Writer Identification using GMM Supervectors

Vincent Christlein David Bernecker Florian H¨onig Elli Angelopoulou

- Report By:
  - Nikhil Bansal (20161065)
  - Mudit Surana (20161100)
  - Rohit Kumar Agarwal (20161011)

- Github Repo link : https://github.com/surana-mudit/Writer-Identification-using-GMM-and-CNN

Abstract

This paper present a new offline model for writer identification and writer verification. The methods is inspired from GMM-UBM supervector method used in the field of speaker verification.
The proposed method uses GMM supervectors to encode the feature(computed using RootSIFT descriptors) distribution of individual writers. Each supervector is computed using an individual GMM which has been adapted from a background model via a MAP step followed by mixing the new statistics with the background GMM model.
The paper compares the accuracy of identification using GMM supervector with identification using Fisher and VLAD encodings.

Dataset

The dataset was taken from the one used in **ICDAR-2013 Competition on Writer Identification**. Both the Experimental and Benchmarking Datasets can be found here.

Methodology

Gaussian Mixture Model (GMM)

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.
The parameters of GMM are estimated using the EM algorithm:

- Let, λ, denotes the set of parameters of mixture of gaussians
  - $\lambda = \{\omega_i, \mu_i, \Sigma_i, | i = 1,.....,.....N\}$
  - Where
    - $\sum_{i=1}^{N} \omega_i = 1$
    - N = Number of mixture
- The likelihood function for EM can be given by:

- $P(x \mid \lambda) = \sum_{i=1}^{N} \omega_i g_i(x ; \mu_i, \Sigma_i)$
- Where
  - $g_i(x) := g_i(x ; \mu_i, \Sigma_i)$ are weighted Gaussians
  - Σ_i is assumed to be diagonal

We used sklearn.mixture package which enables one to learn Gaussian Mixture Models with flag 'diagonal'. GaussianMixture.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it most probably belong to using the GaussianMixture.predict method:

```
from sklearn import mixture

def compute_gmm_params(features):
      'features is a matrix of features(row-wise) to train gmm model'

      ## using default number of clusters i.e N = 100 as is used in paper
      gmm = mixture.GaussianMixture(n_components=100,
covariance_type='diag')
      gmm.fit(features)

      return gmm
```

Universal Background Model (UBM)

The UBM is the background model that is used generally in speaker recognition which is created by estimating a GMM from large spectral feature vectors. In this paper, the UBM is also created by estimating a GMM but here the model is estimated using a set of SIFT descriptors computed from training documents.

The paper employed **RootSIFT features** i.e a variant of SIFT where the features are normalized using the **square root (Hellinger) kernel** as follows:

```
        ## Hellinger normalization
        descriptors += np.finfo(np.float32).eps
        descriptors /= np.sum(descriptors, axis=1)[:, np.newaxis]
        descriptors = np.sqrt(descriptors)
```

GMM Adaptation and Mixing

The final UBM is adapted to each document individually, using all M SIFT descriptors computed at document W:
$X_W = \{x_1, \ldots\ldots, x_M\}$
For the MAP estimation, first the posteriors probabilities are computed for all mixtures i and each feature vector x_j, $j \in \{1 \ldots\ldots M\}$.

$$\pi_j(i) = p(i \mid \mathbf{x}_j) = \frac{w_i g_i(\mathbf{x}_j)}{\sum_{k=1}^{N} w_k g_k(\mathbf{x}_j)}$$

```
def ubm_adaptation(path, outDir, gmm):
      'path denotes the path to the image files'
      fEx = FeatureEx() # creating Feature extraction object

      directory = path

      # features is a matrix of features(row-wise)

      for file in os.listdir(directory):
      print (file)
      if file.endswith(".tif"):
            print(os.path.join(directory, file))
            fp = os.path.join(directory, file)
            print(fp)
            kpts, data = fEx.compute(fp)

            posteriors = gmm.predict_proba(data)
```

```
            enc = supervector(gmm, data, posteriors)
            print(enc)

            #### save the encoding in a folder outDir as pickle file
            fileName, file_ext = os.path.splitext(file)

            if not os.path.exists(outDir):
                  os.makedirs(outDir)

            filepath = os.path.join(outDir, fileName + '.pkl.gz')
            print(filepath)
            with gzip.open(filepath, 'wb') as f:
                  print(f)
                  pickle.dump(enc, f, -1)

      else:
            continue
```

Next, the mixture parameters are adapted. Mixture with high posteriors are adapted more strongly. This is controlled by a fixed relevance factor r^{T} for the adaptation coefficients.

```
def adaptMAP(data, gmm, posteriors, relevance = 16, update='wmc'):

      sum_post = np.sum(posteriors, axis=0) # (N_component x ,)

      nd = len(gmm.weights_) # number of components / gaussians
      fd = data.shape[1] # feature dimension

      data_square = data * data

      def loop(i):
      means_ = posteriors[:,i].reshape(1,-1).dot(data)
      covs_ = posteriors[:,i].reshape(1,-1).dot(data_square)
      return means_, covs_
```

```
means, covs = zip( *map(loop, range(nd)))

means = np.array(means).reshape(nd, fd)
covs = np.array(covs).reshape(nd,fd)

# add some small number
means += np.finfo(float).eps
covs += np.finfo(float).eps

# normalize them
means /= sum_post.reshape(-1,1) + np.finfo(float).eps
covs /= sum_post.reshape(-1,1) + np.finfo(float).eps

# now combine the two estimates using the relevance factor
# i.e. interpolation controlled by relevance factor
def combine(i):
alpha = sum_post[i] / (sum_post[i] + relevance)

# update priors
if 'w' in update:
     weights_ = ( (alpha * sum_post[i]) / float(len(data)) ) \
          + ( (1.0 - alpha) * gmm.weights_[i] )
else:
     weights_ = copy.deepcopy(gmm.weights_[i])

# update means
if 'm' in update:
     means_ = alpha * means[i] \
          + ( (1.0 -alpha) * gmm.means_[i] )
else:
     means_ = copy.deepcopy(gmm.means_[i])
# update covariance matrix
if 'c' in update:
     covs_ = alpha * covs[i] \
          + (1.0 - alpha) * (gmm.covariances_[i] + \
                                  gmm.means_[i] * gmm.means_[i])\
          - (means_ * means_)
```

```
    else:
          covs_ = copy.deepcopy(gmm.covariances_[i])

    return weights_, means_, covs_

    weights, means, covs = zip( *map(combine, range(nd)) )

    weights = np.array(weights)
    means = np.array(means)
    covs = np.array(covs)

    # let weights sum to 1
    if 'w' in update:
    weights /= weights.sum() + np.finfo(float).eps

    # create new mixture
    adapted_gmm = mixture.GaussianMixture(nd)
    # and assign mean, cov, priors to it
    adapted_gmm.weights_ = weights
    adapted_gmm.means_ = means
    adapted_gmm.covariances_ = covs

    return adapted_gmm
```

Where γ is the scaling factor ensuring the weights of all mixtures sum up to one.

For the task of writer identification the parameters of the GMM adapted to one document are stacked into the supervector.

$$\mathbf{s} = (\hat{w}_1, \ldots, \hat{w}_N, \hat{\boldsymbol{\mu}}_1^T, \ldots, \hat{\boldsymbol{\mu}}_N^T, \hat{\boldsymbol{\sigma}}_1^T, \ldots, \hat{\boldsymbol{\sigma}}_N^T)^T$$

Fisher Encoding

The Fisher encoding uses GMM to construct a visual word dictionary.

```
def fisher(data, means, weights, posteriors, inv_sqrt_cov):

      components, fd = means.shape

      def encode(i):
      if weights[i] < 1e-6:
            return np.zeros( (fd), dtype=means.dtype),\
                  np.zeros( (fd), dtype=means.dtype)

      #diff = data * inv_sqrt_cov[i]
      diff = (data - means[i]) * inv_sqrt_cov[i]
      weights_ = np.sum(posteriors[:,i] - weights[i])
      means_ = posteriors[:,i].T.dot( diff )
      covs_ = posteriors[:,i].T.dot( diff*diff - 1 )

      weights_ /= ( len(data) * math.sqrt(weights[i]) )
      means_ /= ( len(data) * math.sqrt(weights[i]) )
      covs_ /= ( len(data) * math.sqrt(2.0*weights[i]) )
      # print weights_, means_, covs_
      return weights_, means_, covs_

      wk_, uk_, vk_ = zip( *map(encode, range(components)) )

      return wk_, uk_, vk_
```

VLAD Encoding

The Vectors of Locally Aggregated Descriptors is similar to Fisher vectors but it does not store second-order information about the features.

```
def vlad(data, means, assignments, components, normalize=['l2c']):
      def encode(k):
      uk_ = assignments[:,k].T.dot(data)

      clustermass = assignments[:,k].sum()
      if clustermass > 0:
            uk_ -= clustermass * means[k]

      if 'l2c' in normalize:
            n = max(math.sqrt(np.sum(uk_ * uk_)), 1e-12)
            uk_ /= n

      return uk_

      uk = list(map(encode, range(components)))
      fin_enc = np.concatenate(uk, axis=0).reshape(1,-1)
      fin_enc = np.sign(fin_enc) * np.sqrt(np.abs(fin_enc))
      fin_enc = preprocessing.normalize(fin_enc)
      return fin_enc
```

Writer Identification

For the identification of the authorship, each distance from the query supervector to all other supervectors of the database is computed.The resulting list of distances is then sorted. Either the list can be further analyzed, e. g. inspecting the first 10 documents, or the author belonging to the smallest distance is assigned to the query document.

Results

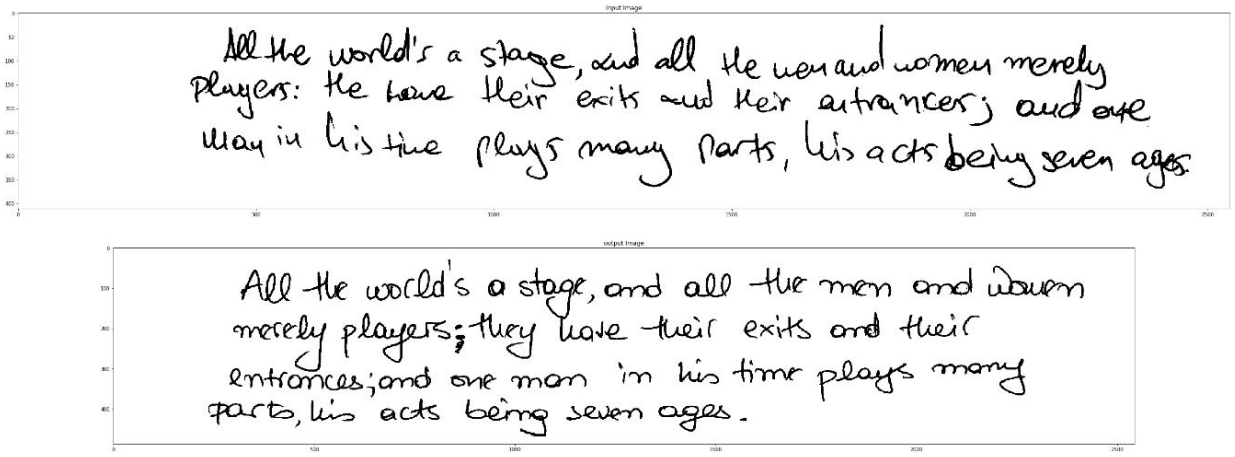
For the evaluation the publicly available **ICDAR13 dataset** is used. It consists of an experimental dataset and a benchmark dataset. The experimental dataset consists of 100 and the benchmark set of 250 writers with four documents per writer. Two

documents were written in Greek, other two in English. The documents of the dataset are in binary image format.

Following table shows the TOP-1 accuracy using the 3 encodings using ICDAR 13 dataset:

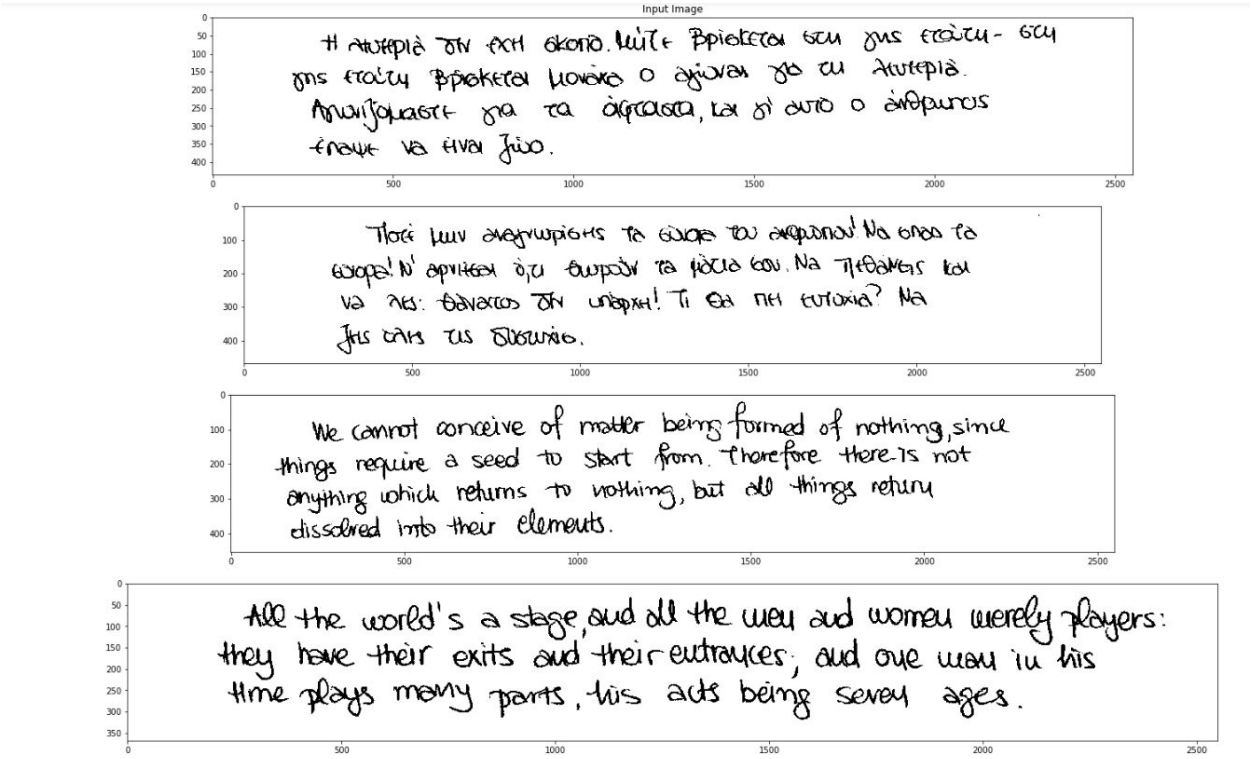
Encoding	TOP-1 Accuracy
GMM Supervector	0.9725
Fisher	0.955
VLAD	0.915

The below document shows one of the incorrect writer identification by the GMM Supervector model:

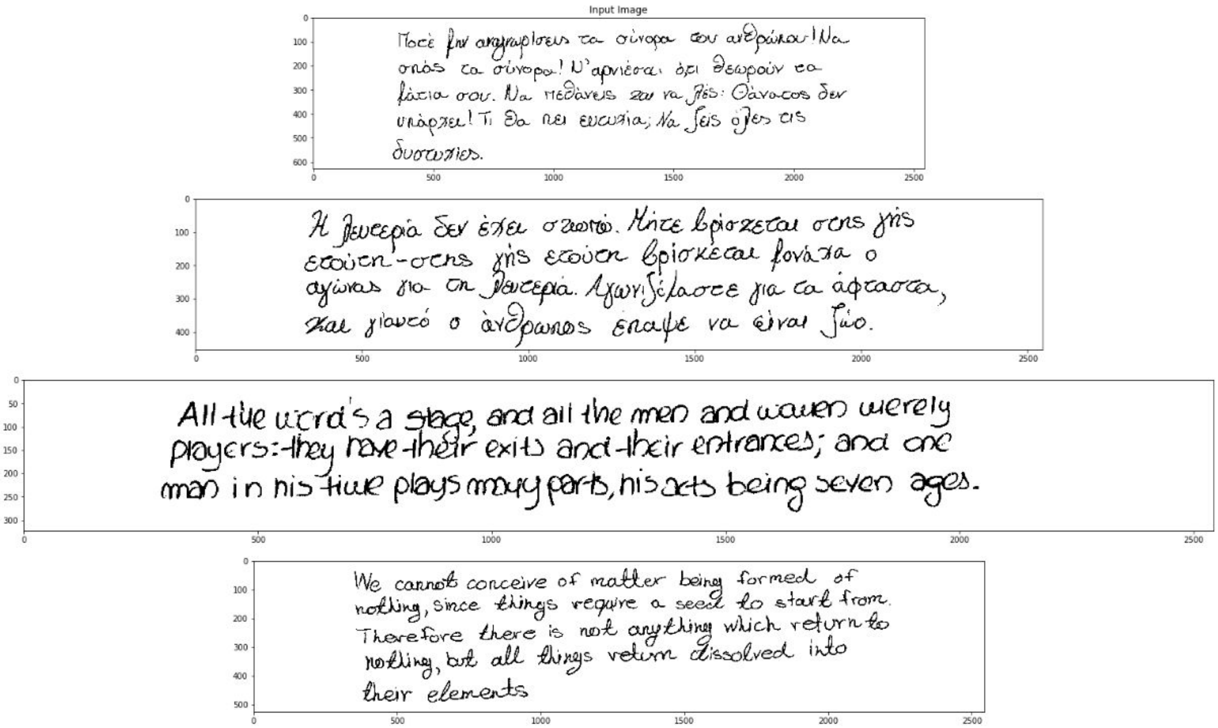


	TOP-1	TOP-2	TOP-3	mAP
VLAD	0.925	0.68	0.4025	0.7125
Fisher	0.94	0.72	0.475	0.748
Supervector	0.9725	0.7575	0.41	0.7525

The below image shows the documents from ICDAR 13 dataset, the first one is the input image and the other three are the top-3 outputs(as per the cos-distance between documents), all of them have correct label.



The below image shows the documents from ICDAR 13 dataset, the first one is the input image and the other three are the top-3 outputs(as per the cos-distance between documents), the second one has the incorrect label:


Input Image
Ποτέ μην αναγνωρίσεις τα σύνορα του ανθρώπου! Να
σπάς τα σύνορα! Ν'αρνιέσαι ό,τι θεωρούν τα
μάτια σου. Να πεθαίνεις και να λες: Θάνατος δεν
υπάρχει! Τι θα πει ευτυχία; Να ζεις όλες τις
δυστυχίες.
Η λευτεριά δεν έχει σκοπό. Μήτε βρίσκεται στης γης
ετούτη - στης γης ετούτη βρίσκεται μονάχα ο
αγώνας για τη λευτεριά. Αγωνίζεται για τα άφταστα,
και γι' αυτό ο άνθρωπος έπαψε να είναι ζώο.
All the word's a stage, and all the men and women merely
players: they have their exits and their entrances; and one
man in his time plays many parts, his acts being seven ages.
We cannot conceive of matter being formed of
nothing, since things require a seed to start from.
Therefore there is not anything which return to
nothing, but all things return dissolved into
their elements

Writer Identification using Deep CNN

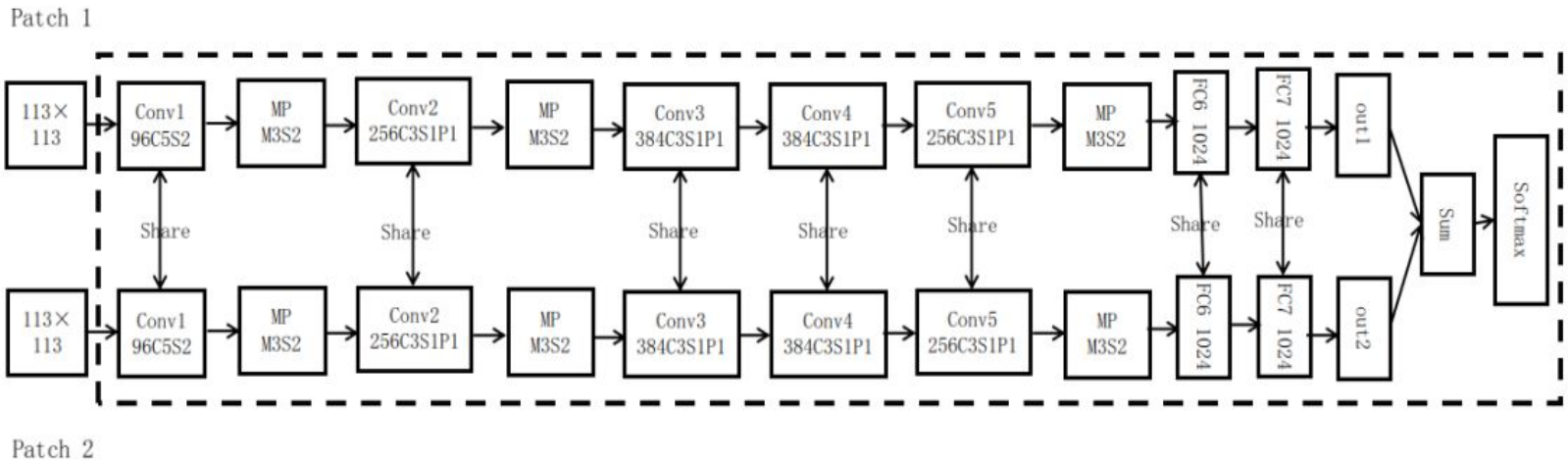
This method uses a deep multi-stream CNN to learn deep powerful representation for recognizing writers. The network takes local handwritten patches as input and is trained with softmax classification loss.

The 3 main contributions are :

1. We design and optimize multi-stream structure for writer identification task
2. We use data augmentation learning to enhance the performance of the Deep CNN
3. We use a patch scanning strategy to handle text image with different lengths.

Here, we have used the **IAM dataset** with handwritings of 50 most common writers for evaluation. The dataset is available here.

Network Architecture



Layer (type)	Output Shape	Param #
zero_padding2d_2 (ZeroPaddin	(None, 115, 115, 1)	0
lambda_2 (Lambda)	(None, 56, 56, 1)	0
conv1 (Conv2D)	(None, 28, 28, 32)	832
activation_7 (Activation)	(None, 28, 28, 32)	0
pool1 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2 (Conv2D)	(None, 14, 14, 64)	18496
activation_8 (Activation)	(None, 14, 14, 64)	0
pool2 (MaxPooling2D)	(None, 7, 7, 64)	0
conv3 (Conv2D)	(None, 7, 7, 128)	73856
activation_9 (Activation)	(None, 7, 7, 128)	0
pool3 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_2 (Flatten)	(None, 1152)	0
dropout_4 (Dropout)	(None, 1152)	0
dense1 (Dense)	(None, 512)	590336
activation_10 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense2 (Dense)	(None, 256)	131328
activation_11 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
output (Dense)	(None, 50)	12850
activation_12 (Activation)	(None, 50)	0

==========
Total params: 827,698
Trainable params: 827,698
Non-trainable params: 0
_____

Input

- The input to the model are not unique sentences but rather random patches cropped from each sentence.
- Resize each sentence so that new height is 113 pixels and new width is such that original aspect ratio is maintained, since, distorting the shape of image by changing the aspect ratio resulted in a big drop in model performance.
- From the adjusted image, patches of 113x113 are randomly cropped, which are then given as input to the model.

Testing

- Scan the testing image to generate image patches
- Compute the final score for each writer by averaging scores of all image patches for that writer
- Return the writer with the highest score

Results

Following table shows the accuracy comparison between the CNN results and the 3 encodings.

CNN	0.988
GMM Supervector	0.9725
Fisher	0.94
VLAD	0.925