Face Recognition of Pedestrians from Live Video Stream using Apache Spark Streaming and Kafka

Abhinav Pandey, Harendra Singh

Abstract: ***Face recognition of pedestrians by analyzing live video streaming, aims to identify movements and faces by performing image matching with existing images using Apache Spark Streaming, Kafka and OpenCV, on distributed platform and derive decisions. Since video processing and analysis from multiple resources become slow when using Cloud or even any single highly configured machine, hence for making quick decisions and actions, Apache Spark Streaming and Kafka have been used as real time analysis frameworks, which deliver event based decisions making on Hadoop distributed environment. If continuous live events analysis is possible then the decision can make there-after or at the same time. And large amount videos in parallel processing are also not a bottleneck after getting the involvement of Hadoop because base of all real time analysis distributed tools is Hadoop. This event based analysis can be applied at any place where an immediate action is required like monitoring border areas of countries by cameras and drones, road traffic monitoring, life science domain, airlines, logo recognition and where-ever continuous monitoring and decision making involved in large scale data set.***

Keywords: Distributed System, Hadoop, Spark, Spark Streaming, Kafka, Open CV.

I. INTRODUCTION

1.1. Importance of Distributed Data Processing:

With the explosive growth rate data, it gradually becomes humungous and the term buzzing around the world is BigData, it is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them. Challenge includes analysis, capture, data accuracy search, sharing, transfer, visualization, querying, updating and information privacy. The term BigData often refers simply to the use of predictive analytics, user behaviour analysis, or certain other advanced data analysis methods that extract value from data and seldom to a particular size of data set. BigData usually includes data sets with sizes beyond the ability of commonly used software tools to capture, accurate, manage and process data within a tolerable elapsed time. BigData "size" is a constantly moving target, ranging from new dozen terabytes to many petabytes and even Exabyte of data. Big data requires a set of techniques and technologies with new forms of integration to reveal insight from datasets that are diverse, complex, and of a massive scale. Organizations are increasingly turning to BigData to discover new ways to improve decision-making, Opportunities and overall performance. For example, Big Data can be harnessed to address the challenges that arise when information that is dispersed across systems, BigData can help improve decision making capabilities.

Revised Version Manuscript Received on January 11, 2018.

Abhinav Pandey, Department of Computer Science & Engineering, Lakshmi Narain College of Technology Excellence, Bhopal (M.P.), India. E-mail: abhinavpndy@gmail.com

Mr. Harendra Singh, Assistant Professor, Department of Computer Science & Engineering, Lakshmi Narain College of Technology Excellence, Bhopal (M.P.), India. E-mail: harendra.cse07@gmail.com

It also can augment data warehouse solutions by serving as a buffer to process new data for inclusion in the data warehouse or to remove infrequently accessed or aged data. BigData can lead to improvements in overall operations by giving organizations greater visibility into operational issue. Operational insights might depends on machine data, which can include anything from computers to sensors or meters to GPS devices, BigData provides unprecedented insight on customers' decision-making processes by allowing companies to track and analyze shopping patterns, recommendations, purchasing behaviours and other drivers that knows to influence the sales. Cyber security and fraud detection is another use of BigData. With access to real-time data, Business can enhance security and intelligence analysis platforms. They can also process, store and analyze a wider variety of data types to improve intelligence, security and law enforcement insight.

1.2. Importance of Hadoop:

Apache Hadoop is an open source software framework for storage and processing large scale of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users.

All the modules in Hadoop are designed with fundamental assumption that hardware failures are a common occurrence and should be automatically handled by the framework. Hadoop was created by Doug Cutting and Mike Cafarella in 2005. It was originally developed to support distribution for the Nutch search engine project. Doug named Hadoop project after his son's yellow coloured toy elephant.

1.3. Hadoop and BigData:

Problem comes first and then solution comes, similarly BigData is currently a problem for IT world and Hadoop and It's frameworks introduced as a solution. Hadoop Eco-System is providing a complete solution for processing and analyzing BigData.

II. APACHE SPARK

Apache Spark provides a programming model that supports a much wider class of applications than MapReduce, while maintaining its automatic fault tolerance. In particular, MapReduce is inefficient for multi-pass applications that require low-latency data sharing across multiple parallel operations. These applications are quite common in analytics, and include:

- Iterative algorithms, including many machine learning algorithms and graph algorithms like PageRank.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

- Interactive data mining, where a user would like to load data into RAM across a cluster and query it repeatedly.
- Streaming applications that maintain aggregate state over time.

Traditional MapReduce and DAG engines are suboptimal for these applications because they are based on acyclic data flow: an application has to run as a series of distinct jobs, each of which reads data from stable storage (e.g. a distributed file system) and writes it back to stable storage. They incur significant cost loading the data on each step and writing it back to replicated storage.

Spark offers an abstraction called resilient distributed datasets (RDDs) to support these applications efficiently. RDDs can be stored in memory between queries without requiring replication. Instead, they rebuild lost data on failure using lineage: each RDD remembers how it was built from other datasets (by transformations like map, join or groupBy) to rebuild itself. RDDs allow Spark to outperform existing models by up to 100x in multi-pass analytics. The RDDs can support a wide variety of iterative algorithms, as well as interactive data mining and a highly efficient SQL engine (Shark).

Below are some Spark component which makes Spark better in many dimensions:

- SparkR: Scaling R Programs with Spark, Shivaram Venkataraman, Zongheng Yang, Davies Liu, Eric Liang, Hossein Falaki, Xiangrui Meng, Reynold Xin, Ali Ghodsi, Michael Franklin, Ion Stoica, and Matei Zaharia. SIGMOD 2016. June 2016.
- MLlib: Machine Learning in Apache Spark, Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Journal of Machine Learning Research (JMLR). 2016.
- Spark SQL: Relational Data Processing in Spark. Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia. SIGMOD 2015. June 2015.
- GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. Reynold S. Xin, Daniel Crankshaw, Ankur Dave, Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica. OSDI 2014. October 2014.
- Discretized Streams: Fault-Tolerant Streaming Computation at Scale. Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica. SOSP 2013. November 2013.
- Shark: SQL and Rich Analytics at Scale. Reynold S. Xin, Joshua Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, Ion Stoica. SIGMOD 2013. June 2013.
- Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, Ion Stoica. HotCloud 2012. June 2012.
- Shark: Fast Data Analysis Using Coarse-grained Distributed Memory (demo). Cliff Engle, Antonio Lupher, Reynold S. Xin, Matei Zaharia, Haoyuan Li, Scott Shenker, Ion Stoica. SIGMOD 2012. May 2012. Best Demo Award.
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. NSDI 2012. April 2012. Best Paper Award.
- Spark: Cluster Computing with Working Sets. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. HotCloud 2010. June 2010.

Video processing may involve a number of steps and algorithms, for example, face detection, shadow removing and background subtraction, feature extraction, and so on. Some of these steps can be processed in parallel, and some may be needed to processed online immediately. An example of this that face recognition (e.g, the algorithm come with OpenCV) can be executed very quickly, it is better to put such kind of lightweight algorithm on more than one nodes in order to get the fast response. This is beneficial for some applications, for example, a smart tourist guide can help to count people in real time in order to help on visiting route planning. Some heavy weight algorithms need to be implemented in the batch processing layer offline, e.g. GMM (Gaussian mixture model) for background subtraction is comparatively heavy weight algorithm, which should be put into effect offline if a system needs to process streaming video data.

Therefore in order to process video effectively and efficiently, we need to utilize both batch processing capabilities and also real time in-memory processing capabilities of other platforms. This is actually what is missing in the current video processing research. Hence forth in this paper, presenting work on a high-performance video on distributed platform which integrates both distributed and fast processing.

Image processing requires Stream data processing for the following reasons:

1. Video processing requires on time processing for surveillance
2. Processing video data after some span of time is hard and memory intensive.
3. In order to process Image data, Data streaming or live data transfer from source to the stream data processing cluster has to be dealt efficiently.
4. Apache Kafka deals the problem of live data transfer with high efficiency.

III. APACHE KAFKA IMPORTANCE

- Apache Kafka is a distributed streaming platform that provides a system for publishing and subscribing to streams of records. These records can be stored in fault-tolerant way and consumers can process the data.

It is an open-source message broker project developed by the Apache Software

- Foundation wrote in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. The design is heavily influenced by transaction logs.
- To derive the real value from big data, any kind of information loss cannot be afforded. Apache Kafka is designed with O (1) disk structures that provide constant-time performance even with very large volumes of stored messages, which is in order of TB.

IV. APACHE STORM IMPORTANCE

Apache Spark is a fast, generalised cluster-computing system. It provides modules for SQL, structured, semi-structured and unstructured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. OpenCV (Open source Image Processing library): It is an open-source BSD-licensed library, designed for computational efficiency and strongly focused on real-time applications. This library is written in C++ but provides a Java API as well. OpenCV includes several hundred CV algorithms that can be used for processing and analysing image and video files. Open-source image detection libraries , used to images extracted from the videos. It is a well documented and widely used project. More than 500 functions are implemented in OpenCV to cover areas such as robotics, security, medical imaging or factory product inspection.

It also includes a general-purpose Machine Learning Library (MLL) focused on statistical pattern recognition. On the other hand, OpenIMAJ (OPEN Intelligent Multimedia Analysis in Java) is a Java library and tool for scalable multimedia content analysis and image indexing. It includes broad state-of-the-art computer vision techniques. The distribution is made using a modular set of jar files under a BSD-style license. The design and implementation keep all the components modular to maximize code maintainability. Both libraries employ similar algorithms to detect images. With all these similarities, we selected Open IMAJ to implement our code that use the Scale-Invariant Feature Transform (SIFT) to extract some interest pixels from images and describe them. To find analog pixels, we use Random Sample Consensus (RANSAC) to fit a geometric model called an Affine Transform to the initial set of matches. After the pattern detection process, we obtain a number of matches or similarities. Closeness can be compared with a threshold. If the number is greater, we have a frame that contains the logotype. Notice that the goodness of the systems greatly depends on the goodness of the library.

V. EXISTING METHODOLOGIES

For Pedestrian and Face detection:

Apart from face detection in the research paper, authors have successfully implemented pedestrian detection techniques, which are for finding people on the road. Most research in this field has focused on finding people standing than sitting or lying, although it could be useful for saving a life in a disastrous situation and many other use cases. The applications of pedestrian detection technique are striking.

Detectors in OpenCV, open vision library, are representative among published detectors.

The histogram of Oriented Gradients (HOG) (INRIA) – HOG detector of

Dalal 2005, which is learned through INRIA Person Database. It is difficult to

detect the object of a template which is the smaller size than 64(w)x128(h).

HOG(Daimler) – HOG detects which is learned through Daimler Pedestrian Dataset. It could detect objects of a small template.

Hogcascades – Detector which is applied cascade technique in HOG feature.

Haarcascades – Detector of Viola2001, whose speed to detect objects is fast than others.

Haarcascades has been used as our detector for finding pedestrians , since our purpose of this paper is just detecting in distributed environment rather than focusing on an accuracy of detection.

Problem type: Face recognition has been a sought after the problem of biometrics and it has a variety of applications in modern life. The problems of face recognition attract researchers working in biometrics, pattern-recognition field, and computer vision. Several face recognition algorithms are also used in many-different applications apart from biometrics, such as video compressions, indexings etc. They can also be used to classify multimedia content, to allow fast and efficient searching for material that is of interest to the user. An efficient face recognition system can be of great help in forensic sciences, identification for law enforcement, surveillance, authentication for banking and security system, and giving preferential access to authorized users i.e. access control for secure areas etc. The problem of face recognition has gained even more importance after the recent increase in the terrorism related incidents. Use of face recognition for authentication also reduces the need of remembering passwords and can provide a much greater security if face recognition is used in combination with other security measures for access control. The cost of the license for an efficient commercial Face recognition system ranges from 30,000 $ to 150,000 $ which shows the significant value of the problem.

VI. PROPOSED METHODOLOGY

Pedestrian's face detection is the process of finding the change in position and identifying face of an object (often a human) relative to its surroundings. It is used mostly in video-surveillance systems that continuously monitor a specific area. An algorithm provided by the CV libraries analyses the video feed sent by such a camera and looks for any motion and face recugnition. Detecting motion triggers an event that can send a message to an application or alert the user.

This work dedicated for video stream analytics, has three main components:

1. Video stream collector
2. Stream data buffer
3. Video stream processor

The video stream collector receives the video stream data from a cluster of IP cameras. This component serializes the video frames to stream data buffer, which is a fault-tolerant data queue for streaming video data. The video stream processor consumes the stream data from buffer and processes it. This component will apply video-processing algorithms to detect motion in the video-stream data. Finally, the processed data or image files will be stored in a S3 bucket or HDFS directory. This framework performs face recognition of pedestrians from live video streaming where The Kafka, Spark Streaming and OpenCV are being used to analyze Surveillance video. The system will extract human faces from surveillance video and compares with faces in database to detect and the decision will be forwarded to concerned departments. The system is a real time face recognition for surveillance video. The Kafka – Agent collects the images from various videos sources at the desired frequency.

It transmits the image string data to the Kafka Cluster. The entry level of the Spark Streaming, which receives the binary data from Kafka Cluster. The following sections provide design and implementation details of the video stream collector, stream data buffer, and video stream processor in the sample application.

6.1. Video Stream Collector:

The video stream collector works with a cluster of IP cameras that provide live video feeds. The component must read the feed from each camera and convert the video stream into a series of video frames. To distinguish each IP camera, the collector maintains the mapping of camera ID and URL with camera.url and camera.id properties in a stream-collector. properties file. These properties can have comma-separated lists of camera URLs and IDs. Different cameras may provide data with different specifications such as the codec, resolution, or frames per second. The collector must retain these details while creating frames from the video stream.

The video stream collector uses the OpenCV video-processing library to convert a video stream into frames. Each frame is resized to the required processing resolution (e.g. 640x480). OpenCV stores each frame or image as a Mat object. Mat needs to be converted in serialise-able (byte-array) form by keeping intact the details of frame — i.e. rows, columns, and type. The video stream collector uses the following JSON message structure to store these details.

{"cameraId":"cam-01","timestamp":1488627991133,"rows":12,"cols":15,"type":16,"data":"asdfh"}

cameraId is the unique ID of the camera. timestamp is the time at which the frame was generated. rows, cols, and type are OpenCV Mat-specific details. data is a base-64 encoded string for the byte array of the frame.

The video stream collector uses the Gson library to convert the data to JSON messages, which are published in the video-stream-event topic. It sends the JSON messages to the Kafka broker using the KafkaProducer client. KafkaProducer sends data into the same partition for each key and order of these messages is guaranteed.

```
JsonObject obj = new JsonObject();
obj.addProperty("cameraId",cameraId);
obj.addProperty("timestamp", timestamp);
obj.addProperty("rows", rows);
obj.addProperty("cols", cols);
obj.addProperty("type", type);
obj.addProperty("data",
Base64.getEncoder().encodeToString(data));
String json = gson.toJson(obj);
producer.send(new ProducerRecord<String,
String>(topic,cameraId,json),new
EventGeneratorCallback(cameraId));
```

Kafka is primarily designed for text messages of small sizes but a JSON message comprising the byte array of a video frame will be large (e.g. 1.5 MB), so Kafka will require configuration changes before it can process these larger messages. The following KafkaProducer properties need to be adjusted:

batch.size
max.request.size
compression.type

6.2. Stream Data Buffer:

To process a huge amount of video stream data without loss, it is necessary to store the stream data in temporary storage. The Kafka broker works as a buffer queue for the data that the video stream collector produces. Kafka uses the file system to store the messages, and the length of time it retains these messages is configurable.

Keeping the data in storage before processing ensures its durability and improves the overall performance of the system as processors can process data at different times and at different speeds depending on the load. This improves the reliability of the system when the rate of data production exceeds the rate of data processing.

Kafka guarantees the order of messages in a single partition for a given topic. This is extremely helpful for processing data when the order of the data is important. To store large messages, the following configurations might need to be adjusted in the server.properties file of the Kafka server:

message.max.bytes
replica.fetch.max.bytes
Video Stream Processor

The video stream processor performs three steps:

Read the JSON messages from the Kafka broker in the form of a VideoEventData dataset.

Group the VideoEventData dataset by camera ID and pass it to the video stream processor.

Create a Mat object from the JSON data and process the video stream data.

6.3. Video Stream Processor

The video stream processor is built on Apache Spark. Spark provides a Spark Streaming API, which uses a discretized stream or DStream, and a new Structured Streaming API based on a dataset.

This application's video stream processor uses the Structured Streaming API to consume and process JSON messages from Kafka. Please note this application processes structured data in the form of JSON messages and The unstructured video data is an attribute of these JSON messages that the video stream processor will process. The Spark documentation states "Structured Streaming provides fast, scalable, fault-tolerant, end-to-end exactly-once stream processing without the user having to reason about streaming." This is why the video stream processor is designed around Spark's Structured Streaming. The Structured Streaming engine provides built-in support for structured text data and state management for aggregation queries. This engine also provides features like processing of non-aggregate queries and external state management of datasets (a new feature in Spark 2.2.0).

To process large messages, the following Kafka consumer configurations must be passed to the Spark engine:

max.partition.fetch.bytes

max.poll.records

The main class for this component is VideoStreamProcessor. This class first creates a SparkSession object that is the entry point for working with the Spark SQL engine. The next step is to define a schema for incoming JSON messages so that Spark can use this schema to parse the string format of a message into JSON format. Spark's bean encoder can transform this into Dataset<VideoEventData>. VideoEventData is a Java bean class that holds the data of JSON message.

```
Dataset<VideoEventData> ds = spark.readStream().format("kafka")
.option("kafka.bootstrap.servers",prop.getProperty("kafka.bootstrap.servers"))
.option("subscribe",prop.getProperty("kafka.topic"))
.option("kafka.max.partition.fetch.bytes",prop.getProperty("kafka.max.partition.fetch.bytes"))
.option("kafka.max.poll.records",
prop.getProperty("kafka.max.poll.records"))
.load().selectExpr("CAST(value AS STRING) as message")
.select(functions.from_json(functions.col("message"),schema).as("json"))
.select("json.*").as(Encoders.bean(VideoEventData.class));
```

Figure 3. Code snippet for processing of kafka messages by spark streaming

Next, groupByKey groups the dataset by camera ID to get KeyValueGroupedDataset<String, VideoEventData>. It uses a mapGroupsWithState transformation to work on a group of VideoEventData (Iterator<VideoEventData>) for the current batch of video frames that are grouped by camera ID. This transformation first checks that the last processed VideoEventData (video frame) is present and passes that to the video processor for next step of processing. After video processing, the last processed VideoEventData (video frame) is returned from the video processor and the state updates. To start the streaming application, the writeStream method is called on the dataset with console sink and update output mode.

The video stream processor uses the OpenCV library to process video stream data. Our application is meant to detect motion; VideoMotionDetector is the class with the logic for detecting motion in a series of frames. The first step in this process is to sort the list of VideoEventData (Iterator<VideoEventData>) by timestamp for a given camera ID to compare video frames in order. The next step is to iterate the sorted list of VideoEventData objects and convert them to an OpenCV Mat object. If the last processed video frame is available, then it uses that as the first video frame for processing the current series of frames. VideoMotionDetector compares two consecutive frames and detects the differences using an API provided by the OpenCV library. If it finds differences that exceed defined criteria, these are considered to be motion. VideoMotionDetector will save this detected motion in form of image file to a preconfigured S3 bucket or HDFS directory. This image file can undergo further processing by another application or VideoMotionDetector can trigger an event to notify a user or application it has detected motion then the face recugnotion process will be started and faces will be compared with existing faces in local DataBase, An alarm will be raised if face got matched.

Block Diagram:

6.4. Tool Used:

Ubuntu 16, Hadoop 2.6, Spark 2.1, Kafka, HUE

VII. ALGORITHM

For pedestrian: Haarcascades – Detector of Viola2001, whose speed to detect objects is fast than others. For Face recognition: Eigenfaces, the problem with the image representation we are given is its high dimensionality. Two-dimensional grayscale images span a -dimensional vector space, so an image with pixels lies in a -dimensional image space already. The question is: Are all dimensions equally useful for us? We can only make a decision if there's any variance in data, so what we are looking for are the components that account for most of the information. The Principal Component Analysis (PCA) was independently proposed by Karl Pearson (1901) and Harold Hotelling (1933) to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components Fisher faces: The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximize the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information.

may be lost when throwing components away. Imagine a situation where the variance in your data is generated by an external source, let it be the light. The components identified by a PCA do not necessarily contain any discriminative information at all, so the projected samples are smeared together and a classification becomes impossible (see http://www.bytefish.de/wiki/pca_lda_with_gnu_octave for an example).

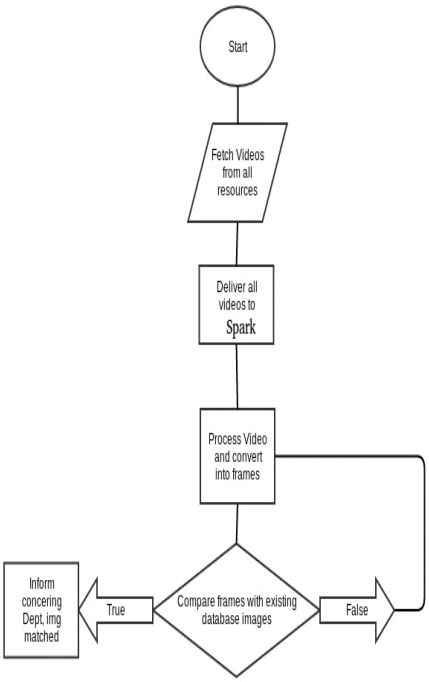
The Linear Discriminant Analysis performs a class-specific dimensionality reduction and was invented by the great statistician Sir R.A Fisher. He successfully used it for classifying flowers in his 1936 paper The use of multiple measurements in taxonomic problems. In order to find the combination of features that separates best between classes the Linear Discriminant Analysis maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation.

Local Binary Patterns Histograms: Eigenfaces and Fisherfaces take a somewhat holistic approach to face recognition. You treat your data as a vector somewhere in a high-dimensional image space. We all know high-dimensionality is bad, so a lower-dimensional subspace is identified, where (probably) useful information is preserved. The Eigenfaces approach maximizes the total scatter, which can lead to problems if the variance is generated by an external source, because components with a maximum variance over all classes aren't necessarily useful for classification (see http://www.bytefish.de/wiki/pca_lda_with_gnu_octave). So to preserve some discriminative information, applied a Linear Discriminant Analysis and optimized as described in the Fisherfaces method. The Fisherfaces method worked great... at least for the constrained scenario we've assumed in our model.

Now real life isn't perfect. You simply can't guarantee perfect light settings in your images or 10 different images of a person. So what if there's only one image for each person? The covariance estimates for the subspace may be horribly wrong, so will the recognition. Remember the Eigenfaces method had a 96% recognition rate on the AT&T Facedatabase? How many images do we actually need to get such useful estimates? Here are the Rank-1 recognition rates of the Eigenfaces and Fisherfaces method on the AT&T Facedatabase, which is a fairly easy image database. So in order to get good recognition rates, you'll need at least 8(+-1) images for each person and the Fisherfaces method doesn't really help here. The above experiment is a 10-fold cross validated result carried out with the framework at: https://github.com/bytefish/facerec. This is not a publication, so I won't back these figures with a deep mathematical analysis. Please have a look into for a detailed analysis of both methods, when it comes to small training datasets.

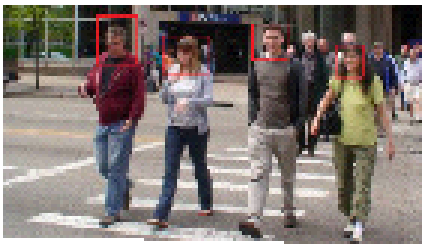
So some research concentrated on extracting local features from images. The idea is to not look at the whole image as a high-dimensional vector, but describe only local features of an object. The features you extract this way will have a low-dimensionality implicitly. A fine idea! But you'll soon observe the image representation we are given doesn't only suffer from illumination variations. Think of things like scale, translation or rotation in images - your local description has to be at least a bit robust against those things. Just like SIFT, the Local Binary Patterns methodology has its roots in 2D texture analysis. The basic idea of Local Binary Patterns is to summarize the local structure in an image by comparing each pixel with its neighborhood. Take a pixel as center and threshold its neighbors against. If the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. You'll end up with a binary number for each pixel, just like 11001111. So with 8 surrounding pixels you'll end up with 2^8 possible combinations, called Local Binary Patterns or sometimes referred to as LBP codes. The first LBP operator described in literature actually used a fixed 3 x 3 neighborhood just like this:

Flowchart:



VIII. RESULT ANALYSIS

Framework successfully detects pedestrians and faces by implementing face recognition and Haarcascades algorithms.



Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

S. No.	Comparison Criteria	Proposed System	Previous Systems
1	Capability to perform match with existing image and image received from live streaming video	Yes	No
2	System is distributed	Yes	Yes
3	Time effective	Yes (comparatively similar timings with previous system)	Yes
4	Integration with other frameworks like Kafka	Yes	Yes

IX. CONCLUSION AND FUTURE WORK

This paper studies real-time pedestrians and faces detection from a live video stream, and proposes a generic framework that converts an offline pedestrian and face recognition algorithm to a real-time one running on distributed platform in order to obtain high efficiency, scalability. Compared to baseline solutions, the proposed framework is carefully designed to enable high parallelism, eliminate performance bottlenecks. Experimental evaluations using real data validate this framework performance claims. Regarding future work, an interesting direction is to apply tracking techniques recognized more objectives like ammunition and weapons, which are generally more efficient than recognizing pedestrians and faces in every frame. Meanwhile, intend to further optimize like fast processing and analysis with low latency time and better front-end for analyzing and decision making.

LITERATURE SURVEY

1. Md Azher Uddin , Joolekha Bibi Joolee, Aftab Alam, And Young-Koo Lee Department of Computer Science and Engineering, Kyung Hee University, Yongin 1732, South Korea Corresponding author: Young-Koo Lee (yklee@khu.ac.kr) “] Human Action Recognition Using Adaptive Local Motion Descriptor in Spark”
2. Du-Hyun Hwang, Yoon-Ki Kim and Chang-Sung Jeong, "Real –Time Pedestrian Detection Using Apaches Torm In A Distributed Environment".
3. Shreenath Dutt, Ankita Kalra, "A Scalable and Robust Framework for Intelligent Real-time Video Surveillance"
4. Lokesh Babu Rao, C. Elayaraja 2, "Image Analytics on Big Data In Motion - Implementation of Image Analytics CCL in Apache Kafka and Storm"
5. OpenCV http://docs.opencv.org/2.4/modules/contrib/doc
6. StormCV https://github.com/sensorstorm/StormCV
7. Jianbing Ding, Hongyang Chao, Mansheng Yang, "Real-Time Logo Recognition from Live Video Streams using an Elastic Cloud Platform".
8. Weishan Zhang, Liang Xu, Pengcheng Duan, Wenjuan Gong, Xin Liu, Qinghua Lu, "Towards a High Speed, Video Cloud Based on Batch processing Integrated with Fast processing"
9. Hongyang Chao, "Real-Time Logo Recognition from Live Video Streams Using an Elastic Cloud Platform"
10. Vaithilingam Anantha Natarajan, Subbaiyan Jothilakshmi, Venkat N Gudivada, "Scalable Traffic Video Analytics using Hadoop MapReduce"
11. Supun Kamburugamuve, Leif Christiansen, and Geoffrey Fox, "A Framework for Real Time Processing of Sensor Data in the Cloud"