Language script identification using bag of visual words

Abhyudai Nouni
University of Massachusetts Amherst
anouni@umass.edu

Ayush Sharma
University of Massachusetts Amherst
asharma@cs.umass.edu

Abstract

Script and language identification are precursors to modern multilingual optical character recognition (OCR). The task of visual script recognition is often non-trivial since we are analyzing images containing text and not the text itself. It poses an interesting task for researchers to find ways to represent the uniqueness of a script in tangible terms and to compute differences or similarity between these representations. In this project, we have looked at the problem of classification of images containing text based on the structural patterns of these scripts.

1. Introduction

In the multitude of languages used all over the world, we encounter many different writing systems, also known as scripts. It is almost imperative to know the script of a language to read or understand a particular text written in that language. There are up to 36 different kinds of written scripts that are actively being used today. Our project is aimed at identifying scripts from images of printed text. We chose 11 unique scripts and built a model that would classify a text image as one of these 11 scripts.

For the purpose of our experiment, we have considered text which is composed entirely of a single script. If we can identify that most of the characters in a text belong to a particular script, then there is a high likelihood that the entire text belongs to that script. Our approach can be broadly divided into three distinct parts: extraction of features from the text images; generating a bag of words model from these features; and using a discrimnative model to classify images based on their bag of words representation.

The task of detecting scripts from images lies in the domain of object recognition. We propose different a bag of visual words model with SIFT (Scale Invariant Feature Transform) features as visual words for addressing our problem of identifying script from images of text. We will compare it with other feature descriptors like histogram of gradients (HOG) and local binary patterns (LBP). We will also compare different classification techniques like nearest neighbor, logistic regression, random forest and support vector machines.

We will introduce a new data set for our experiments and describe the process of how we generated it. For the scope of this project, we have focused on a limited set of fonts per script class. However, we also analyzed the performance of our method on images of text with a variety of different font styles and suggest directions for future work.

The challenges which we had for this task include building a data set from scratch, identifying which features work best for classification, generating a meaningful visual vocabulary and dealing with similarity of features across languages.

2. Related Work

In the domain of document image analysis, script identification has been widely researched upon and dealt with in different ways. There are two broad categories in which these methods can be grouped under - structure based and visual appearance-based. In the first category, [Spitz and Ozaki, 1995] proposed a method for page-wise script identification by using vertical distribution of upward concavities and optical density in connected components. [Lee et al, 1996] and [Waked et al, 1998] among others have also built on top of the work done by Spitz by adding more connected-component based features.

For methods which do not use segmentation or structure based techniques, they tend to rely on the overall visual appearance of scripts, rather than breaking down the script into characters and strokes. [Wood et al, 1995] experimented with the use of horizontal and vertical projection profiles of images of entire document pages. More recent research methods like [Tan, 1995] and [Pan et al, 2005] use texture features like Gabor filters.

[Sharma et al, 2015] proposed several techniques using bag of visual words for word-wise script identification in video-overlaid text. They extracted patch based SIFT descriptors, which were then combined to generate a bag of features (BOF). They also use spatial pyramid matching (SPM) with the SIFT based descriptors. They found that collecting SIFT features using SPM results in superior accuracy as opposed to other feature descriptors like HOG or LBP.

[Nicolaou et al, 2016] showcased a technique to extract texture features which produce state of the art results in script identification for video overlaid-text as well as scene-based text. They use hand-crafted features and a variant of LBP with a deep multi-layered perceptron algorithm and perform K-nearest neighbor classification. They make use of an interesting preprocessing technique wherein they replace the image with its negative, in order to get consistent LBP encoding, whenever the central band is darker than the image average.

While there is a lot of research in the field of script identification for document analysis, recent papers have explored many different domains. [Gllavata et al, 2005] was one of the first ones to deal with the task of script identification on non-traditional paper layouts. They proposed a wavelet transform representation of images which detects edges in images with overlaid-text. More recently, [Gomez et al, 2016] look at script identification in scene text images, where they have proposed a patch-based classification framework which uses ensembles of conjoined networks. They employ deep CNN architectures in order to learn discriminative representations for the individual image patches. A few works have also tried to identify scripts "in the wild". [Shi et al, 2015] have used multi-stage spatially sensitive pooling (MSPN) which is a variant of CNNs for this purpose.

In this project, we approach the task taking cues from the the work of [Sharma et al, 2015] for feature extraction and quantization. We used a bag of words model and experimented with different feature extraction and classification methods. The details of the architecture of our model and the results of our experiments are described in the detail in the next sections.

3. Overview of the Bag of Visual Words model

The bag of visual words model is borrowed from the bag of words model used in natural language processing. Here we use features extracted from the images as visual words. To make a bag of words model, we need to define words. This is achieved by the following steps:

3.1. Keypoints detection

We use a Laplacian of Gaussian to find points of interest or keypoints in the image ([David G. Lowe]). The radius of the blobs denoted by r depends on the value of σ (parameter discussed in next section) by the following relation:

$$r = \sqrt{2}\sigma$$

The radius of the blobs are taken over several scales so that it can extract words of small as well as big fonts. We use the Harris Corner detector to eliminate edges in the keypoints selected. The Harris corner detector is given by the Harris matrix H.

$$H = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

where I_x and I_y are derivatives in x and y direction over all the pixels.
A score is calculated for each keypoint, to determine if it can possibly contain a corner:

$$R = det(H) - trace(H)^2$$

A keypoint with a score greater than a threshold is estimated to be a corner.

3.2. Feature representation

We used the following feature representations for the keypoints detected:
a) *Scale Invariant Feature Transform (SIFT)*: We represent the keypoint detected by a 128-dimensional feature vector. We divide the image into 16 grids and use 8 direction bins to represent the oriented gradients. After this step, each image is a collection of vectors of the same dimension, where the order of different vectors is of no importance.
b) *Local Binary Patterns (LBP)*: We divide the patch into 3x3 patches around every pixel and extract 8 bit LBP features from the image. In a 3x3 patch, we assign the value of 1 to the pixel whose value is greater than or equal to that of the center pixel and 0 to those whose value is less than that of the center pixel. We then then make a binary string from these 8 pixels(leaving the center pixel) and the decimal represntation(0-255) is the representation of the center pixel. In this way, the binary represntation of each pixel is computed and the histogram of these representations is the LBP representation of the image.

3.3. Making the codebook

We perform normalization of the vector representation we obtained in the previous part. We run K-means clustering on the features extracted. We then use these clusters to train our classifier. K-means uses L_2 norm to cluster

vectors. The final step in training involves generating histogram representations of training images in terms of visual words in the codebook.

3.4. Training the Classifier

We use classifiers with the goal of statistically classify an object's characterisitcs to identify which class (or group) it belongs to. We choose different classfiers for training our model. For our project, we have experimented with the following classifiers:

a) *Linear Classifier*: Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems. We use softmax function for classification. The hyperparameters are discussed in the next section.

b) *Nearest Neighbour classifier*: The K-nearest neighbour classifier is a very simple classifier. The idea here is that, to classify a query point X find its closest k neighbours and assign the majority label to the query point X. It is conceptually simple and takes less time to learn. It is also effective when there is a lot of data and there is a good distance metric.

c) *Random Forest classifier*: In random forest classifier, we take multiple decision trees and use bootstrapping to get an ensembled result. We randomly select K features from the feature vector and train a decision tree on it. We do this for many trees and then while testing take a majority voting on for the final prediction.

Figure 1: Keypoints detection in the images

4. Our Approach

4.1. Building the codebook

For our project, we created our own dataset which has imag.es of 11 languages. The code was implemented in MATLAB(academic license).

The keypoints were extracted using the Laplacian of Gaussian described in the previous section. The starting σ value was 3.0 and the number of levels used were 6 incrementing the σ by a factor of $\sqrt{2}$. The threshold for the response was 0.05 i.e only the blobs with response value ≥ 0.05 were selected. Further, blobs which contained only edges were eliminated by Harris corner detector using threshold value 12. (See Figure 1).

The SIFT and LBP features were extracted as described in the previous section. These features were subsequently normalized by dividing the feature vector with its L_2 norm. K-means clustering was used to cluster these features. Here, we used L_2 norm or Euclidean distance for clustering and 60 clusters per image category were obtained. A maximum of 100 iterations were performed for the algorithm to converge.

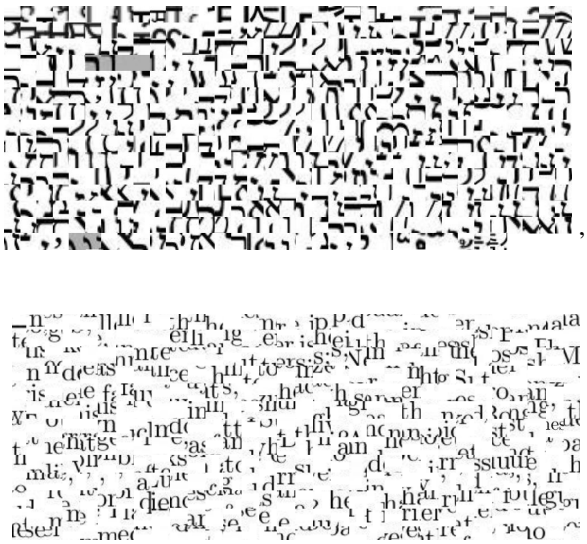

Figure 2: Examples of good visual words

We also wrote a script to visualize the visual vocabulary. The code involved saving the location and radius of all detected blobs in an image and an image identifier along with their SIFT descriptors, so that when the features are clustered together, we can get the image via the image id and also get the square patches(approximation of blobs) in that image for visualization. This was useful to understand if the SIFT features we extracted were being clustered effectively. Looking at Figure 2, we can clearly see that the visual words have a certain type of images clustered together. More specifically, in our case, the vocabulary contained visual words which were largely of the same script. This is intended behavior, and was confirmation for us that the features being chosen were good. Figure 3 shows an example of not so good vocabulary which resulted in less than opti-

mal classification accuracy since the visual words were all mixed up between scripts. This step was very important for our end goal.

Figure 3: Example of bad visual words

4.2. Training the classifier

For training the linear classifier, the regularization parameter λ was set to 0.01, the learning rate η to 0.01 and the maximum iterations done for the algorithm to converge were taken to be 200. These results were obtained after several experiments on validation set.

For K-nearest neighbor classifier, we are using the most intuitive form of classification which suits our needs. We use K=1, which means we will find the single most data point from the training data which is closest to the feature vector of the test image. For our purposes, we will be computing a distance metric between the test image feature vector and all the training set feature vectors. Since the training set is already labeled, we can then find the class label for the feature vector in the training set which is closest to the test image feature vector, and predict it as the class for the test image. For calculating the distance metric between two feature vectors, we chose two different methods. The first one calculates the euclidean distance or the L-2 distance. The second one calculates the Chi-2 distance. We are dealing with histograms of feature vectors, since we want to represent every image as a histogram of visual words. For comparing distances between histograms, the Chi-2 distance metric is often used.

$$\frac{1}{2}\sum_{i=1}^{n}\frac{(x_i^2 - y_i^2)}{x_i + y_i)}$$

For random forest classifier, the treebagger library of MATLAB was used. The number of trees were taken to be 500 for training the model to get best results.

For testing the model, the feature vector representation of images in validation set was obtained using the same method we did for the training set. We used hard classification for this purpose. Each keypoint in an image was assigned a predicted label based on the model and the predicted label for the entire image was decided by a majority voting.

5. Experimental setup

5.1. Dataset

We decided to create our own dataset by collecting online books and articles written in different languages. Firstly, we shortlisted a list of languages which we would choose as our classification classes. These scripts were - Latin (English), Devnagari (Hindi), Chinese, Japanese, Hebrew, Greek, Punjabi, Telugu, Kannada, Thai, Korean. These scripts were chosen mainly due to their distinctive writing styles in terms of strokes and characters. The next step was to collect printed material online containing mostly text in each of these 11 scripts. We sourced books from [Project Gutenberg] in .epub format and articles from [Wikipedia] in PDF format.

Figure 4: Image samples from our final dataset

5.2. Preprocessing

The next step was to convert all files in a consistent format (PDF) and then separate it into pages which were saved as image files (JPEG). Finally we wrote a small script to slide a 150 x 150 px window without overlap across every

image and generate much more smaller patches of every document page. This helps us to look at around 3-4 vertical lines of text in every window with around 1-4 words per line. Furthermore, we can eliminate the boundaries of the images which contain no text in this step. We randomly sampled around 1200 images per script from this collection of 150 x 150 px images. Finally, we manually filtered our dataset to replace any unwanted images like the ones which contain pictures or the ones which contain multiple scripts.

5.3. Training

We chose to split our dataset in a ratio of 8:1:1. Thus, out of 1000 filtered images per script, 800 were chosen at random to be a part of the training set and the remaining 200 images were randomly and evenly split to form the validation and test sets. Once the splits were decided, every image in the dataset was annotated with an id, set and class.

6. Results

In this section, we present the results of our experiments done on the test set. First, we will show a summary comparison of the best results obtained using different features and different classifiers and later on we will discuss how accuracy varies with hyper-parameters for specific methods. We will also analyze the confusion matrix for our best combination.

Accuracy Matrix	Linear	Nearest neighbor	Random forest
SIFT	75.27	94.95	87.80
LBP	70.46	91.23	87.86

Table 1: Best accuracies(%) across different methods.

Comparing feature extraction methods, we found that SIFT and LBP features were both resulted in good feature vector representations, however SIFT resulted in slightly better classification in most cases. The comparison is shown in Table 1. For future reference, if the feature extraction method is not mentioned, it is assumed to be SIFT.

Linear Classifier: The results obtained for the linear classifier are given below. Accuracy is shown in percentages.The linear classifier uses a softmax function for classification.

Random Forest Classifier: The results obtained for this classifier are shown in the table below. The base σ for keypoint detection was taken to be 3.2 with 6 levels scaled by a factor of $\sqrt{2}$ at each step. We have presented a table with hyper-parameters as its rows and columns and accuracy values as its entries.

Accuracy Matrix	τ = 0.03	τ = 0.05	τ = 0.08	τ = 0.10
λ = 0.006	63.82	67.36	67.63	52.32
λ = 0.01	71.93	**75.27**	75.17	68.58
λ = 0.05	58.68	64.29	61.30	60.40

Table 2: Accuracy matrix for Linear classifier on SIFT features: Learning rate η is fixed at 0.01

Accuracy Matrix	τ = 0.03	τ = 0.05	τ = 0.08	τ = 0.10
#trees = 50	81.80	83.80	74.50	70.60
#trees = 200	82.80	**87.80**	78.30	72.40
#trees = 500	79.60	80.20	72.60	70.80

Table 3: Accuracy matrix for random forest using SIFT features: τ = threshold in keypoint detection, #trees are the trees used to create an ensemble.

Nearest Neighbor Classifier: For classification with 1-nearest neighbor and SIFT feature extraction, the base σ for blob detection was kept constant at 6 with 5 levels scaled by a factor of 1.2 at each step. K-means was run with K = 500. Identifying patches or blobs was crucial as accuracy varied based on these hyper-parameters. The threshold(T_m) for a good patch was calculated based on the content of black pixels in the image. The tuning of hyper-parameters is presented in Table 4

Accuracy Matrix	T_m= 0.85	T_m= 0.89	T_m= 0.90
L2 distance	77.95	87.69	82.72
Chi-2 distance	86.24	**94.95**	91.73

Table 4: Accuracy matrix for 1-Nearest neighbor classification parameters.

From the confusion matrix in Figure 5, it is observed that languages like Hindi, English, Greek, Japanese, Telugu, Hebrew and Punjabi are classified with very high accuracy. The language Thai gets confused with Greek and English, Korean gets confused with Chinese and vice versa, and a very small fraction of Kannada gets confused with Telugu. This is because these scripts are very close to each other in character appearance and have some characters which have similar shapes.

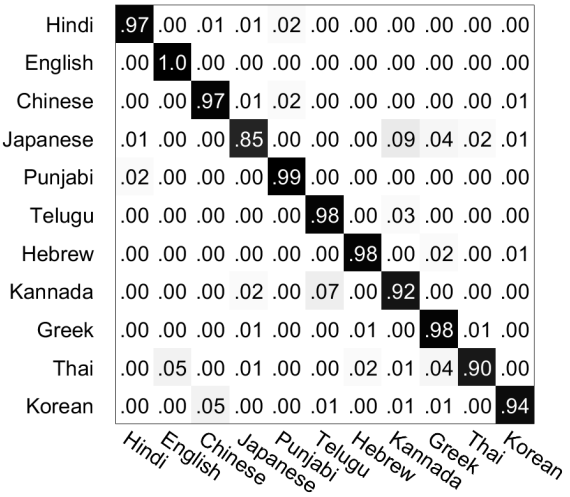

Figure 5: Confusion matrix for Chi-2 distance nearest neighbor classification with K = 500 for K-means

7. Conclusion and Future Work

We presented a method of script identification in print media. The method was based on creating bag of visual words model with localized SIFT and LBP features extracted from certain keypoints in the images and quantization of those features to create a visual vocabulary such that every image could be represented as a histogram of these visual words in the vocabulary. SIFT features performed better than LBP features on our dataset. Classification with Random decision forest was comparable to the nearest neighbor method. Experimented performed on our original dataset with minimal variations in font resulted in very high accuracy as shown in our results. However our model trained on only one font set, when used to classify different font styles and sizes performed poorly. This was as expected and can be fixed by training on a similar dataset which contains multiple fonts for every language. In addition, we also introduced a new dataset containing of more than 11000 text images across 11 different scripts. Our work demonstrates the viability of using SIFT features for script identification in documents and print media.

References

[Gomez et al, 2016] Lluis Gomez, Anguelos Nicolaou, Dimosthenis Karatzas (2016). Boosting patch-based scene text script identification with ensembles of conjoined networks. *arXiv*, 1602.07480v1.

[Sharma et al, 2015] N. Sharma, R. Mandal, R. Sharma, U. Pal, M. Blumenstein (2015). Bag-of-visual words for word-wise video script identification: A study. *International Joint Conference on Neural Networks (IJCNN)*

[Nicolaou et al, 2016] A. Nicolaou, A. D. Bagdanov, L. Gomez-Bigorda, D. Karatzas (2016). Visual script and language recognition. *DAS*

[Shi et al, 2015] B. Shi, C. Yao, C. Zhang, X. Guo, F. Huang, X. Bai (2015). Automatic script identification in the wild. *ICDAR*

[Spitz and Ozaki, 1995] A. L. Spitz, M. Ozaki (1995). Palace: A multilingual document recognition system. *Document Analysis Systems*

[Lee et al, 1996] D. Lee, C. R. Nohl, H. S. Baird (1996). Language identification in complex, unoriented, and degraded document images. *Series in machine perception and artificial intelligence*

[Waked et al, 1998] B. Waked, S. Bergler, C. Suen, S. Khoury (1998). Skew detection, page segmentation, and script classification of printed document images. *IEEE International Conference on Systems, Man, and Cybernetics*

[Tan, 1995] T. Tan (1995). Rotation invariant texture features and their use in automatic script identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*

[Pan et al, 2005] W. Pan, C. Y. Suen, T. D. Bui (2005). Script identification using steerable gabor filters. *Eighth International Conference on Document Analysis and Recognition*

[Wood et al, 1995] S. L. Wood, X. Yao, K. Krishnamurthi, L. Dang (1995). Language identification for printed text independent of segmentation. *International Conference on Image Processing*

[Gllavata et al, 2005] J. Gllavata, B. Freisleben (2005). Script recognition in images with complex backgrounds. *SPIT*

[Kanungo et al, 2002] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu (2002). An efficient k-means clustering algorithm: analysis and implementation *IEEE Transactions on Pattern Analysis and Machine Intelligence*

[Project Gutenberg] Free ebooks by Project Gutenberg *www.gutenberg.org/*

[Wikipedia] Wikipedia - The Free Encyclopedia. *www.wikipedia.com/*

[David G. Lowe] Distinctive Image Features from Scale-Invariant Keypoints *International Journal of Computer Vision*, 2004