Computer Vision
Project Report

Word Spotter

Kirandevraj (2019701001)
Vivek Chandela (20171195)
Ritvik Agarwal (2018122005)

Github Repo:
https://github.com/Kirandevraj/Handwritten-Word-Spotting-with-Corrected-Attributes

Contents

Abstract

The project's focus is to provide an approach to multi-writer word spotting, where the goal would be to find a query word in a dataset comprised of document images. It is an attributes-based approach that leads to a low-dimensional, fixed-length representation of the word images that is fast to compute and, especially, fast to compare. This approach would lead to a unified representation of word images and strings, which seamlessly allow one to indistinctly perform query-by-example, where the query is an image, and query-by- string where the query is a string.

Challenges with previous works

- Out of Vocabulary words(words not there in training images, but exist in the test images)
- Time taken for the image retrieval
- Same word, different handwriting

Objective

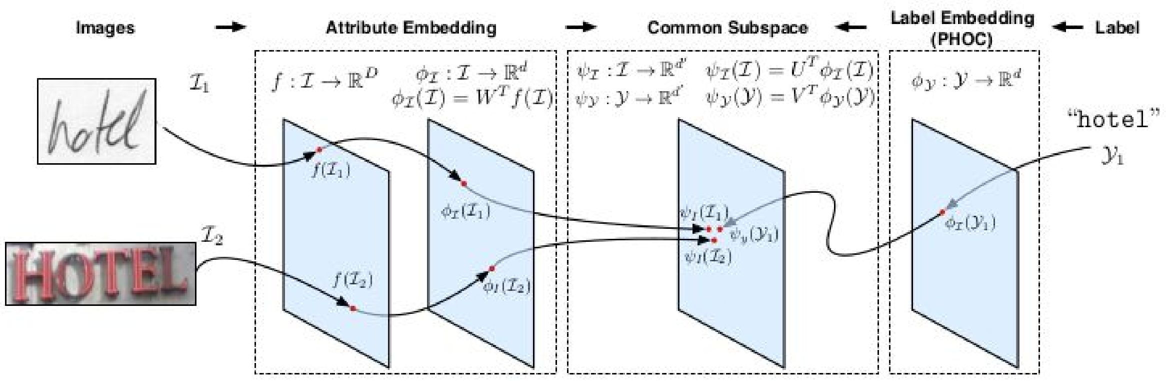
To find all instances of a given word in a potentially large dataset of document images. The various types of Queries to be handled are:

- Query by example (Image)
- Query by string (Text)

Approach

Build a unified classifier to predict the attribute representation(PHOC) from a given Fisher Vector representation computed over SIFT descriptors extracted densely from the word images. The word strings are encoded as PHOCs which are learned using SVM. After that these learned attributes and PHOCs are projected to the common subspace where they are maximally correlated.

Given a QBS/QBE, get the matches with the cosine similarity score among all the document images.



Images
Attribute Embedding
Common Subspace
Label Embedding (PHOC)
Label
$\mathcal{I}_1$
$\mathcal{I}_2$
$f : \mathcal{I} \to \mathbb{R}^D$
$\phi_\mathcal{I} : \mathcal{I} \to \mathbb{R}^d$
$\phi_\mathcal{I}(\mathcal{I}) = W^T f(\mathcal{I})$
$\psi_\mathcal{I} : \mathcal{I} \to \mathbb{R}^{d'}$
$\psi_\mathcal{Y} : \mathcal{Y} \to \mathbb{R}^{d'}$
$\psi_\mathcal{I}(\mathcal{I}) = U^T \phi_\mathcal{I}(\mathcal{I})$
$\psi_\mathcal{Y}(\mathcal{Y}) = V^T \phi_\mathcal{Y}(\mathcal{Y})$
$\phi_\mathcal{Y} : \mathcal{Y} \to \mathbb{R}^d$
"hotel"
$\mathcal{Y}_1$
$f(\mathcal{I}_1)$
$f(\mathcal{I}_2)$
$\phi_\mathcal{I}(\mathcal{I}_1)$
$\phi_I(\mathcal{I}_2)$
$\psi_I(\mathcal{I}_1)$
$\psi_y(\mathcal{Y}_1)$
$\psi_I(\mathcal{I}_2)$
$\phi_\mathcal{I}(\mathcal{Y}_1)$

Fig1: Overview of the implemented method. Images are first projected into an attributes space with the embedding function after being encoded into a base feature representation. Then embedded labels and attributes in a learned common subspace

1. Computing Fisher Vectors

Fisher Vectors can be understood as a bag of words representation that encodes higher order statistics and has been shown to be a good encoding method. First SIFT features are extracted densely over the image. For calculating SIFT features we have used vlfeat's phow implementation, it is Dense SIFT calculated at various levels by Gaussian smoothing of the image.

The resulting SIFT features calculated from phow has 128 dimensions. They are reduced to 62 dimensions with the help of PCA (Principal Component Analysis). The SIFT descriptors of the image are then enriched by appending the normalized x and y coordinates and the scale they were extracted at. Thus a total of 64 dimension vector is calculated for each sift.

Let $I = (x_1, ..., x_N)$ be a set of D dimensional feature vectors extracted from image let $\Theta = (\mu_k, \Sigma_k, \pi_k \ : \ k = 1, ..., K)$ be parameters of K Gaussians fitting the distribution of descriptors. The GMM associates each vector x_i to model k in the mixture model. For each model k, consider the mean and covariance deviation vectors.

$$u_{jk} = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ik} \frac{x_{ji} - \mu_{jk}}{\sigma_{jk}},$$
$$v_{jk} = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ik} \left[\left(\frac{x_{ji} - \mu_{jk}}{\sigma_{jk}} \right)^2 - 1 \right]$$

Where $j = (1, ..., D)$ spans the vector dimensions. The FV of the image I is the stack of the above vectors. $\Phi(I) = [...u_k...v_k...]^T$. To capture the spatial features of the image, we have divided the image into $2x6$ grid. Then the Above procedure is applied individually to each rectangle of the grid.

Algorithm 1 Compute Fisher vector from local descriptors

```
Input:
  • Local image descriptors $X = \{x_t \in \mathbb{R}^D, t = 1, \ldots, T\}$,
  • Gaussian mixture model parameters $\lambda = \{w_k, \mu_k, \sigma_k, k = 1, \ldots, K\}$
Output:
  • normalized Fisher Vector representation $\mathcal{G}_\lambda^X \in \mathbb{R}^{K(2D+1)}$
1. Compute statistics
    • For $k = 1, \ldots, K$ initialize accumulators
        – $S_k^0 \leftarrow 0, \quad S_k^1 \leftarrow 0, \quad S_k^2 \leftarrow 0$
    • For $t = 1, \ldots T$
        – Compute $\gamma_t(k)$
        – For $k = 1, \ldots, K$:
            * $S_k^0 \leftarrow S_k^0 + \gamma_t(k)$,
            * $S_k^1 \leftarrow S_k^1 + \gamma_t(k) x_t$,
            * $S_k^2 \leftarrow S_k^2 + \gamma_t(k) x_t^2$
2. Compute the Fisher vector signature
    • For $k = 1, \ldots, K$:
        $\mathcal{G}_{\alpha_k}^X = \left(S_k^0 - T w_k\right) / \sqrt{w_k}$
        $\mathcal{G}_{\mu_k}^X = \left(S_k^1 - \mu_k S_k^0\right) / \left(\sqrt{w_k}\sigma_k\right)$
        $\mathcal{G}_{\sigma_k}^X = \left(S_k^2 - 2\mu_k S_k^1 + (\mu_k^2 - \sigma_k^2) S_k^0\right) / \left(\sqrt{2 w_k}\sigma_k^2\right)$
    • Concatenate all Fisher vector components into one vector
      $\mathcal{G}_\lambda^X = \left(\mathcal{G}_{\alpha_1}^X, \ldots, \mathcal{G}_{\alpha_K}^X, \mathcal{G}_{\mu_1}^{X\prime}, \ldots, \mathcal{G}_{\mu_K}^{X\prime}, \mathcal{G}_{\sigma_1}^{X\prime}, \ldots, \mathcal{G}_{\sigma_K}^{X\prime}\right)'$
3. Apply normalizations
    • For $i = 1, \ldots, K(2D+1)$ apply power normalization
        – $\left[\mathcal{G}_\lambda^X\right]_i \leftarrow \text{sign}\left(\left[\mathcal{G}_\lambda^X\right]_i\right) \sqrt{\left|\left[\mathcal{G}_\lambda^X\right]_i\right|}$
    • Apply $\ell_2$-normalization:
      $\mathcal{G}_\lambda^X = \mathcal{G}_\lambda^X / \sqrt{\mathcal{G}_\lambda^{X\prime} \mathcal{G}_\lambda^X}$
```

2. Encoding words as PHOC

PHOC, the pyramidal histogram of characters, is binary histogram encodes whether a particular character appears in the represented word or not. By using a spatial pyramid we add some coarse localization, e.g., this character appears on the first half of the word, or this character appears in the last quarter of the word (see Fig. 2). The approach embeds text strings into a d−dimensional binary space. It encodes a particular character appears in a particular spatial region of the string. The histogram can also encode bigrams or other combinations of characters. When computing the attribute representation, we use levels 2, 3, and 4 $(2+3+4)x26 = 234$ dimensions. We also used 75 common bigrams at level 2, leading to an extra 150 dimensions for a total of 384 dimensions.

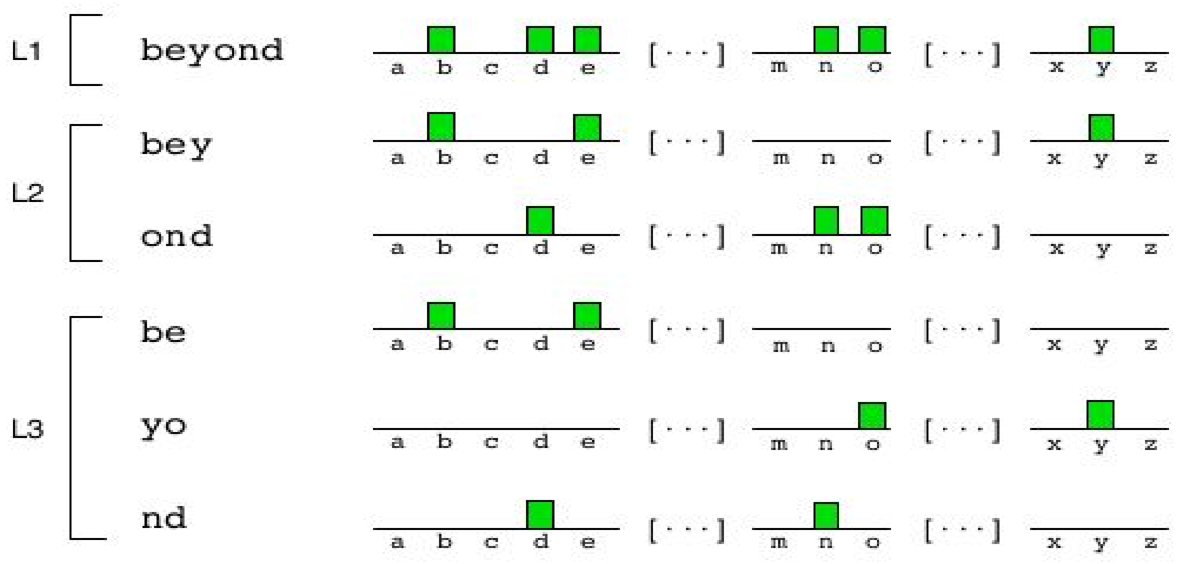


Figure 2: PHOC histogram of a word at levels 1, 2, and 3. The final PHOC histogram is the concatenation of these partial histograms.

3. Learning Attributes

SVM is trained using the Fisher Vector calculated in step 1 as input and PHOCs calculated from transcription in step2 as output. To train the attributes we use a one-versus-rest linear SVM with an SGD solver. We've used sklearn's SGD solver.

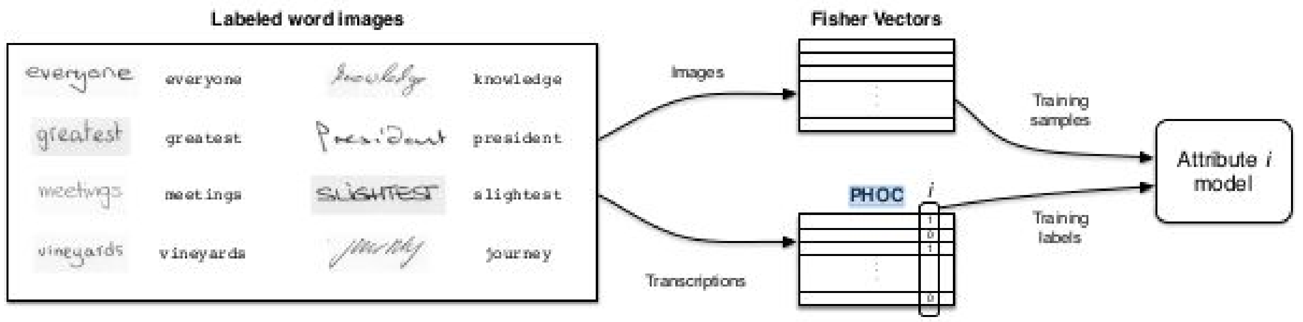


Figure 3: Training process for i-th attribute model. An SVM classifier is trained using the Fisher vector representation of the images and the i-th value of the PHOC representation as labels.

4. Calibration of scores

Although the SVM results are writer independent, special care has to put when comparing different words, since the scores of one attribute may dominate over the scores of other attributes scores is necessary. This is particularly true when we will be performing QBS. We have calibrated scores jointly since this can better exploit the correlation between different attributes. In common subspace where they are maximally correlated.

To achieve this goal we've used Canonical Correlation Analysis to embed the attribute scores and the binary attributes in a common subspace where they are maximally correlated(Fig. 4)

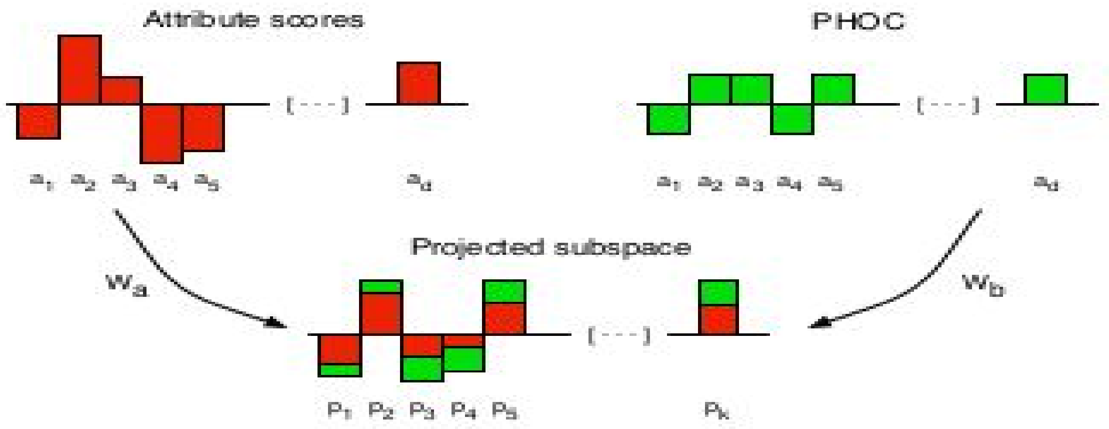


Figure 4. Projection of predicted attribute scores and attributes ground truth into a more correlated subspace with CCA.

Let us assume that we have access to N labeled samples for training purposes, where Let us assume that we have access to N labeled samples for training purposes, where $A \in R^{D \times N}$ is the D dimensional attribute score representation of those samples, and where $B \in \{0, 1\}^{D \times N}$ is their binary attributes representation. Let us denote with μ_a and μ_b the sample means of A and B. Let us also define the matrices $C_{aa} = \frac{1}{N}(A-\mu_a)(A-\mu_a)' + \rho I$, $C_{bb} = \frac{1}{N}(B-\mu_b)(B-\mu_b)' + \rho I$, $C_{ab} = \frac{1}{N}(B-\mu_a)(B-\mu_b)'$ and $C_{ba} = C'_{ba}$, where

ρ is a regularization factor used to avoid numerically ill-conditioned situations an I is the identity matrix.

The goal of CCA is to find a projection of each view that maximizes the correlation between the projected representations. This can be expressed as:

$$argmax_{w_a, w_b} \frac{w'_a C_{ab} w_b}{\sqrt{w'_a C_{aa} w_a}\sqrt{w'_b C_{bb} w_b}}$$

In general, we are interested in obtaining a series of projections $W_a = \{w_{a1}, w_{a2}, \ldots, w_{ak}\}$, with $w_{ai} \in R^D$, subject to those projections being orthogonal. This is solved through a generalized eigenvalue problem, $Zw_{ak} = \lambda^2_k w_{ak}$, with $Z = C_{aa}^{-1} C_{ab} C_{bb} C_{ba}^{-1}$. The k leading eigenvectors of Z form the $W_a = \{w_{a1}, w_{a2}, ..., w_{ak}\}$ projection vectors that project the scores A into the k-dimensional common subspace. Similarly, we can solve for W_b and arrive at an analogous equation to obtain the $W_b = \{w_{b1}, w_{b2}, ..., w_{bk}\}$ projection vectors that project the attributes B into the k-dimensional common subspace. Note that this equation has to be solved only once, offline. Since D is the number of attributes, which is usually small (384 in our case), solving the eigenvalue problem is extremely fast. At testing time, given a sample $x \in R^D$ we can embed it into this space by computing $W_a'(x - \mu_a)$ or $W_b'(x - \mu_b)$, depending on whether x represents attribute scores or pure binary attributes.

Implementation details

We use the Fisher vectors as our base image representation. SIFT features are densely extracted at 6 different patch sizes (bin sizes of 2, 4, 6, 8, 10, and 12 pixels) from the images and reduced to 62 dimensions with PCA. Then, the normalized x and y coordinates are appended to the projected SIFT descriptors. To normalize the coordinates, we use the whole image as a reference system. These features are then aggregated into an FV that considers the gradients with respect to the means and variances of the GMM generative model.

To learn the GMM we use 1 million SIFT features extracted from words from the training sets. We use 16 Gaussians per GMM, and learn the GMM in a structured manner using a 2 × 6 grid leading to a GMM of 192 Gaussians. This produces histograms of 2 × 64 × 192 = 24, 576 dimensions. The descriptors are then power- and L2- normalized. When computing the attribute representation, we use levels 2, 3 and 4 as well as 75 common bigrams at level 2, leading to 384 dimensions considering the 26 characters of the English alphabet

For learning the attributes we've used 39756 (40%) images to train one vs rest SGD classifier. The regularization parameter λ equal to 1e-4 was used. We've trained sklearn's one-versus-rest classifier parallelly to learn 384 attributes from 24,576 features learned per image.

For CCA we've used 41032 images to learn the common subspace. We've reduced the 384 dimensions to 196 dimensions in the process.

We've used 32 core intel skylake virtual machine with 300 GB RAM on google cloud. Also, to increase the training speed we've used python's multiprocessing library. The training takes about 8 hours and testing for 20K images take about 4 hours.

Algorithm

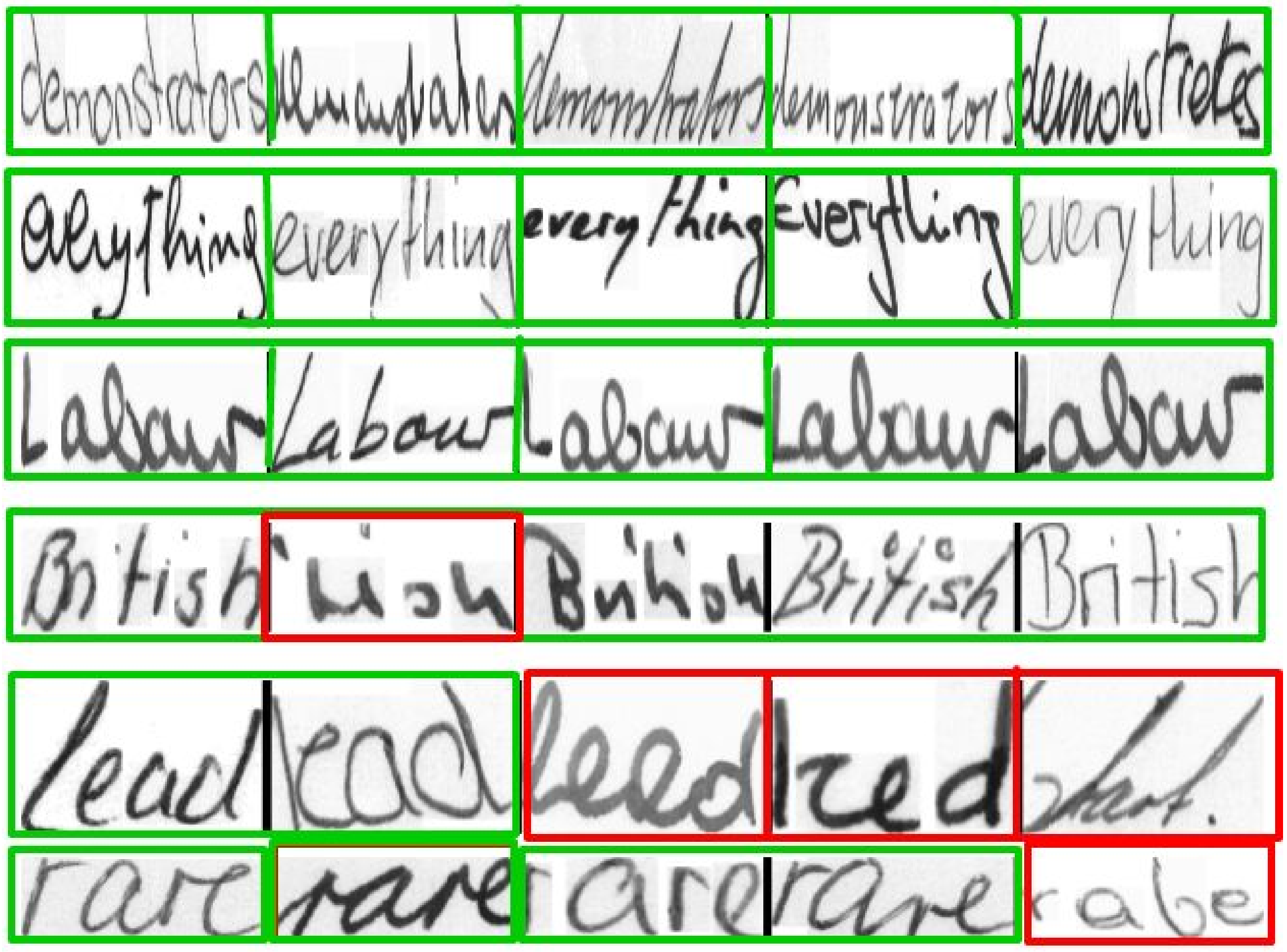
1. Get the SIFT features of the image and find their Fisher vectors
2. Predict/train the PHOC attributes using a classifier(SVM), given the FV.
3. Since we have the image and string for a word, actual PHOC attributes can be found by using the string input.
4. Using CCA, get the projections of the predicted scores and the ground truth values.
5. Use cosine similarity to compute the mean average precision score.

Results

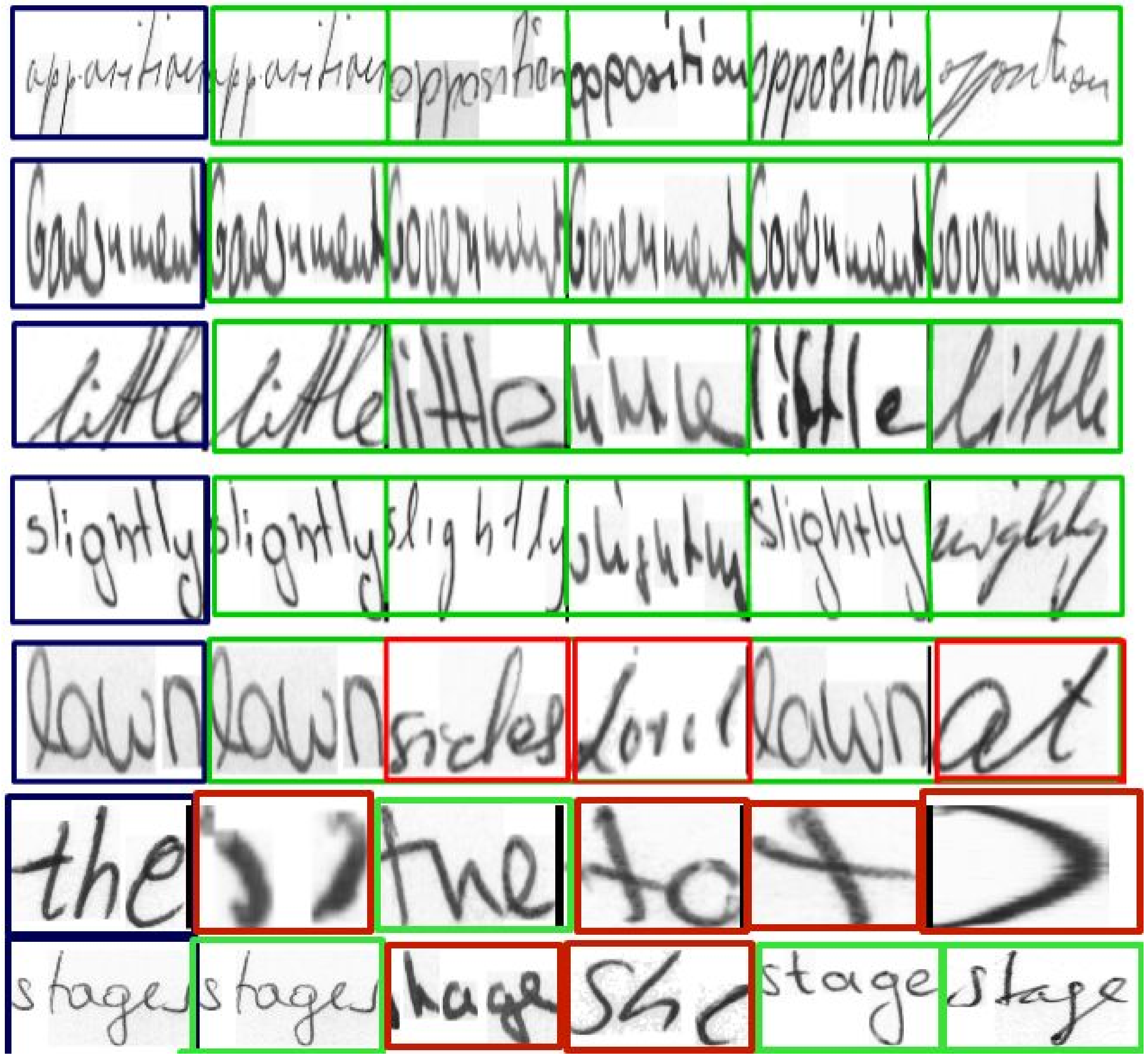
	FV	Attributes	Attributes + CCA
QBS	-	0.42	0.48
QBE	0.11	0.28	0.37

Table 1: Retrieval results on the IAM dataset. Accuracy measured in mean average precision.

Query by String(QBS)



Query by Example(QBE)



Observations

1) The MAP score with just the FV as attribute representation is used as a benchmark score for other embedding techniques.
2) The attribute based representation gives a better result than the benchmark as the SVM learns the handwriting differences and what factors that make a word unique.
3) Initially when λ = 1e-3 the SVM was overfitting hence was giving poor results. Later for λ = 1e-5, the model was performing better.
4) Also there are some mismatches in the resulting images above, this is happening as the SVM might be overfitting and not have seen similar images in training.

Challenges

1) Lack of resources related to the implementation details of the original research paper.
2) High Computational power required as the dataset is huge (~1M images). Hence, rented google cloud's virtual machine.
3) Since we're using Dense SIFT many features were coming out to be zero which was resulting in erroneous results. Excluded such features.
4) Initially used sklearn's GMM but it had convergence issues. So used vlfeat's implementation of GMM.
5) Used vlfeat's cython implementation of FV as our own implementation in python was slow.

References

1) http://www.cvc.uab.es/~almazan/wp-content/uploads/2013/10/almazan_ICCV13.pdf
2) http://www.cvc.uab.es/~afornes/publi/journals/2014_PAMI_Almazan.pdf
3) http://www.vlfeat.org/
4) https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
5) https://scikit-learn.org/stable/modules/multiclass.htm