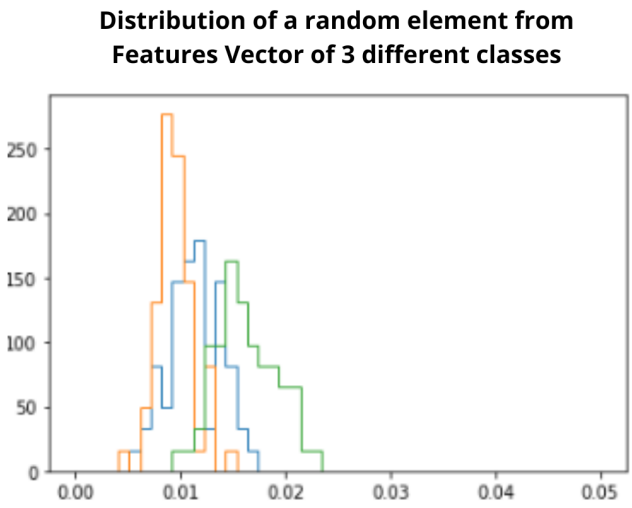


Figure 3.8: Distribution of Random feature of representation vectors from 3 random classes. (from Han Nom book dataset)

CHAPTER 4. EXPERIMENTS AND RESULTS

Chapter 4 presents my experiments and results. The experiment is divided into multiple steps. First, I train the model on a artificial dataset generated from Chinese characters and fonts so that my model can learn the characters' embeddings better. Second, we sample the most popular characters from all books and split them to support and query set, the support set contains only few images for each characters. On the book dataset, we fine-tune our pre-trained model on the support set, then proceed with making prediction on the query set. The details of the steps are demonstrated below.

4.1 Dataset

In order to learn the embedding for character classification, we use 2 types of dataset. An artificial dataset and a book dataset.

The artificial dataset is created from computer encoded Chinese characters and fonts. The dataset contains around 8k distinct characters associated with 25 fonts. Transformations are then applied to the generated dataset to make the model trained more robust. Some techniques I have used are:

- **Color Jitter**: a technique that randomly changes the image's brightness, contrast as well as saturation.
- **Random Box Blur**: compared to the original image, the resulting image after box blurred has the pixel values equal to the average of its neighbors' value in the original image.
- **Random Perspective**: randomly changing the image's perspective. Different perspectives of an image are different representation of an object in 2D scenario.
- **Random Affine**: a technique of transformation that preserves planes, lines and points.
- **Random Erasing**: randomly choose a small rectangle in the image and erase it.
- **Random Resize Crop**: randomly crop a portion of the image then resize it back to the size of the original image.

Different fonts for a character are shown in figure 4.1, while the transformation's results for the characters are shown in figure 4.2. Among all the classes, 6000 distinct

classes of characters are chosen to pre-train the model, and 200 novel classes are picked to evaluate the model's representation learning.

Figure 4.1: Different fonts for a character

The book dataset is created from seven historical Han-Nom books, the details of which has been described in Chapter 1. We use Differential Binarization model [12] to get bounding box for the images. Then, we crop the characters and make them as input to our experiment, the cropped characters for each book can be found in figure 4.3. From figure 4.3, we can see that most characters are correctly cropped, however, in Dai Nam Quoc Su Dien Ca or Truy Mon Canh Huan, due to the nature of the books (characters are written and printed too close to each other), many of the bounding boxes contain 2 or more characters, or the characters do not reside in the center of the boxes. However, since in reality, cases where human make mistakes labelling are always possible, so I still leave the cropped characters as they are. However, later it still turns out that my approach still produces decent results on this kind of dataset.

In order to get the labels of characters, we relied on EasyOCR - an open source tool to generate the pre-labeling from the characters' images. Then, a further crosscheck is carried out to remove invalid classifications: we removed labels that contain non-Chinese characters, such as special symbols and numbers, we also discarded labels whose length is different from 1 since some images contain 2 or more characters. After removal, our dataset has a total of 49k characters as well as their labels. However, to maintain the class balance in the support set and the query set, I have selected 100 most common character classes from all books only. For each class of characters, I sample out 80 characters, in which 20 is for the support set and 60 is for the query set. The distribution of character

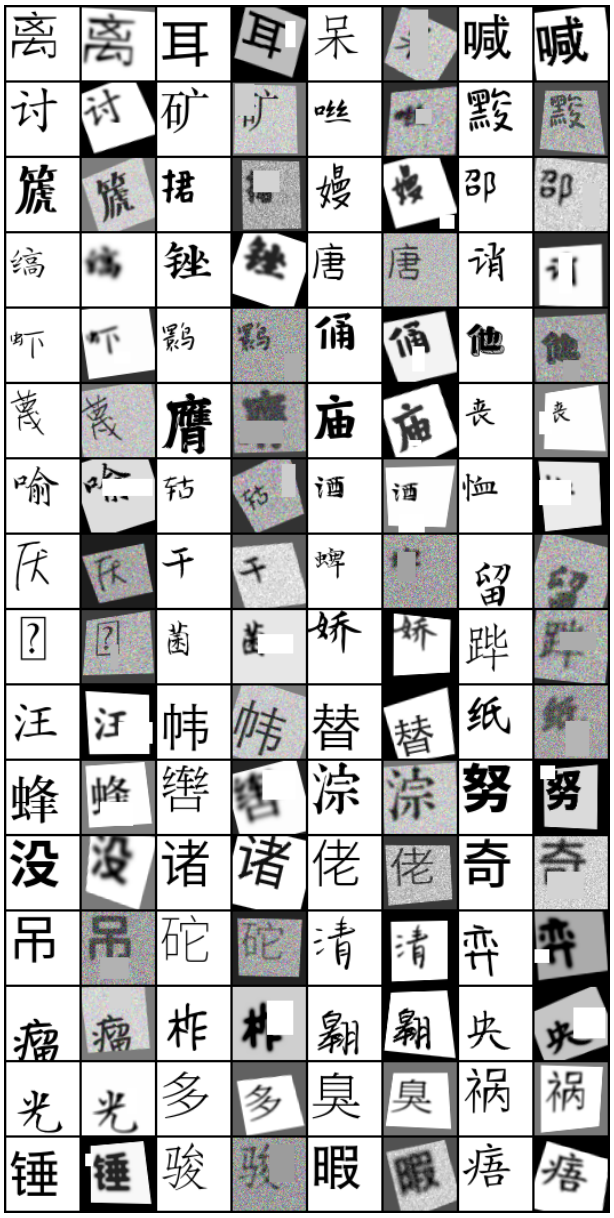


Figure 4.2: Transformations of the images

in 7 Han Nom books are shown in table 4.1

The setting is similar to a setting where users could input a small amount of labeled samples which then can be used to train a classifier to recognize the other images of the label but on a larger scale, this user-provided dataset is usually small and can contain misleading labels.

4.2 Pretrain and evaluate on artificial data

For embedding representation learning, we first train the embedding model on the artificial data generated from 25 fonts and a corpus of Chinese texts consists of 6k characters, making the total of 150k characters in the pre-trained dataset, we augmented the

Figure 4.3: Illustration of uncurated characters on 7 books

Table 4.1: Book distribution of the selected characters

Book	Selected characters	Number of Classes
Dai Viet Su Ky Toan Thu Printed	662	81
Dai Viet Su Ky Toan Thu Handwritten	691	83
Truy Mon Canh Huan	814	97
Khoa Hu Luc	1720	99
Dai Nam Thuc Luc Tien Bien	432	75
Dai Nam Quoc Su Dien Ca	130	47
Phap Hoa De Cuong	3551	100

data with extensive transformations. Two different models are trained with Triplet loss and Manifold Mix-up as discussed in the previous section and serves as the initial weights for later fine-tuning on actual data.

After training the embeddings by different losses, 200 novel classes from the artificial dataset are sampled for the testing purpose. In this phase, we separates 25 fonts in each class into support and query set, the support set contains 5 fonts and the query set contains 20 fonts of each class.

For the model trained with Triplet loss, I run the query images through the model and compare the distance from them to the support images' embeddings. Top k closest labels are then chosen for prediction and accuracy calculation. L2 distance is employed for this task. The triplet model tested on fonts dataset gives promising results when it achieves 89.4% for top 1 closest label and 97.7% for top 5 closest labels.

For the model trained with Manifold Mixup loss, I use both K Nearest Neighbors and PT-MAP to select the best character class for prediction. For both techniques, Top 1 and Top 5 closest labels are also selected for accuracy selection. The K Nearest Neighbors technique gives around 95.9% for top 1 and 97.6% for top 5, while PT-MAP techniques give similar result of around 95.5% for top 1 and 96.5% for top 5.

To better understand the insights of the learned feature space in the this step, I conducted a small visualization on the pretrained model (with Triplet loss) to gain some insight of the embedding space. I picked randomly 12 characters on the book dataset and generated these characters with fonts that we used to pretrain the model. The embedding of these characters, both fonts and books, are then processed by projecting them with tSNE on a 2D plane, the result is shown in figure 4.4. We observed that the model learned good representations of the generated characters as they are nicely separated on the projected plane, but it also indicates that this space does not work very well with actual data as all the actual 12 characters from books are grouped into the same cluster. This suggests that we need an additional fine tuning step later to improve the performance on the actual book data.

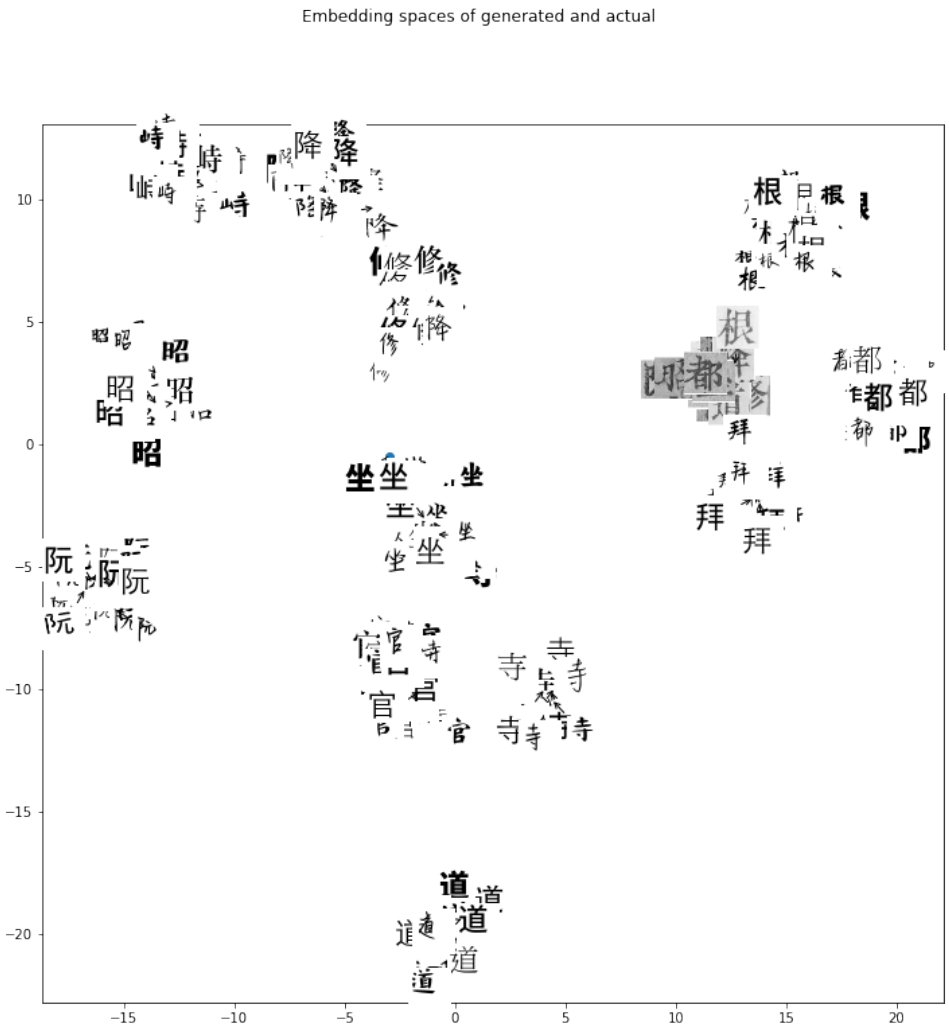


Figure 4.4: Visualization of the embedding space before the training step on mixed data. All 12 generated characters from fonts are seperated on different groups but actual characters all go to the same group. This suggests the need of an additional fine tuing step.

4.3 Fine-tune and evaluate on Han-Nom books dataset

In the next step, the book dataset is used. In order to get the ground truth labels for learning, I used Easy OCR to extract raw predictions and clean them to make it better suit for training purposes. For label cleaning, I only kept the labels that satisfied some predefined conditions that I thought might help removing irrelevant and misleading labels, such as predicted string length, occurrence of special characters, bounding box ratio and so on. After this step, the total number of characters I acquired from all books is around 49k. Among them, I picked images which belongs to 100 most common characters and splitted them into support and query sets. The support set contains 2000 characters' images, belonging to 100 classes, each class has 20 images in the support set. The query set contains 6000 characters' images, belonging to 100 classes, each class has 60 images in the query set. The distribution of images in the support set and in the query set are visualized in figure 4.5.

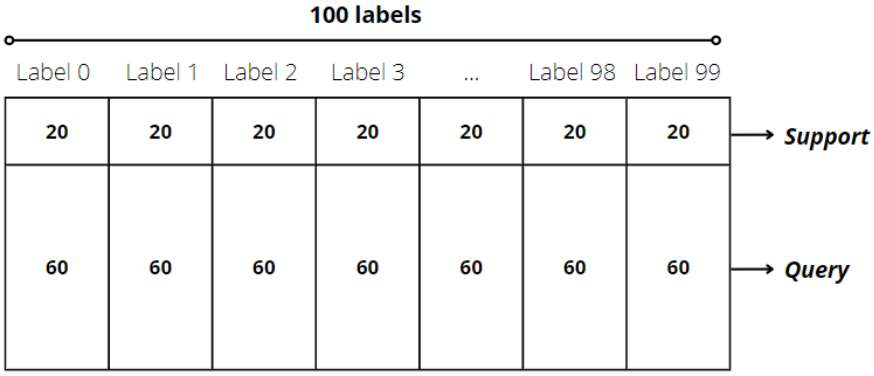


Figure 4.5: Support, query set split from books

After separating the support set and the query set, two models using Triplet loss and Manifold Mix-up Loss are fine-tuned on the support set using the same loss as when they are trained with fonts. Then, the query images are passed through the fine-tuned models to produce embeddings. The query embeddings are then compared with the support embeddings by different strategies. With triplet model, k Nearest Neighbors are employed to pick embeddings from the support set which are the closest to the target embeddings. With model trained using Manifold Mix-up loss, both k Nearest Neighbors and PT-MAP are used to match the embeddings. The details of K Nearest Neighbors and PT-MAP algorithm are described in Chapter 3.

4.4 Results

As explained in the previous sections of Chapter 4, my experiment includes the following steps:

- Pre-train embeddings model on artificial fonts dataset. The artificial dataset has 6000 Chinese classes with 25 elements for each class.
- Test on 200 novel classes from the artificial fonts dataset (no fine-tune)
- Fine-tune and evaluate on 100 classes of characters from book dataset, each class has 80 elements, 20 for support set and 60 for query set.

The test results on 200 novel characters in artificial dataset are shown in table 4.2. The results show that the models have adapted well and are able to clearly separate the representations for different classes, even for classes that it has never seen before (figure 4.4). However, when directly applied to another dataset with different characteristics like the book dataset, further fine-tuning step are needed as explained in section 4.2.

Table 4.2: Experiment results on font dataset

Type of Loss	Comparison technique	Top 1	Top 5
Triplet	KNN	89.4	97.7
Manifold Mix-up	KNN	95.9	97.6
Manifold Mix-up	PT-MAP	95.5	96.5

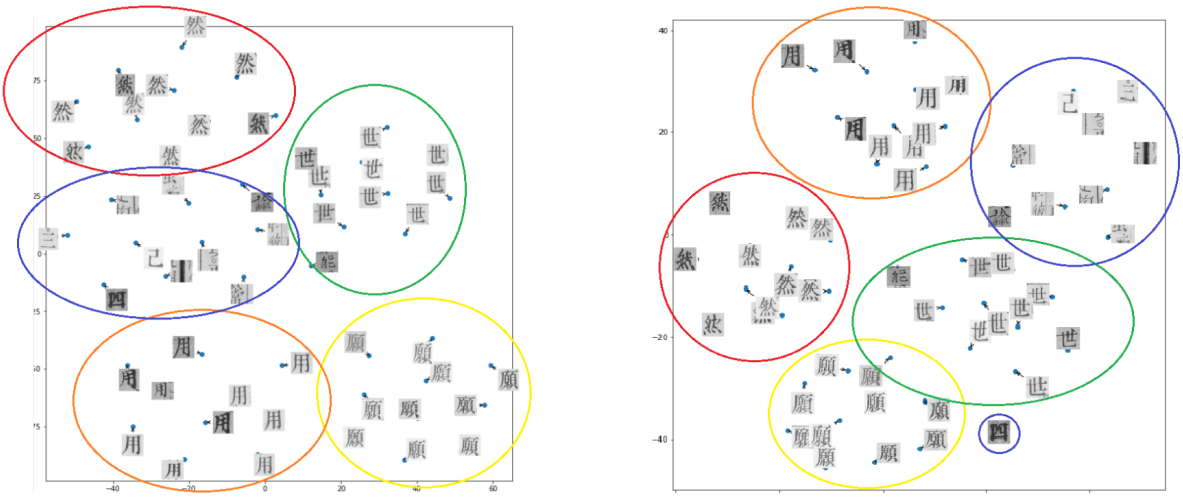
After fine-tuning on the book dataset, the models trained with Triplet loss and Manifold Mix-up loss return results with high accuracy. I fine-tune the Triplet model on the book dataset for up to 50 epochs and fine-tune with Manifold Mix-up for 10 epochs only. To prove that my approaches outperform traditional machine learning methods, I conduct another experiment with Resnet50 using only limited support and query sets from the 7 books. The experiment results on the book dataset are shown in table 4.3.

Table 4.3: Top-1 and Top-5 accuracy results in each book with our proposed methods (Triplet and Manifold Mix-up (MM) models with K Nearest Neighbor (KNN) and PT-MAP matching techniques) and training with Resnet. We use number to denote different books as followed: 0 is Phap Hoa De Cuong, 1 is Truy Mon Canh Huan, 2 is KhoaHuLuc, 3 is Dai Nam Quoc Su Dien Ca, 4 is Dai Nam Thuc Luc Tien Bien, 5 is Dai Viet Su Ky Toan Thu (printed), 6 is Dai Viet Su Ky Toan Thu (hand-written)

Book	**Resnet50**		**Triplet + KNN**		**MM + KNN**		**MM + PT-MAP**	
	Top 1	**Top 5**	**Top 1**	**Top 5**	**Top 1**	**Top 5**	**Top 1**	**Top 5**
0	93.6%	98.0%	98.6%	98.8%	99.0%	99.1%	98.9%	99.8%
1	67.4%	83.7%	80.9%	82.4%	83.7%	90.0%	82.7%	94.6%
2	90.0%	97.0%	96.7%	97.0%	97.4%	97.8%	97.4%	99.2%
3	15.8%	35.6%	34.7%	44.6%	38.6%	55.5%	44.6%	73.3%
4	80.2%	88.0%	90.4%	91.3%	91.9%	93.7%	91.3%	96.7%
5	67.8%	82.9%	83.5%	86.1%	86.1%	90.0%	86.5%	95.1%
6	51.9%	71.6%	73.7%	76.9%	78.2%	84.5%	75.9%	92.2%
Overall	82.5%	91.2%	91.4%	92.5%	92.8%	94.8%	92.6%	97.5%

The results table show the performance of different pretrain models as well as embeddings matching techniques on the book dataset and compare them to Resnet 50. All the of fine-tuned models with representation learning techniques achieve accuracy scores greater than 90%. This means that with a few fine-tuning epochs, the representations learning models have efficiently separate the classes by their embeddings. To validate this, I also plot the embeddings of images from 5 random characters (from the query set) after going through models with Triplet loss and Manifold Mix-up Loss (figure 4.6).

3 observations can be made on figure 4.6. First, in both embedding spaces, images from the same class tend to concentrate in one place and separate from the other classes, hence simple matching techniques like K Nearest Neighbor can already produce decent results. Moreover, from the visualization of the embedding space, we can observe that the Manifold Mix-up model's embedding space looks much more concentrated than embedding



Embeddings by Triplet Loss Model *Embeddings by Manifold Mix-up Loss Model*

Figure 4.6: Visualization of the embedding spaces after fine-tuning on the support set of 7 Han Nom books. I pick out 5 random classes of character, then sample 10 images in the query set per class (which means the models have never trained on it) and plot out their embeddings on the 2D planes. Ellipses of different colors represent different classes. The 2 figures present 2 embedding spaces produced by 2 models trained by Triplet and Manifold Mix-up loss on the same set of images.

space of Triplet model, which can also explain the superiority of model trained by Manifold Mix-up loss when compared to model trained by Triplet loss. Second, we can see that the blue ellipses from both embedding space contain mistaken localized characters, which I still allow to be in my dataset to make the experiment closer to real life scenario. The mistaken localized characters to some extent can affect the models' learning procedure. But despite of that, the models still learn good representations, as seen in the visualization in figure 4.6. Third, if we compare between embedding spaces on the book dataset in figure 4.6 and the font dataset in figure 4.4, the fonts' embeddings are clearly more concentrated, thus results in higher accuracy in the font dataset compared to the book dataset.

Another observation is that for top 1 accuracy, models using representation learning techniques surpass traditional resnet50 by 10 to 15% (600 to 900 more correct characters) with less training (10 and 50 epochs compared to Resnet 50's 100 epochs). However, the accuracy of Resnet 50 model on Han Nom book is already acceptable. That raises the question of whether or not I need to gather artificial data and pre-train the representation learning models. In this experiment, my approach beats traditional learning methods by 600 to 900 images. But in reality, there are hundreds of Han Nom documents have not been labelled, and 10% accuracy in my experiment is much larger in reality, 600 to 900 images difference can become thousands. As a result, more human labor is needed to re-label the wrong one. Moreover, since Resnet 50 is a traditional machine learning model, it can only classify between a fixed number of class. When images from new class appear, it will have to retrain the model. My representation learning models, on the other hand, learn a vector to represent characteristics of an image, so when a new label appear, we

do not have to retrain the model but only need to output embeddings from the existed model, then compare the embeddings with other images.

Next, we will look at some wrong predictions by the models. Figure 4.7 shows typical examples where all models return wrong results. The total number of wrong predictions account for about 2% of all query sets. From the figure, we can observe that most of the wrong predictions here are wrongly labelled, or the image contains multiple character, which confuses EasyOCR in the labelling process. Only a small portion of the wrong predictions from all models are correctly labelled. So if we have a more thorough process of labelling the data, the result can have been even better.

Figure 4.7: Wrong predictions by all models (split by book)

In conclusion, in this chapter, I have presented my experiment pipeline and show the experiment's results. Representation learning methods have shown its promising capability when achieving over 97% over a typical Han Nom documents dataset. The approach also opens up a new way for us to both efficiently label large amount of Han Nom dataset with less human labor and in the mean time receiving recommended labels from users without having to retrain the model like in traditional machine learning.

CHAPTER 5. MODELS DEPLOYMENT

Chapter 5 presents my deployment of Triplet model for Han Nom text recognition by emphasizing the main purpose of the system, main features and technology used for deployment. The system is created with two main purposes: (1) Deploy the Model Architecture for detecting characters in Sino-Vietnamese and (2) Provide users with an interface to interact, extract characters' region, suggest labels, and use the information for further purpose such as converting books into digital format for reading and research.

5.1 Technology Overview

In this section, I briefly introduce frameworks and technology that I used to build our system. The system is a web-application that serves both as a place for storing Han-Nom documents and assisting the process of digitizing books. I used ReactJs to create the front end of the application and Mongodb and Pytorch with Python to build the back end, including the database and deep learning model. Following sections briefly touch on these technologies.

MongoDB

My project makes use of MongoDB [3] store the data. MongoDB is a type of document database. The term Table in MySQL is similar to Collection in MongoDB. A collection in MongoDB consists of a list of documents, each document is structured like a JSON object (in key-value pairs). The values of a field may consist of arrays and nested documents. Moreover, MongoDB documents may have dynamic schema, which makes it much more flexible than MySQL. The comparison between RDBMS and MongoDB are shown in table 5.4

Restful API

REST (Representational State Transfer) is a software architecture which defines constraints in the Web Services Development Process. Applications designed based on REST architecture are called RESTful. REST API allows different systems communicate, send and receive data in a simple way. It regulates the usage of HTTP method (GET, POST, PUT, DELETE...) and format URL for Web Services to manage the resources.

Table 5.4: Summary of the difference in terminology used in the two database systems.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Default Primary Key

REST has constraints such as: stateless, cachable, client-server architecture, Layered system, and uniformed interface. In our system, we used ExpressJs [2] to implement the Restful architecture between client-server.

React

React [4] is a library written in Javascript, used to build user interface. It allows code reusability within project so we can avoid code duplication. React is maintained by Meta and often used to create Single Page Application.

5.2 Main Features and Demo

The goal in building the system is to create a platform to store, manage and assist the digitization of Han-Nom documents. The system has two main features: first, it allows users to upload books to the server, users can later read the books they have already uploaded and second, it helps convert and digitize the uploaded Nom documents using machine learning techniques that we discussed in the previous sections. Additionally, the system also has an user management system that permits users to register new accounts. Users with account will have full access to the mentioned features: they can upload books, read books and use character localization and detection for their documents. (Figure 5.1, 5.2)

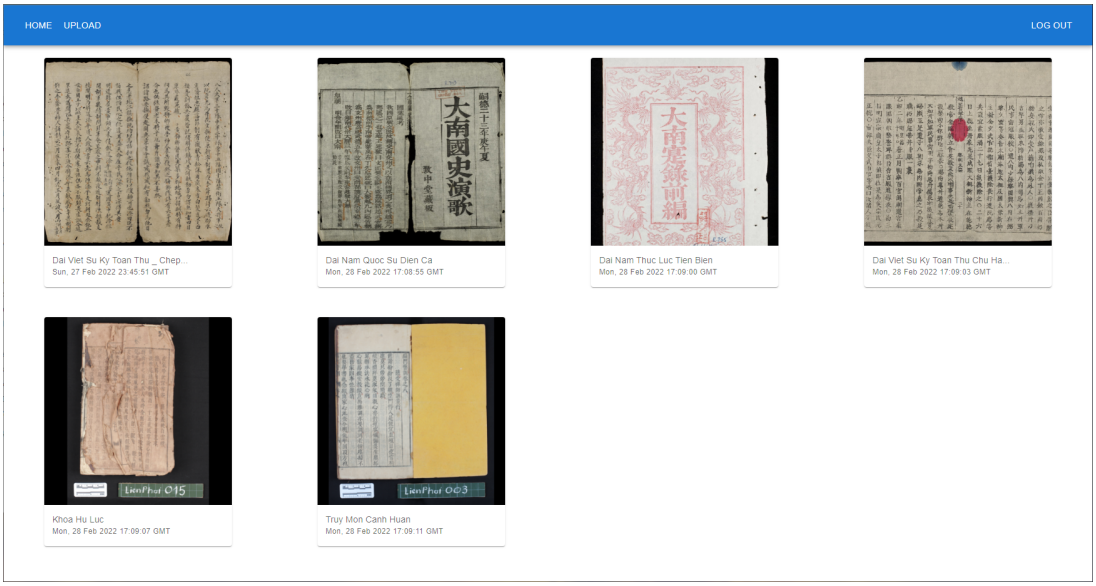


Figure 5.1: Main page.

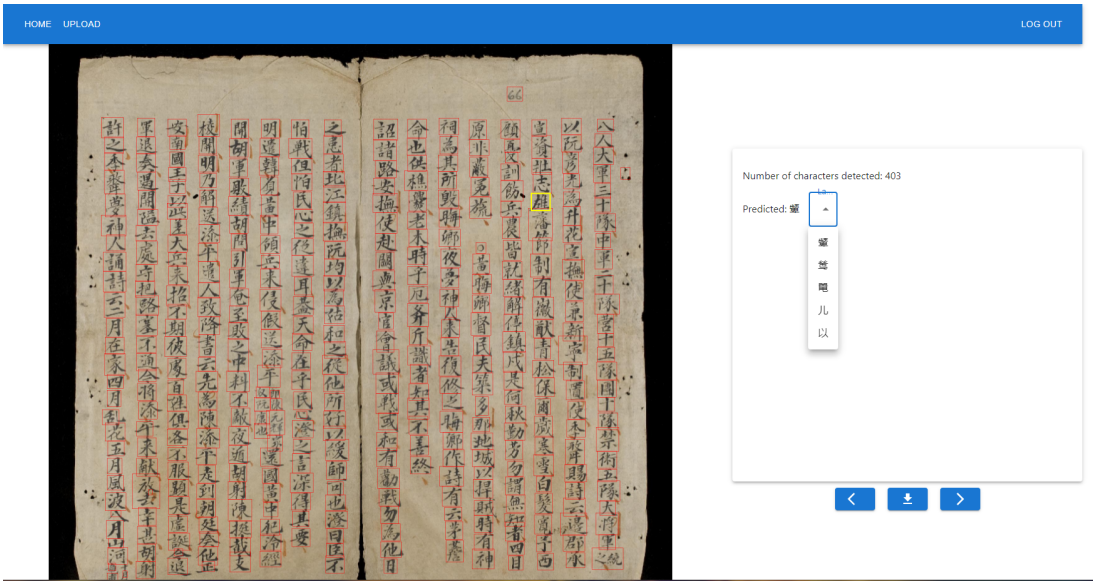


Figure 5.2: Reading page, each page is shown along with bounding boxes and suggested labels.

CHAPTER 6. CONCLUSION AND FUTURE WORK

In this report, we have proposed an approach to deal with the currently large amount of unlabelled Han Nom dataset. We make use of representation learning techniques for characters classification. Both models can easily adapt to new dataset given a few sample.

The carried out experiments show promising results on the proposed the method. The method achieves high results on real world data with limited training data. From the original data of seven Han Nom books, we make use of the open source OCR to localize and pre-label the characters. Then I clean the labelled images, filter and split them in accordance to few-shot setting for the characters classification task. In classification tasks, embedding learning gives good results when the Triplet models give top 5 accuracy up to around 93% on average although being trained on a small amount of data, and the models trained with Manifold Mix-up loss give top 5 accuracy up to around 98% on average.

In addition to the proposed learning method, we also built a web-application that assists users on storing and analyzing Han Nom documents. The initial application has basic use cases that allows users to login, upload books and extract character boxes as well as their suggested labels. In the future, we will focus on extending the application's new features, for example to record the corrected labels from users and add new types of characters to the database.

Given the promising results of our approach, the work could be extended in several directions. Firstly, the dataset could be added with more character and books. Our approach allows the usage of unlabelled data, which is an advantage to the traditional approach. The second one is the direction of using domain adaption methods which consider each book is a specific domain. The embedding learning process are designed to adapt from well-known domain to the new one.

Our work is going to be submitted as a paper with title: *Learning Embeddings for Recognizing Han-Nom Characters in Vietnamese Historical Books*

Bibliography

[1] The essential guide to neural network architectures. https://www.mongodb.com/. Accessed: 2022-01-03.

[2] Express js home page. https://expressjs.com/. Accessed: 2022-01-03.

[3] Mongodb home page. https://www.mongodb.com/. Accessed: 2022-01-03.

[4] React home page. https://reactjs.org/. Accessed: 2022-01-03.

[5] Under-fitting and over-fittign in machine learning. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. Accessed: 2022-01-03.

[6] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. Leveraging the feature distribution in transfer-based few-shot learning. In *International Conference on Artificial Neural Networks*, pages 487–499. Springer, 2021.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[11] Neil D Lawrence and John C Platt. Learning to learn with the informative vector machine. In *Proceedings of the twenty-first international conference on Machine learning*, page 65, 2004.

[12] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, and Xiang Bai. Real-time scene text detection with differentiable binarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11474–11481, 2020.

[13] Moshe Lichtenstein, Prasanna Sattigeri, Rogerio Feris, Raja Giryes, and Leonid Karlinsky. Tafssl: Task-adaptive feature sub-space learning for few-shot classification. In *European Conference on Computer Vision*, pages 522–539. Springer, 2020.

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[15] Puneet Mangla, Nupur Kumari, Abhishek Sinha, Mayank Singh, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Charting the right manifold: Manifold mixup for few-shot learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2218–2227, 2020.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[17] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[18] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[19] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[20] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[21] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[22] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[23] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. *Advances in neural information processing systems*, 26, 2013.

[24] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019.

[25] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning, 2020.

[26] Yi Yao and Gianfranco Doretto. Boosting for transfer learning with multiple sources. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 1855–1862. IEEE, 2010.

[27] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.