Real Time Face Recognition Streaming System

Yuxin Ding
Electrical Engineering Department
Columbia University
NY, USA
yd2406@columbia.edu

Linlin Tian
Electrical Engineering Department
Columbia University
NY, USA
lt2667@columbia.edu

Jialin Zhao
Electrical Engineering Department
Columbia University
NY, USA
jz2862@columbia.edu

***Abstract*—This project designs a real time face recognition streaming system for videos. The system includes a haar cascade classifier, face landmark estimation, convolutional neural network and a linear SVM classifier. The system is implemented in spark using python. To improve throughput, optimization methods including data parallelize and load shedding were applied. In the experiment section, the system accuracy was computed, the threshold for load shedding was selected, the throughput improvement by optimization was calculated and a real time webcam video application featuring the three group members was implemented. The system throughput after optimization is 14.54 frames per second, increasing the system throughput by 197.13%. Frames dropped is 40.2% of total frames.**

***Index Terms*—Face Recognition, Video Streaming System**

I. Introduction

Live video streaming has become a popular way of entertainment and processing streaming videos has become a rising research area. Compared with images, the demand on system through put is much stricter.

Therefore, the project objective is to design a real time face recognition streaming system for videos and do optimizations for improving throughput.

The project has three major innovations and challenges:

1) Implement a face recognition streaming system in spark
2) Applied optimization techniques to improve system throughput, including data parallelize and load shedding
3) Experimented system on video datasets and live webcam streaming videos

II. System Overview

The following shows the workflow of single input image in the streaming face recognition system.

1) Collect an image frame from webcam video stream.
2) Detect faces with a pre-trained haar cascade model.
3) Apply face landmark estimation to get perfectly centered faces. The objective of face landmark estimation is locating 68 specific points, also called landmarks, that exist on every face, including Łthe top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then use affine transformations to get as perfect as possible centered faces. This operator can increase the face recognition accuracy.
4) Extract face features from the image from embed a convolutional neural network.
5) Label the person name using a SVM classifier.
6) Show image with bounding boxes and name labels on computer.

Every box in Fig 1 is an operator.

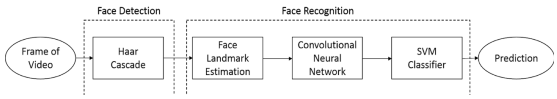

Fig. 1: Face Recognition Streaming System

III. Algorithm and Implementation

The system is developed in python environment and used pySpark to implement the system in a distributed system. Other than pySpark, python libraries opencv, sklearn, openface, pyTorch and dlib were used. The system is not implemented on a streaming processing system as Spark Streaming has no API for reading in images. However, since videos are a continuous sequence of frames, the system can be easily implemented on a streaming processing system with an image reading function.

A. Face Detection

The face detection algorithm applied in the system is Haar feature-based cascade classifiers. Haar feature-based cascade classifiers is an effective and efficient object detection method. [1] used a pre-trained classifier in python library opencv was applied for this project.

Haar-like features are extracted from images by using haar features as a convolution kernel and applying convolution. The haar features are as below. For this algorithm, a 24 * 24 size window was applied on the image. Among the over 160000 features computed, only 6000 was used to constructed the haar feature-based cascade classifiers. The feature selection was done by adaboost.

In the haar feature-based cascade classifiers, haar-like features are grouped into different stages of classifiers and applied one-by-one. If a window fails the first stage, discard it. The remaining features on it is no longer considered. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. For every window, the detector in total uses over 6000 features with 38 stages. However, only 10 features are evaluated among the 6000 features per window on average.

The map function in spark and opencv was used to implement this operator. The code is shown is shown in Fig. 2.

```
rdd_bbx = gray.map(lambda x: FaceDetection(x))
def FaceDetection(gray):
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    detected_faces = face_cascade.detectMultiScale(gray)
    return detected_faces
```

Fig. 2: Face Detection spark implementation

B. Face Landmark Estimation

The objective of face landmark estimation is locating 68 specific points, also called landmarks, that exist on every face, including Łthe top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. The specific landmark coordinates and location are shown below.

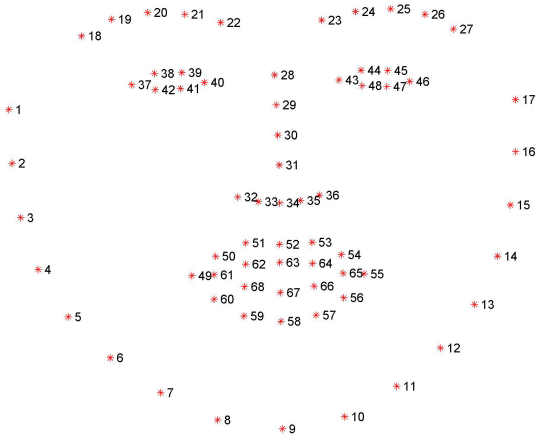

Fig. 3: Face Landmark Coordinates

For this project, a pre-trained face landmark estimation machine learning model in python library openface was used. The face landmark estimation algorithm is an ensemble of regression trees used to estimate directly from a sparse subset of pixel intensities the faces landmark positions by Kazemi and Sullivan. [2]

After locating the face landmarks in the image, the image was transformed by rotating or scaling the image, also called affine transformations, to get to get as perfect as possible centered faces.

The map function in spark and python library openface was used to implement this operator. The code is shown is shown in Fig. 4.

```
rdd_landmark = rdd_bbx.map(lambda x: Landmark(img, x))
def Landmark(img, face_rect):
    predictor_model = "shape_predictor_68_face_landmarks.dat"
    face_aligner = openface.AlignDlib(predictor_model)
    rect = dlib.rectangle(face_rect.left(), face_rect.top(), face_rect.width(), face_rect.height())
    alignedFace = face_aligner.align(96, img, rect, landmarkIndices=openface.AlignDlib.OUTER_EYES_AND_NOSE)
    return alignedFace
```

Fig. 4: Face Landmark Estimation Spark Implementation

C. Convolutional Neural Network

For the embedded convolutional neural network, a pre-trained neural network, FaceNet, was used. [3] It can extract 120 features from the face image. The structure of neural network is a modified version of FaceNets NN4 network model. It has one convolution layer and ten inception layers.

The map function in spark and pyTorch was used to implement the SVM classification operator. The implantation code is shown is shown in Fig. 5.

```
rdd_feature = rdd_landmark.map(lambda x: NeuralNet(x))
def NeuralNet(alginedFace):
    net = openface.TorchNeuralNet("./nn4.small2.v1.t7")
    feature = net.forward(alginedFace)
    feature = feature.reshape(1, -1)
    return feature
```

Fig. 5: Convolutional Neural Network Spark Implementation

D. Support Vector Machine Classifier

A linear support vector machine classifier was trained off-line and used the classifier for on-line prediction.

The map function in spark and python library sklearn was used to implement the SVM classification operator. The code is shown in Fig. 6.

```
rdd_label = rdd_feature.map(lambda x: SVMClassifier(x))
def SVMClassifier(feature):
    with open('./TrainSVM/feature/classifier.pkl', 'rb') as f:
        (le, clf) = pickle.load(f)
    pred = clf.predict(feature)
    person = le.inverse_transform(pred)
    return person[0]
```

Fig. 6: Support Vector Machine Classifier Spark Implementation

E. Output Display

To display the image output, the rdd files in the operators were collected as a list and displayed on the original image frame. The code is shown in Fig. 7.

```
bbx_list = rdd_bbx.collect()
for bbx in bbx_list:
    cv2.rectangle(frame, (bbx[0], bbx[1]), (bbx[0] + bbx[2], bbx[1] + bbx[3])
                  , (255, 0, 0), 2)

label_list = rdd_label.collect()
for person in label_list:
    cv2.putText(frame, person, (face_rect.left(), face_rect.top() - 10),
                cv2.FONT_HERSHEY_TRIPLEX, 0.5, (0, 255, 0), 1)

cv2.imshow('frame', frame)
```

Fig. 7: Display Frame Implementation

IV. Optimization

In order to increase the throughput of the streaming system, optimization techniques were applied to the system, including data parallelism and load shedding. The following section will discuss the optimization algorithm in detail.

A. Data Parallelism

In order to optimize the whole system, the first optimization technique applied was data parallelism. There can be more than one faces in the image. Therefore, after the face detection operator, haar casscade classifier, the face recognition operators can be ran on each of the faces in parallel. The face region are the regions located by bounding box by the face detection algorithm. Finally, the face detection and recognition results can be displayed on the screen.

The system overview with data parallelism is show in Fig 8. There can be multiple parallelized operators for face recognition and this is implemented by using the *parallelize* in pySpark.

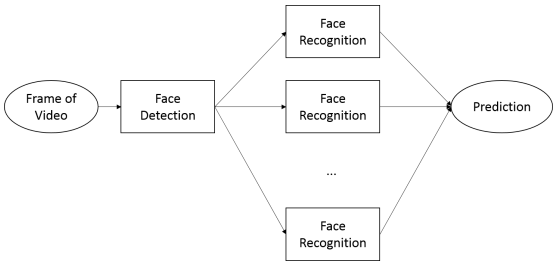

Fig. 8: Data Parallelism Optimization System Overview

The experiment results show that data parallelism improved the system throughput by 40.2%. The details of the experiment will be illustrated in the experiment section.

B. Load Shedding

Since adjacent frames in videos are very similar, unless there is a abrupt change of scenes, the positions of the faces are nearly static, and name label of the face are likely the same in these frame, making it meaningless to do face detection on every frame. Thus, load shedding technique was applied to improve the throughput of the system.

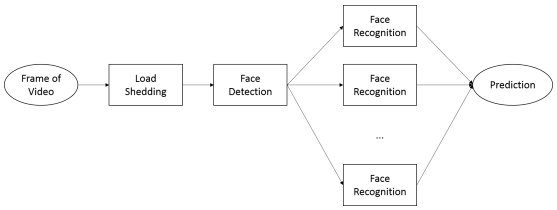

Fig. 9: Load Shedding Optimization System Overview

Fig. 9 shows the system overview with the added load shedding operator. The reason the load shedding operator is positioned in this position is because it can best increase efficiency. After examining the whole system, the face detection operator was tested to be the most time consuming operator in the whole system. Therefore, to efficiently increase throughput, the load shedding operator was positioned before the face recognition operator.

Mean square error(MSE) is used as the load shedding criteria as it is a simple indicator measuring the similarity of two images and the computation cost is very low. The pseudo-code for the load shedding algorithm is shown below.

Algorithm 1 LoadShedding(InputFrames,threshold)

```
ReferenceFrame ⟵ the first frame
Calculate Boundings and Labels on ReferenceFrame
while there exists unread frame do
  CurrentFrame ⟵ read the next frame
  if MSE(CurrentFrame,ReferenceFrame)<threshold then
    Show(CurrentFrame, Boundings and Labels)
  else
    ReferenceFrame ⟵ CurrentFrame
    Update Boundings and Labels on ReferenceFrame
    Show(CurrentFrame, Boundings and Labels)
  end if
end while
```

The first frame of the video is set as a reference frame. Then, for every input frame, the mean square error (MSE) between the current frame and reference frame is calculated. If the MSE value is less than a predefined threshold, then the face detection and face recognition of the frame is skipped, displaying the face detection and recognition result of reference frame and the current frame. If the MSE value is grater the predefined threshold, the current frame will be set as the new reference frame and the face detection and face recognition operators will be run to update the results and displayed on the screen.

Based on the experiment results, the predefined threshold is set as 700, as it is a good balance between efficiency and accuracy. When the threshold is set as 700, the system throughput is increased by 197.13%. The details of the experiment and results will be illustrated in the experiment section.

V. Dataset

Three datasets were used in the experiments,they are

- Face Detection Data Set and Benchmark (FDDB) [4]
- Public Figures Face Database (PubFig) [5]
- Movie Trailer Face Dataset [6]

The Face Detection Data Set and Benchmark (FDDB) is a dataset of face regions designed for studying the problem of unconstrained face detection. This data set contains the annotations for 5171 faces in a set of 2845 images taken from the Faces in the Wild data set. The dataset was used for testing face detection accuracy.

The Public Figures Face Database (PubFig) dataset is a large, real-world face dataset consisting of 58,797 images of 200 people collected from the Internet. The dataset was used for testing face recognition accuracy.

The Movie Trailer Face Dataset has 113 movie trailers from YouTube released in 2010 that contained celebrities present in the supplemented PubFig dataset. The Movie Trailer Face Dataset was used to detected.

VI. Experiment

The experiment section will introduce the four experiments done to test system accuracy, choose load shedding threshold, test system performance and a webcam application. The

system accuracy was measured by the accuracy of the face detection algorithm and the face recognition algorithm. The system performance was measured by the system throughput and the improvement by optimization techniques applied.

A. System Accuracy

1) Face Detection Accuracy: The face detection algorithm, haar cascade classifier, accuracy is tested on the Face Detection Data Set and Benchmark (FDDB) dataset. Intersection over Union (IoU) was measured for each image which is the union area of the predicted bounding boxes with the ground truth bounding boxes over the intersection area of these two bounding boxes.If the intersection over Union (IoU) for the image is larger than 0.5, then the image is considered as correctly predicted; Otherwise, not correctly predicted.

The face detection algorithm accuracy is 87.1%.

2) Face Recognition Accuracy: The face recognition algorithm accuracy is tested on the Public Figures Face Database (PubFig) dataset. The training-testing ratio for the dataset was 80-20.

The face detection algorithm testing accuracy is 93.6% and training accuracy is 90.6%.

B. Load Shedding Threshold Selection

To choose a perfect threshold to balance system accuracy and throughput, a range of thresholds from 0 to 1000 with step size of 50 were tested the results are shown in Fig.9.

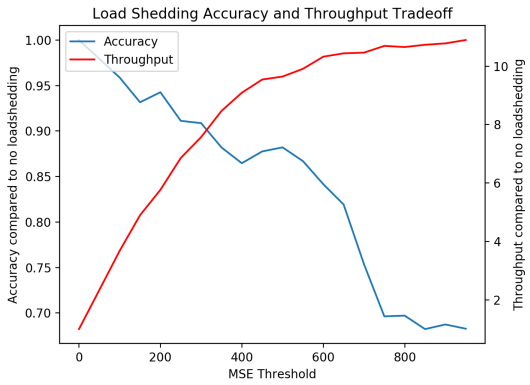

Fig. 10: Load Shedding Threshold Selection

The accuracy is a relative accuracy compared with when there is no load shedding, in other words when the threshold is set to zero. The throughput is the proportion increased compared with when there is no load shedding.

From the results shown in Fig. 9, when threshold is set to 700, the system has a good balance between accuracy and throughput. Therefore, the load shedding threshold is set to 700.

C. System Performance

In this experiment, the system throughput was chosen as a representative parameter to measure the system performance. The Movie Trailer Face Dataset was used to compute the system throughput.

The results of experimented the system throughput is shown in table I.

TABLE I: System Throughput Results

	System 1	System 2	System 3
Throughput (frames/s)	4.89	6.85	14.54
Increment	0	40.2%	197.13%

System 1 refers to the original system shown in Fig. 1 without any optimization. System 2 refers to the original system shown in Fig. 8 with data parallelism optimization. System 3 refers to the system shown in Fig. 9 with data parallelism and load shedding optimization.

From the results show, data parallelism and load shedding increased the system throughput by 197.13% and the optimized system throughput is 14.54. Frames dropped is 40.2% of total frames.

Interestingly, the experiment results show as resolution ratio increases, the throughput improvement after optimization increases as well. The reason is that the time needed to calculate a higher resolution image is longer than a smaller resolution ration image. Thus, using load shedding to skip similar frame contributes more to improving the speed of the system. There representative resolution ratio results are sown in table II.

TABLE II: System Throughput Optimization with Different Resolution

Resolution Ratio	System 1 Throughput	System 3 Throughput	Improvement	Frames Dropped
1280×720	3.55	12.96	264.98%	42.18%
720×480	8.82	12.69	54.44%	22.48%
460×240	26.99	33.27	23.67%	24.97%

D. Webcam Application

The system was implemented on a webcam video streaming application. The face recognition machine learning model was trained on the three group members. A screen shot of the application is shown in Fig.11.

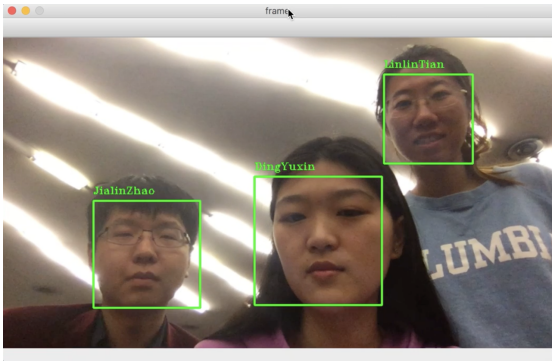

Fig. 11: Webcam Application Screen Shot

VII. Conclusion

In conclusion, the real time face recognition streaming system has four operators: haar cascaded classifier, face alignment, embedded convolutional neural network and linear support vector machine. Two optimization techniques were applied to the system: data parallelism and load shedding. The face detection algorithm accuracy is 87.1% and the face recognition accuracy is 90.6%. The system throughput after optimization is 14.54 frames per second, increasing the system throughput by 197.13%. Frames dropped is 40.2% of total frames.

VIII. Future Work

A. Streaming system Implementation

The system is implemented on Spark, a distributed processing package. However, according to the features of video, it would be a good idea to implement the system on a streaming application, such as IBM streams. IBM streams is a excellent tool to analyze a broad range of streaming data, including video, audio and make decisions in real-time.

B. System Throughput Optimization

Although two optimization techniques were implemented, there could also be more optimization methods to be experimented with. For example, fusion, load balancing or algorithm selection.

Moreover, in a distributed system, the system can be scaled to process multi-thread of videos in parallel.

C. Accuracy Improvement

The face detection algorithm used in the system is a haar cascade classifier. However, haar cascade classifier has a low detection accuracy when the faces are tilted to an angle or when the face in the image is small. There are more advanced face detection algorithms such as Faster-RCNN and, in the future, implementing these algorithms could increase accuracy and even throughput.

Acknowledgment

The authors would like to thank Professor Deepak S. Turaga for his technical support and advice given to the project.

References

[1] P. Viola and M. Jones, Rapid object detection using a boosted cascade of simple features, in IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition. CVPR 2001, vol. 1. IEEE, 2001, pp. 1511.

[2] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 18671874, 2014. 2, 3

[3] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan, Openface: A general-purpose face recognition library with mobile applications, Tech. Rep., CMU-CS-16-118, CMU School of Computer Science, 2016.

[4] Vidit Jain and Erik Learned-Miller. FDDB: A Benchmark for Face Detection in Unconstrained Settings. Technical Report UM-CS-2010-009, Dept. of Computer Science, University of Massachusetts, Amherst. 2010.

[5] N. Kumar, A. Berg, P. Belhumeur, and S. Nayar, B Attribute and simile classifiers for face verification,[in Proc. IEEE Int.Conf. Comput. Vis., 2009, DOI: 10.1109/ICCV. 2009.5459250.

[6] Enrique G Ortiz, Alan Wright, and Mubarak Shah, Face recognition in movie trailers via mean sequence sparse representation-based classification, in IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 35313538.