Siamese Triple Ranking Convolution Network in Signature Forgery Detection

Solution Design

Signature forgery can be broadly of two types:

1. Blind Forgery: Where the forger has no idea what the signature to be forged looks like. This is easy to detect by machine because it is usually not very close to the appearance of a genuine signature.
2. Skilled Forgery: Either simulation or tracing, in which the forger has a sample of the signature to be forged. In this case, detecting fraud requires more sophisticated tools to differentiate minute but critical details between genuine and forged signatures.

We have developed a framework to detect fraud signatures using Image processing and Deep convolutional Siamese networks wherein a *deep triplet ranking network* is used to calculate the image embeddings. This is coupled with generalized linear model architecture with logistic loss functions and cross validation to arrive at the final model to label images as *authentic* or *forged*.

The best part about this framework is once the model is trained, we require just one base image to determine whether another signature image is genuine or not with one shot learning.

Tech Stack

- Python
- Keras
- Tensorflow
- OpenCV

Data Used

- ICDAR 2009 Signature Verification Competition (SigComp2009)
- ICDAR 2011 Signature Verification Competition (SigComp2011): Dutch Offline Dataset

Accuracy Attained

The model has been validated multiple number of times and the results have been attached.

- The training accuracy achieved through the architecture is ***98.5%***, which indicates it can always capture and detect whenever there is any kind of forgery.
- The validation accuracy has been ***98.95%*** on test signatures which indicates the model is highly generalizable as well.

The same has been tabulated in the below table.

Table1: Test Accuracy Measures

Metrics	Values
Test Accuracy	96.5%
Test Recall	96.7%
Test Precision	95.2%

The output results have also been depicted below in Fig.1 and Fig.2.

Genuine as 0 and Forged as 1

```
In [372]: pos_Y = [0]*len(pos_feat_total)
          #makes that no of ones as the len of the item given
          neg_Y = [1]*len(neg_feat_total)
          #rowwise appends the two arrays
          train_y=np.append(pos_Y, neg_Y)

In [373]: #training on 80% data and test on remaining 20% data
          from sklearn.cross_validation import train_test_split
          x_train, x_test, y_train, y_test = train_test_split(feat_total, train_y, test_size=0.2, random_state=42)

In [374]: from sklearn.linear_model import LogisticRegression
          log_model = LogisticRegression(random_state=0)
          log_model.fit(x_train,y_train)

Out[374]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

In [375]: # train accuracy
          log_model.score(x_train,y_train)

Out[375]: 0.9849483006849614

In [376]: # test accuracy
          log_model.score(x_test,y_test)

Out[376]: 0.9654510556621881

In [377]: # precision recall on test data
          print (recall_score(y_test, log_model.predict(x_test)))
          print (precision_score(y_test, log_model.predict(x_test)))

          0.967558799675588
          0.9528753993610224
```

Figure 1: Model accuracy on Test Signature data to detect forgery

As can be shown from the figure and table that the model has high accuracy, adequacy, high generalisability and is significant as well.

To further understand the model performance, we have done a 10-fold Cross validation multiple times to check the Validation accuracy which gives an average cross validation accuracy of above **96%**.

```
In [380]: y_pred=log_model.predict(x_test)
          confusion_matrix(y_test, y_pred)

Out[380]: array([[3147,  118],
                 [  80, 2386]])

In [381]: cross_val_score(log_model, x_train, y_train, cv=10)

Out[381]: array([0.96685565, 0.95377235, 0.95551679, 0.95769734, 0.96249455,
                 0.96247818, 0.95853339, 0.96377128, 0.95635094, 0.95591445])

In [382]: cross_val_score(log_model, x_train, y_train, cv=10,scoring='precision')

Out[382]: array([0.95837463, 0.94350842, 0.94373149, 0.94752475, 0.95073892,
                 0.957     , 0.94669299, 0.95083579, 0.94204322, 0.93429952])

In [383]: cross_val_score(log_model, x_train, y_train, cv=10,scoring='recall')

Out[383]: array([0.96603397, 0.95104895, 0.95504496, 0.95604396, 0.96403596,
                 0.957     , 0.959     , 0.967     , 0.959     , 0.967     ])
```

Figure 2: 10-fold Cross validation accuracy of the Model

Execution Code

All the code and environment for execution is in the github link attached. Please follow the instructions present in the README.md document in github.

Detailed Algorithm: Siamese Triplet Convolution Network

Step 1: Dataset Preparation & Preprocessing

In the first step, we contrive the dataset in a triplet formation. The data set used is from SigComp2009 data (both training and test), as well as from SigComp2011 data (Dutch Offline data).

We create all combinations of *triplets* from this data: this is done by taking an *anchor image* (genuine signature of a person) and placing it in conjunction with both a *positive sample* (another genuine signature of the same person) and a *negative sample* (a forged signature by someone else of the same person). A total of *119800* such triplet combinations is obtained. We resize all the images and convert them to arrays to be passed in the CNN triplet model.

Step 2: CNN Network Architecture & Embeddings

Once we construct the training data in a triplet formation, we train the model based on convolutional Siamese network. Siamese convolution networks are twin networks with shared weights, which can be trained to learn the feature embeddings where similar observations are placed in proximity and dissimilar are placed apart.

In implementing the CNN architecture, we have modified the pre-trained *Mobile net CNN model* (open source- Keras Inbuilt function) with additional layers and have used *Transfer Learning* in building the same. We have trained the last few layers and built dense layers on top of it to extract the embeddings with triplet loss function.

This Transfer Learning based architecture was chosen due to the computational complexity of Deep CNN models. However, training some of the carefully chosen layers fulfills the objective of getting optimal weights for signature related images.

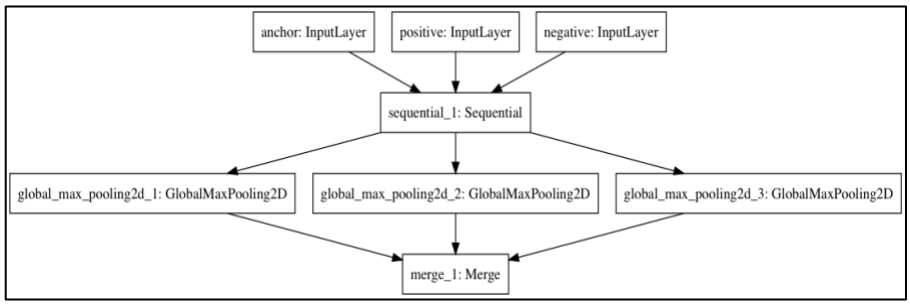

Figure 3: Triplet CNN Architecture Diagram

Layer (type)	Output Shape	Param #	Connected to
anchor (InputLayer)	(None, 128, 128, 3)	0	
positive (InputLayer)	(None, 128, 128, 3)	0	
negative (InputLayer)	(None, 128, 128, 3)	0	
sequential_1 (Sequential)	multiple	3228864	anchor[0][0] positive[0][0] negative[0][0]
global_max_pooling2d_1 (GlobalM	(None, 1024)	0	sequential_1[1][0]
global_max_pooling2d_2 (GlobalM	(None, 1024)	0	sequential_1[2][0]
global_max_pooling2d_3 (GlobalM	(None, 1024)	0	sequential_1[3][0]
merge_1 (Merge)	(None, 1)	0	global_max_pooling2d_1[0][0] global_max_pooling2d_2[0][0] global_max_pooling2d_3[0][0]

Total params: 3,228,864
Trainable params: 1,052,672
Non-trainable params: 2,176,192

Figure 4: Layer Configuration and Weights of Triplet CNN Architecture Diagram

We use the *Adam Optimizer* with *mean absolute error* for back propagation to get to the final encodings. The image embeddings so obtained are such that the dissimilarity between the anchor image and positive image must be low and the dissimilarity between the anchor image and the negative image

must be high for every triplet. This kind of architecture ensures that even small differences in signatures can be captured in order to flag a skilled forgery. The loss function used is shown below.

$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]$$

where,
f(a) refers to the image encoding of the anchor a
f(p) refers to the image encoding of the positive p
f(n) refers to the image encoding of the negative n
alpha is a constant used to make sure that the network does not try to optimize towards f(a) - f(p) = f(a) - f(n) = 0

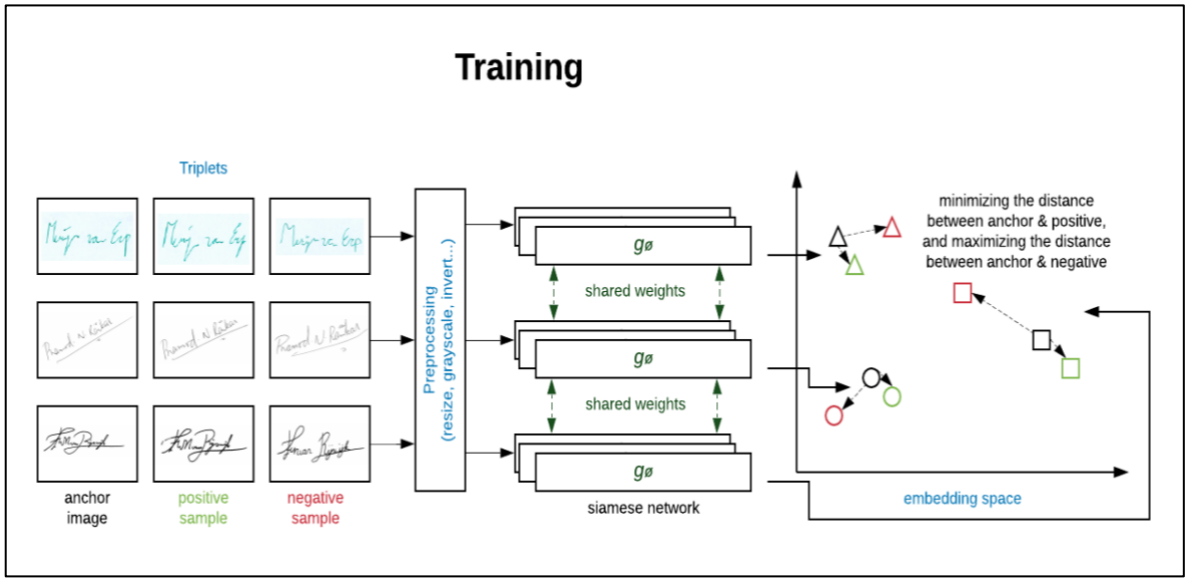

Fig.5: Deep Triplet Ranking CNN Network

Step 3: Logistic Regression Model & Cross Validation
Once we attain the final image embeddings for all the training images through the triplet loss architecture, we train a *generalized linear model* with *logistic loss function* to get the final model that declares any signature as genuine or forged against a base image signature. For training this logistic function, we arrange the images in a *pairwise* manner where each observation is *pair* of images, either both of a person's genuine signature, or one of person's genuine signature, and the other as person's forged signature. These will have labels (class) genuine or fraud assigned to them respectively. A total of about *30,000* such pairs are obtained.

The logistic model is then trained on the subset of dataset pairs to determine that given a base signature image, whether the other image matches the base (genuine) or not (forged). We use cross validation to get to the final logistic model taking the corresponding differences between the embeddings of each of the pairs (*1024 length difference vector of embeddings*) as the feature set and the class labels (genuine/ fraud) as the dependent variable *y*.

Loss: g(sigmoid(Σ(w* X(x_a, x_b) +b)),y)
where,
g() is the Logistic Loss Function
y is the response class (0/1)
x_a, x_b are the image embeddings of corresponding images in the image pair each of dimension m.
X(x_a, x_b) represents the entire set of difference based features computed from the embeddings in space R^m

w and b represent the final weights obtained from generalized linear logistic loss function where the response will be the images are same (0) or forged (1)(y).

This ends the one-time training process for signature verification. We save the image encodings obtained in Step 2 and the weights of trained logistic model obtained in Step 3 as the final model for the real time verification process.

Step 4: Final Framework Architecture

Once the training process is completed, all we need is the trained model outputs (encodings and logistic model weights). Next, we created a database framework to save the original signatures of every new customer that the bank acquires against a unique ID. We pass these *base* images through the encodings we obtained in the training process and get the corresponding image embeddings. This is in the format of a vector of length 1024. We precompute these embeddings and save them against the individuals' unique ID in the database.

Now, whenever we acquire a new signature image against one of the unique ID's that needs to be accessed to determine whether it is genuine or fraud, the framework passes that image through the encoding once again to get its image embeddings.

This new image embedding is then compared to its corresponding embedding of the *base* image of that individual to determine whether it is genuine or forged. This is done by taking the difference vector of the two embeddings and passing it through the logistic model to get the final prediction. If the resultant probability from the logistic model is low, then the framework declares the new image as a genuine, otherwise it is considered a forgery.

The framework flowchart is depicted below in Fig.6 and the test workflow is depicted in Fig7.

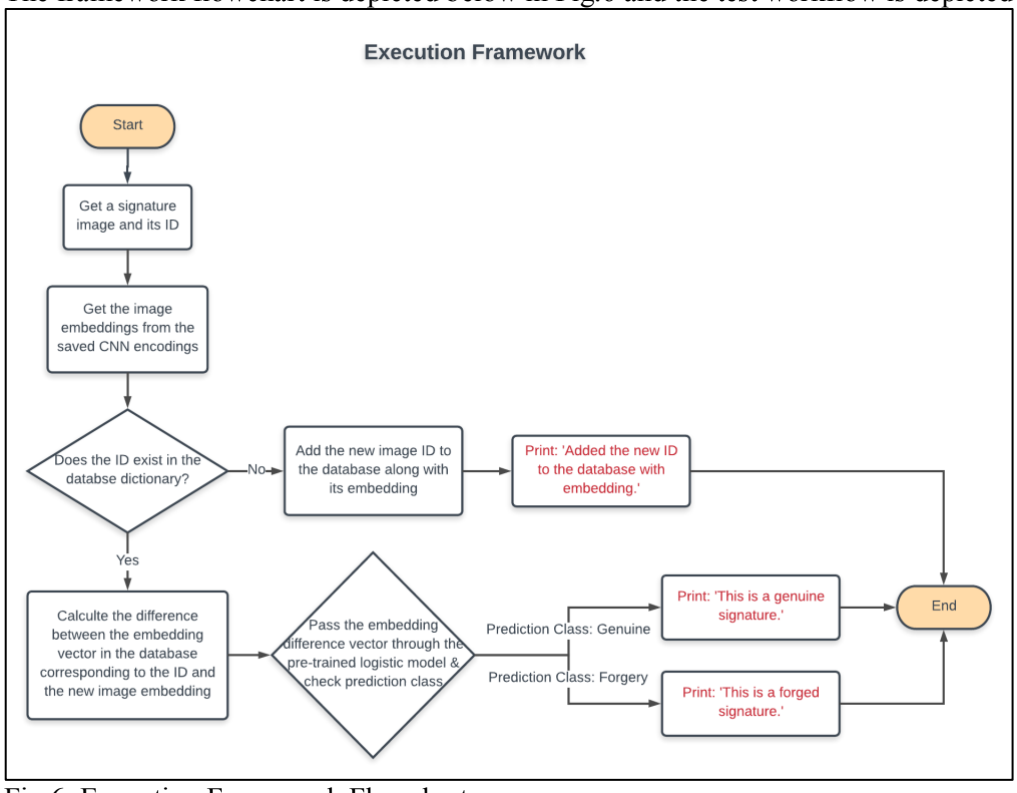

Fig.6: Execution Framework Flowchart

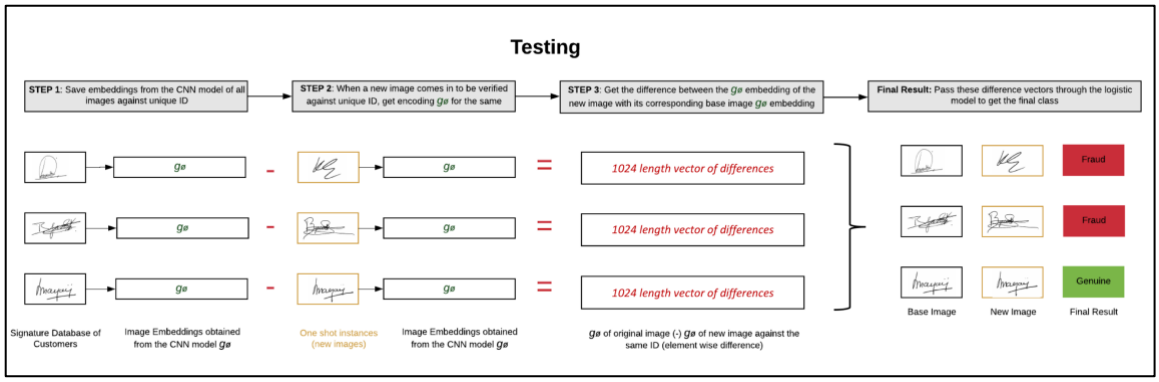

Fig.7: Testing process of determining whether signature is genuine or fraud

Superiority of Triplet Loss over other Architectures

The differences between the images of a genuine signature and its skilled forgery is at times very minute and is challenging to detect by even a trained eye. Deep Triplet loss function used is a very powerful loss function used in the industry for face recognition (facenet). We have created our custom Triplet model architecture with modified Mobile Net CNN and dense layers with Triplet loss function.
Based on this loss, the image embeddings are created in such a way that the dissimilarity between the anchor image and positive image must be low and the dissimilarity between the anchor image and the negative image must be high for every triplet. This kind of architecture ensures that even small differences in signatures can be captured in order to flag a skilled forgery effectively.

Deviations made from the proposal in Stage 1

1. Hierarchical Framework: In the first stage of the competition we proposed a hierarchical framework to first detect blind forgeries and then subsequently detect skilled forgeries through two separately trained CNN models. We abandoned this approach in favor of a single CNN model that detects both skilled and blind forgeries in one-shot because the single CNN we trained along with the logistic model gave optimistic accuracies of >98% on unseen test data of both skilled and blind forgeries.

2. Bayesian optimization: We also proposed a Bayesian optimization and Gaussian Process in the first stage which would choose the model hyperparameters using a Bayesian Process. This is a novel approach that can save a lot on the time complexity of the grid search with selecting the best hyperparameter at each iteration. We did not implement this in the code solely because of time restrictions since the *threshold alpha* we chose in the triplet loss function already gave us a good cross validation accuracy.

Challenges Faced

1. Data Limitations: We used the data from SigComp2009 and SigComp2011(Dutch Data) competition which contains unique genuine signatures of a total of less than 500 people. This is a major limitation since training CNN on so few unique signature patterns can lead to overfitting. Although more datasets relevant to genuine and fraud signatures exist (like SigComp2011 Chinese signatures and BHSig260 containing Hindi & Bengali Signatures), we did not use them for our training process owing to time constraints since the data structure for both is messy and it would have taken a significant amount of time to get them in an organized pattern to form pairs/ triplets.

2. Imbalanced Data: The dataset used were skewed towards forged signatures and they did not have enough genuine signatures corresponding to each person. This led to a data imbalance problem in the training step of logistic regression. We used under-sampling of the forged class to deal with this problem.

3. Compute Restrictions: Training of a Deep CNN architecture to get to the final image encoding takes significant computation time and resources. Since we were dealing with both time constraint and lack of GPU compute, we built a custom CNN architecture trained on CPUs that balanced between compute requirements and accuracy attained. We could not run multiple epochs as well due to the above problem.