

Study and Development of Feature Discovery Model (Cancer cell Analysis)



**Fall 2016
Increment 2nd**

**Student ID:
Gupta, Arunit - 7**

Objective

Nowadays, machine learning and big data is becoming the integral part of intelligent application. There are lot of open source repositories and tools available for machine learning. But there are very few architectures which combines machine learning with big data. The objective of this directed reading is to design the architecture for integration of machine learning and big data which uses available open source tools. The objective includes creation of application for detection of features of the streaming video using the big data framework to achieve high performance. Features of video can be defined as change in scene, frame detection, audio detection, face detection, object tracking. This will help user to analyze the video in terms of features and can help them to search another related videos using features identified, compare video with another, for analyzing the current trends among people. The directed reading involves features extraction of video using libraries, creating distributed framework which includes open source big data software like Apache Storm, Apache Kafka, Apache Spark, using machine learning algorithm for training and testing the model and annotate the video with extracted feature.

The overall objective is to build a distributed model which is capable of detecting the features and functionality present in the streaming video and annotate the video based on the extracted features.

Expected Outcomes:

The primary objective of this course is to develop an application which is capable of detecting the features of the videos based on the define model.

Progress Plan

1. Proposing and Building workflow model for the project.
2. Analysis of Health domain and search for related streaming input videos.
3. Implementing mechanism to fetch the live streaming or static videos
4. Study of Spark, Storm and Kafka frameworks.
5. Extraction of various parameter from videos using OpenIMAJ library.
6. Extracting Image features for processing: SIFT, HOG and Color.
7. Study and Implementation of Video Extraction techniques: Frame slicing and shot transition.
8. Audio Extraction and Feature vector generation/selection/filtering/compression.
9. Implementing storm with feature vector.
10. Integrating and connecting Spark, Storm and Kafka.
11. Base64 implementation for Video conversion from Kafka to Spark.
12. Implementing machine learning algorithm for testing and training video streams.
13. Study and implementation of Annotating videos based on feature extraction
14. Combining all modules and techniques and apply to live streaming video and annotate video based on extracted features

Project Increment 1

Research Covered:

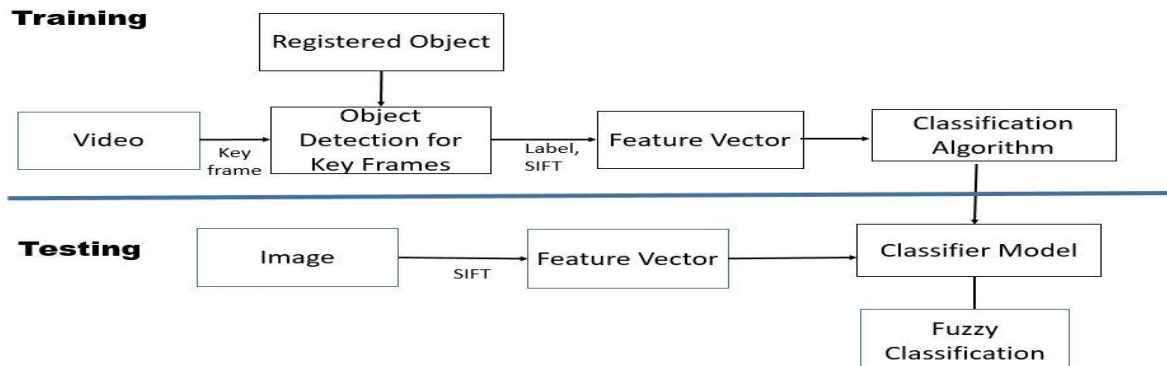
- Storm and Kafka Integration
- Spark and Kafka Integration
- Video Processing and Feature Vector Implementation

Terminology

- **Apache Spark:** It is an open source cluster computing framework. Originally developed at the University of California, Berkeley's AMP Lab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.
- **Apache Storm** is a distributed stream processing computation framework written predominantly in the Clojure programming language. Originally created by Nathan Marz and team at BackType, the project was open sourced after being acquired by Twitter. It uses custom created "spouts" and "bolts" to define information sources and manipulations to allow batch, distributed processing of streaming data. The initial release was on 17 September 2011.
- **Apache Kafka:** Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds

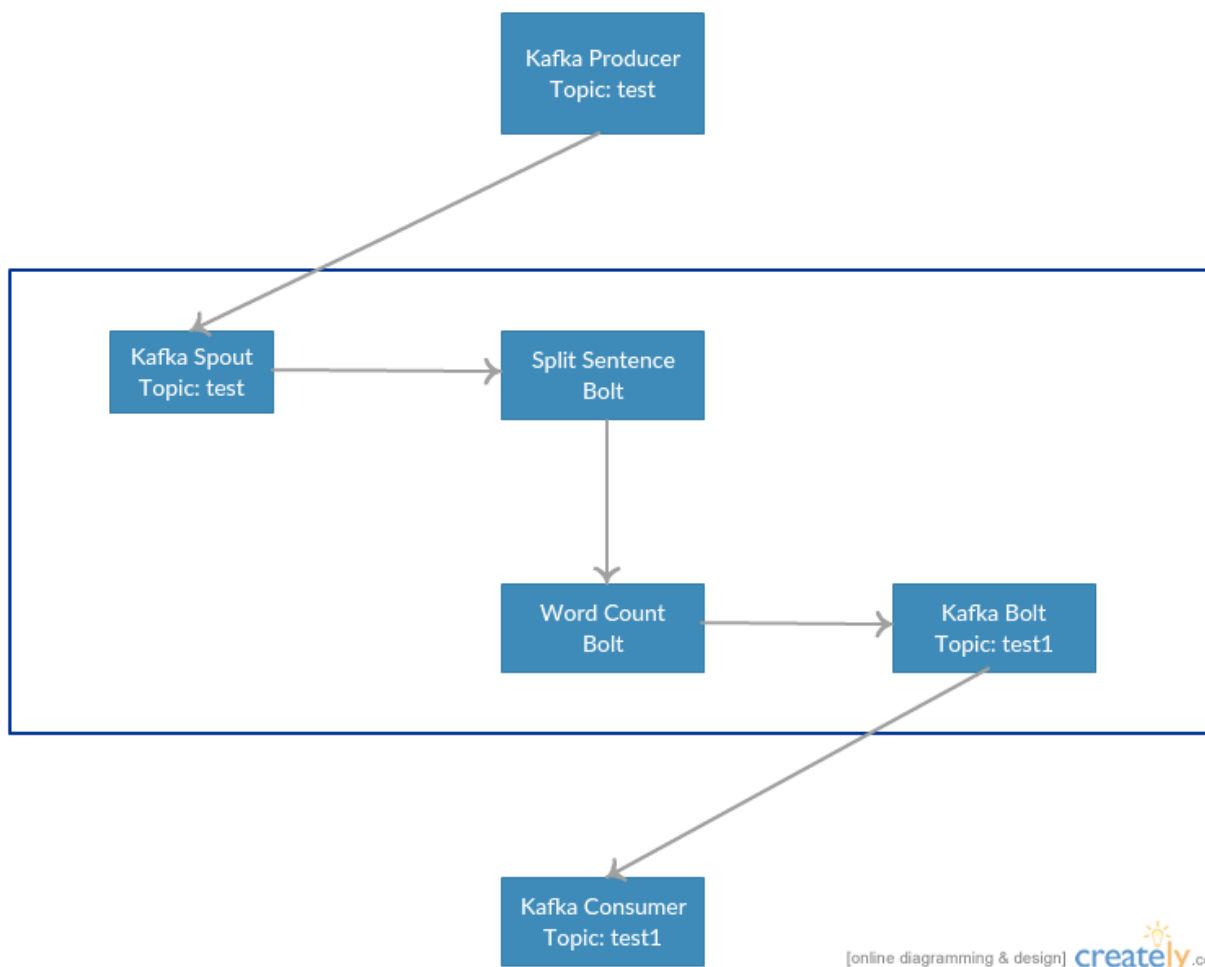
Work Flow Model

Training



Storm -Kafka Connection

Kafka and Storm naturally complement each other, and their powerful cooperation enables real-time streaming analytics for fast-moving big data. Kafka and Storm integration is to make easier for developers to ingest and publish data streams from Storm topologies.



[online diagramming & design] createely.com

Workflow Explanation:

1. Kafka Producer (Kafka) -> Kafka Consumer (Kafka Spout)
Kafka spout which works in the storm framework, consumes data from Kafka producer on some topic i.e. test

2. Kafka Spout -> Split sentence bolt
Spout transfer data to split sentence bolt to split the sentences into words.
3. Split Sentence Bolt -> Word Count Bolt
Split sentence bolt sent words to Word count bolt to count the words.
4. Word Count Bolt -> Kafka Bolt (Kafka Producer)
Counts the words and sent it to Kafka Bolt.
5. Kafka Bolt -> Kafka Consumer
Kafka Bolt is kafka producer in storm framework which produces messages on some topic i.e. test1 and sent it to consumer.

Status: Completed till word count bolt, facing some issue with Word Count bolt to Kafka bolt. It is the last step which needs to be solved for completion of storm – Kafka workflow.

Screenshots

- Word Count Bolt Program

```
/**
 * Created by Mayanka on 17-Sep-15.
 */
public class WordCountBolt extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<>();
    static Logger log;
    @Override
    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        String word = tuple.getStringByField("words");
        log = LoggerFactory.getLogger(BaseBasicBolt.class);
        Integer count = counts.get(word);
        if (count == null)
            count = 0;
        count++;
        counts.put(word, count);
        try {
            BufferedWriter br = new BufferedWriter(new FileWriter(new File("output"), true));
            br.append(word + ":" + count + "\n");
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        log.info("Word:" + word + " Count:" + count);
        // insertIntoMongoDB(word, count);
        log.info("WordStore:" + word + " CountStore:" + count);
        basicOutputCollector.emit(new Values(word, count));
    }

    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
```

- Split Sentence Bolt Program

```
import ...

/**
 * Created by Mayanka on 16-Sep-15.
 */
public class SplitSentenceBolt extends BaseBasicBolt {

    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        String s= tuple.getString(0);
        String a[]=s.split(" ");
        for (int i=0;i<a.length;i++)
        {
            basicOutputCollector.emit(new Values(a[i]));
        }
    }

    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("words"));
    }
}
```

- Storm Kafka Producer

```
*/
public class StormKafkaProducer {
    static Logger log;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        BasicConfigurator.configure();
        log = LoggerFactory.getLogger(StormKafkaProducer.class);

        if (args != null && args.length > 0) {
            try {
                StormSubmitter.submitTopology(
                    args[0],
                    createConfig(false),
                    createTopology());
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } else {
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology(
                "Kafka_Bolt_Test",
                createConfig(true),
                createTopology());
            try {
                Thread.sleep(60000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            cluster.shutdown();
        }
    }
}
```


Spark and Kafka Workflow

Kafka is a (pub-sub) messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. After data is processed, Spark Streaming could be publishing results into another Kafka topic or store in HDFS, databases or dashboards.

Working on Spark and Kafka integration.

Workflow includes:

Kafka Producer → Spark Streaming → Kafka Consumer

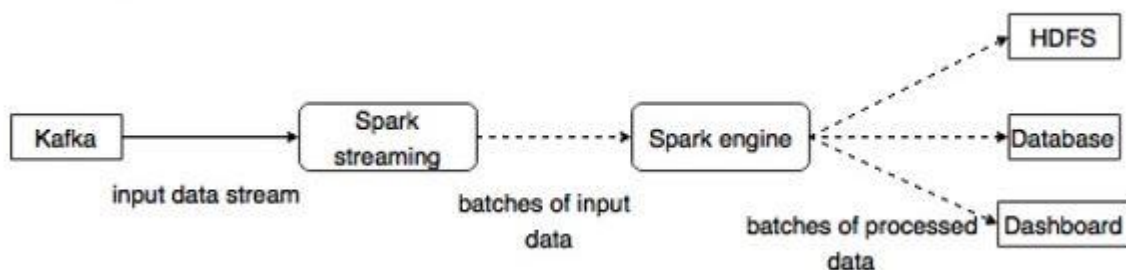


Fig1: Kafka Producer Integration with spark engine

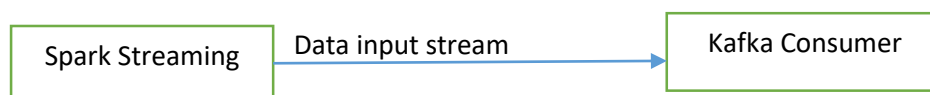


Fig 2: Spark Streaming data to Kafka Consumer

Spark and Kafka Integration

- Kafka Producer Code

```
// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }

    val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args

    // Zookeeper connection properties
    val props = new HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)

    // Send some messages
    while(true) {
      (1 to messagesPerSec.toInt).foreach { messageNum =>
        val str = (1 to wordsPerMessage.toInt).map(x => scala.util.Random.nextInt(10).toString)
          .mkString(" ")

        val message = new ProducerRecord[String, String](topic, null, str)
        producer.send(message)
      }

      Thread.sleep(1000)
    }
  }
}
```

- Spark Kafka Word Count

```
*      my-consumer-group topic1,topic2 1`
*/
object KafkaWordCount {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>")
      System.exit(1)
    }

    StreamingExamples.setStreamingLogLevels()

    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(2))
    ssc.checkpoint("checkpoint")

    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1L))
      .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(2), 2)
    wordCounts.print()

    ssc.start()
    ssc.awaitTermination()
  }
}

// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }
  }
}
```

- Kafka Consumer

```
// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }

    val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args

    // Zookeeper connection properties
    val props = new HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)

    // Send some messages
    while(true) {
      (1 to messagesPerSec.toInt).foreach { messageNum =>
        val str = (1 to wordsPerMessage.toInt).map(x => scala.util.Random.nextInt(10).toString)
          .mkString(" ")

        val message = new ProducerRecord[String, String](topic, null, str)
        producer.send(message)
      }

      Thread.sleep(1000)
    }
  }
}
```

- Kafka Output

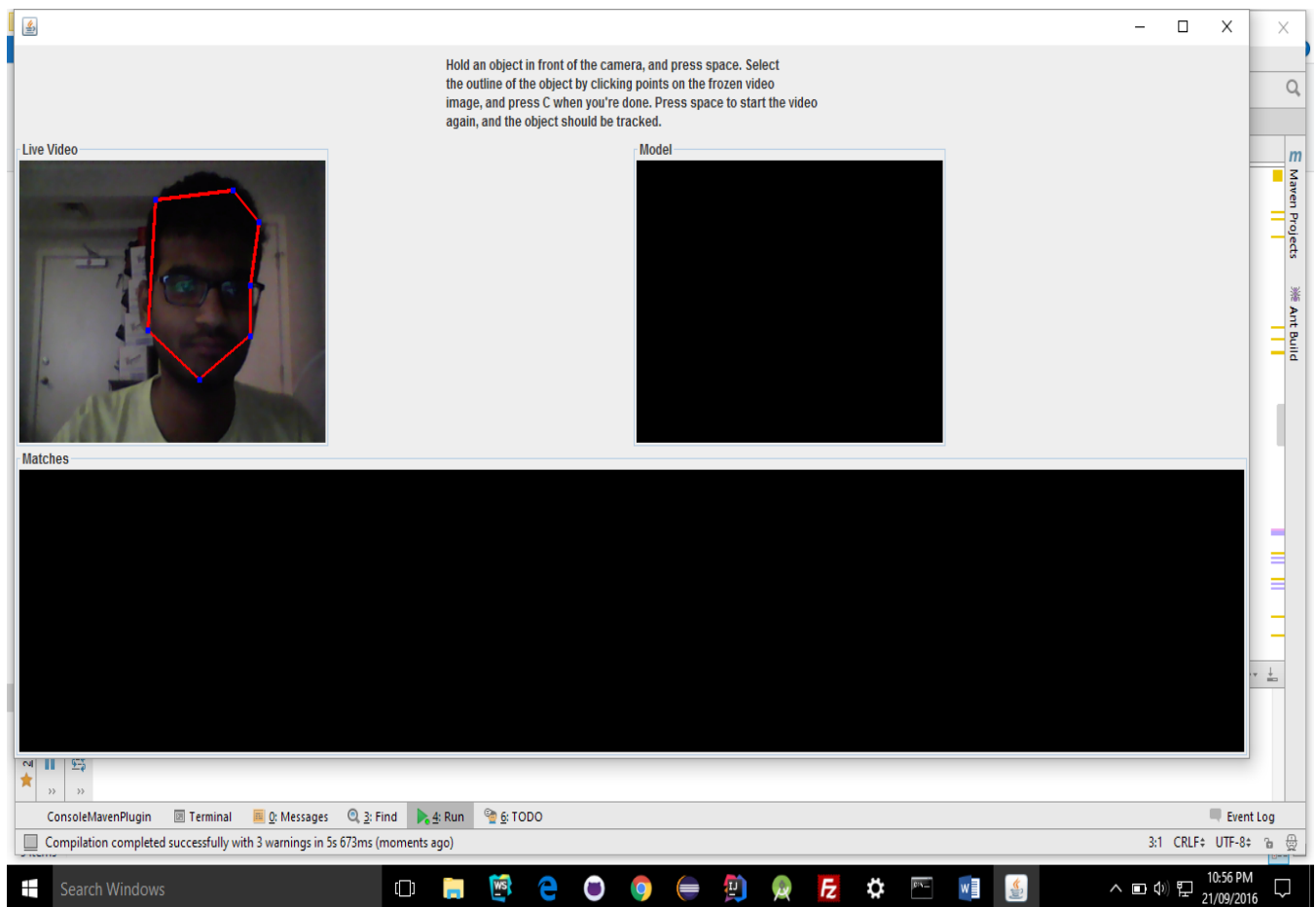
```
bytes result sent to driver
15/10/08 09:21:50 INFO TaskSetManager: Finished task 0.0 in stage 137.0 (TID 181
in 41 ms on localhost (1/1)
15/10/08 09:21:50 INFO TaskSchedulerImpl: Removed TaskSet 137.0, whose tasks hav
all completed, from pool
15/10/08 09:21:50 INFO DAGScheduler: Stage 137 (print at KafkaWordCount.scala:27
finished in 0.037 s
-----
Time: 1444321310000 ms
-----
(4,307359)
(8,307165)
(6,307435)
(0,306762)
(2,305377)
(7,306598)
(5,306727)
(9,307420)
(3,306981)
(1,306376)

15/10/08 09:21:50 INFO DAGScheduler: Job 35 finished: print at KafkaWordCount.sc
a:27, took 0.054576 s
15/10/08 09:21:50 INFO JobScheduler: Finished job streaming job 1444321310000 ms
```

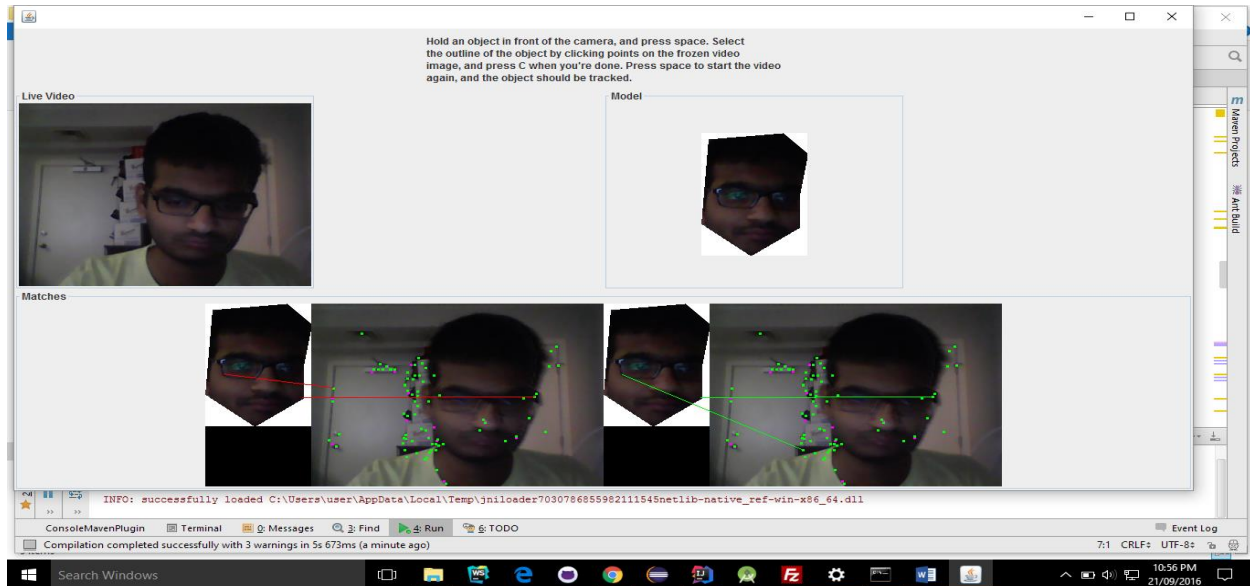
IMPLEMENTATION

Screenshots

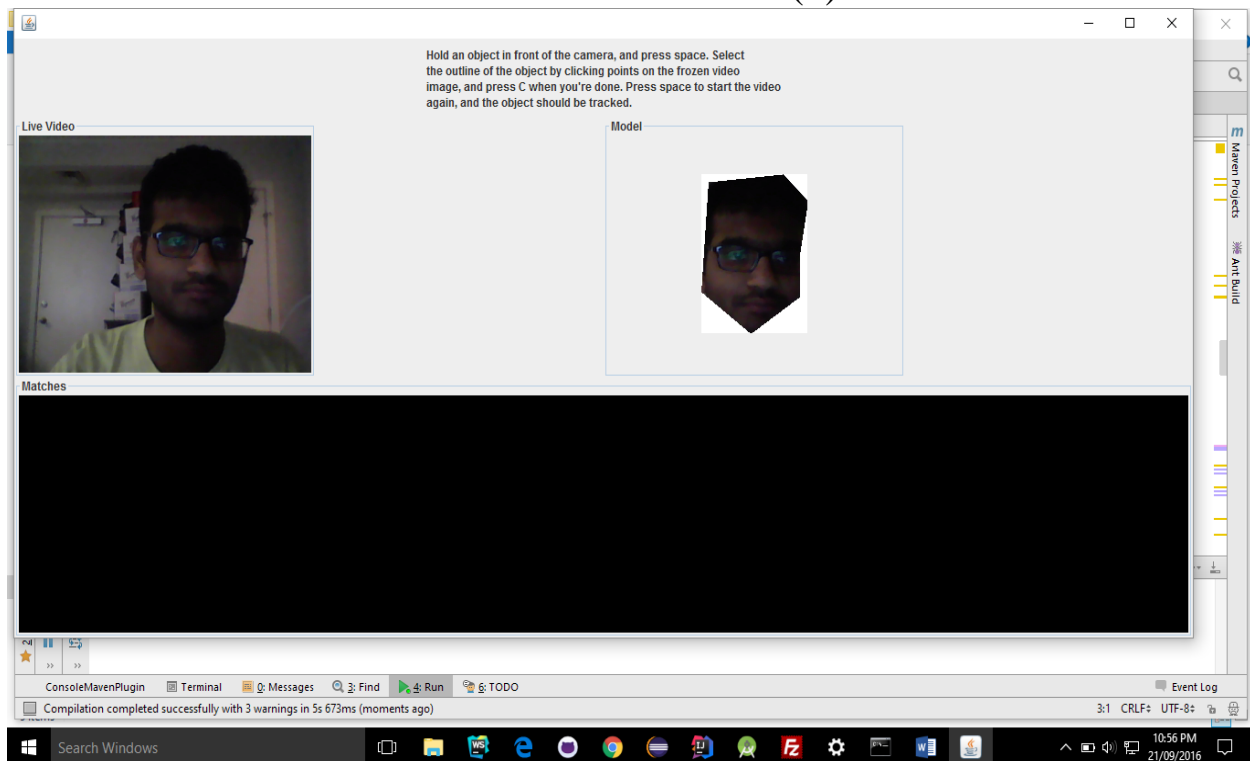
- User Interface



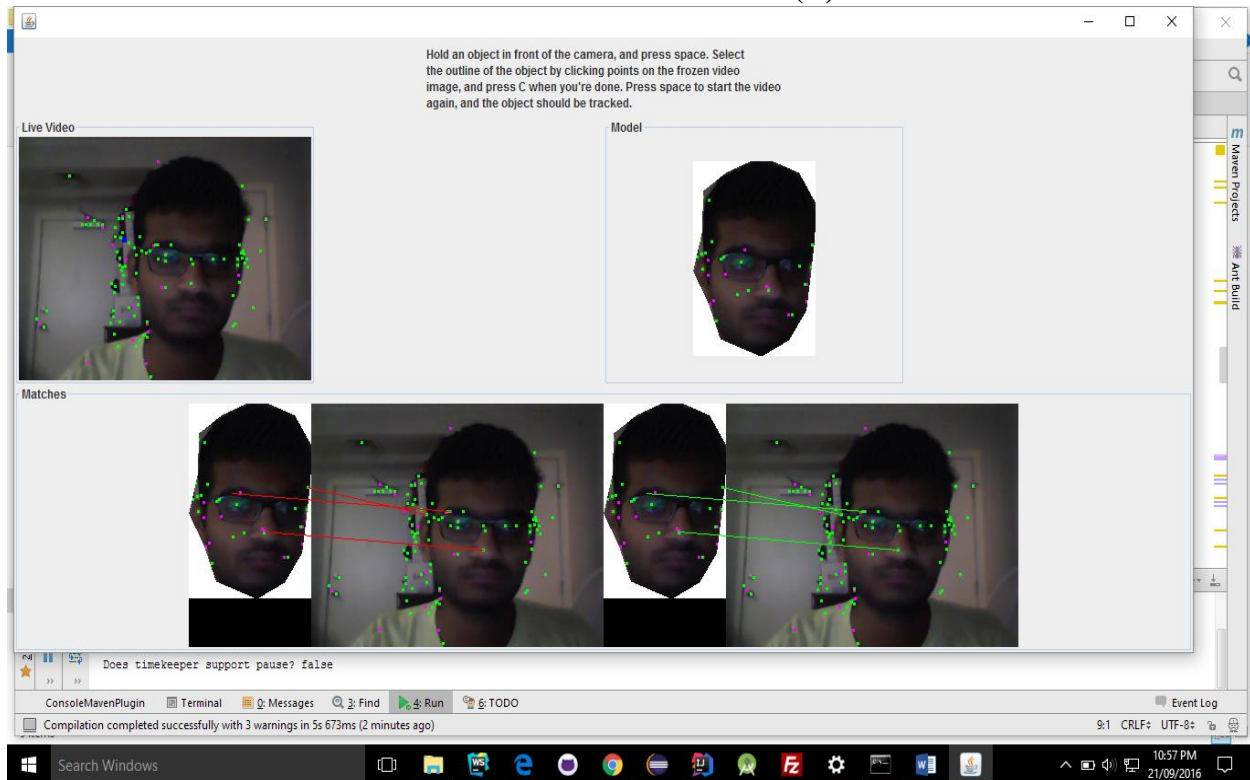
- Image Selection for Comparison



- SIFT Feature extraction for two models (1)



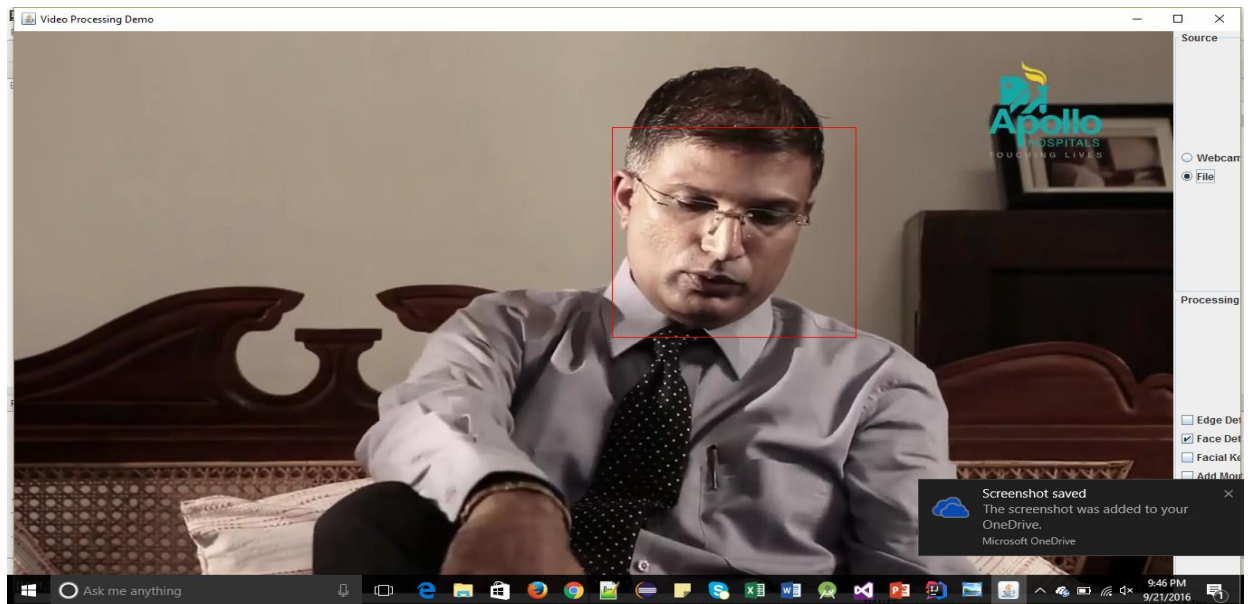
- SIFT Feature extraction for two models (2)



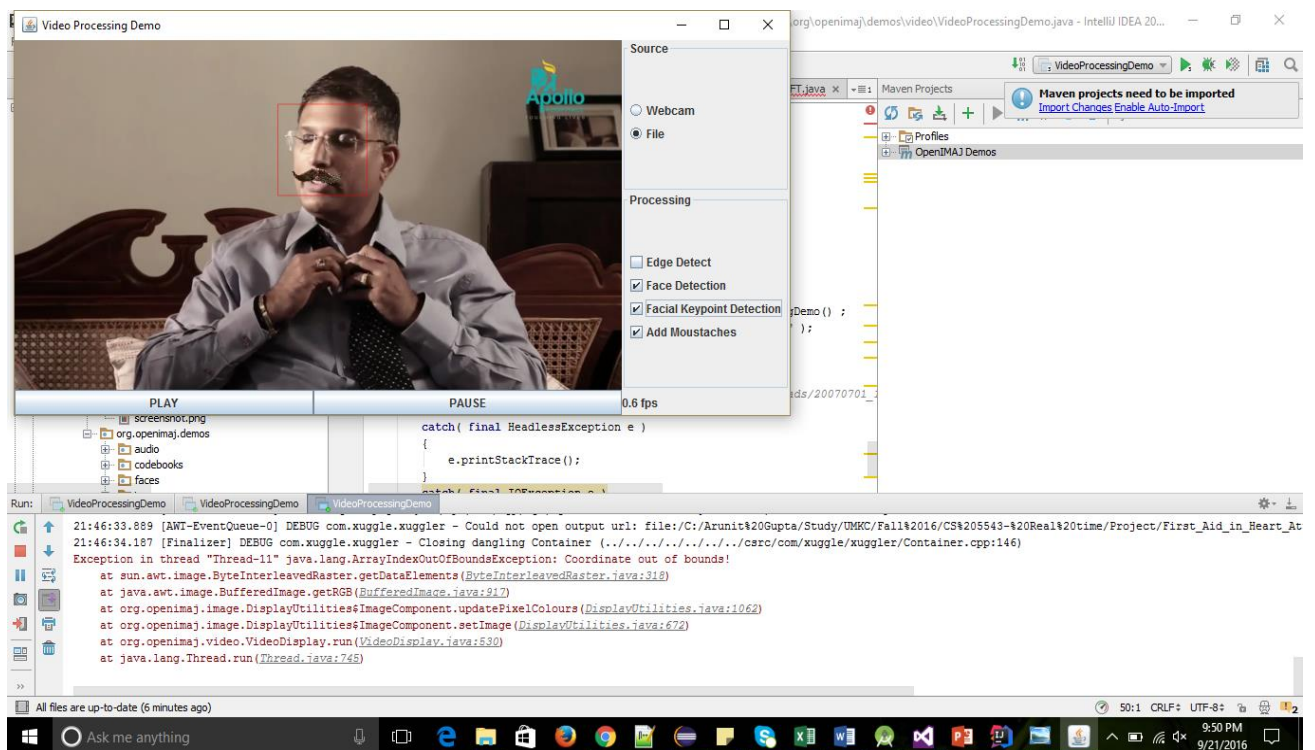
- Face Detection Implementation

The screenshot shows an IDE (IntelliJ IDEA) running a video processing application. The application window, titled "Video Processing Demo", displays a video feed from a webcam. A red bounding box is drawn around a person's face in the video. The "Processing" menu is open, showing options: "Edge Detect", "Face Detection" (checked), "Facial Keypoint Detection", and "Add Moustaches". The video feed shows coordinates [284,3] [127,0,133,0,118,0] and a frame rate of 6.8 fps. The IDE shows the source code for VideoProcessingDemo.java, which implements VideoDisplayListener. The run console shows several exceptions: "org.openimaj.video.capture.VideoCaptureException: Timed out waiting for next frame" and "java.lang.RuntimeException: org.openimaj.video.capture.VideoCaptureException: Timed out waiting for next frame".

- Face Detection from Video



- Face Detection from Video and Features



Work in progress

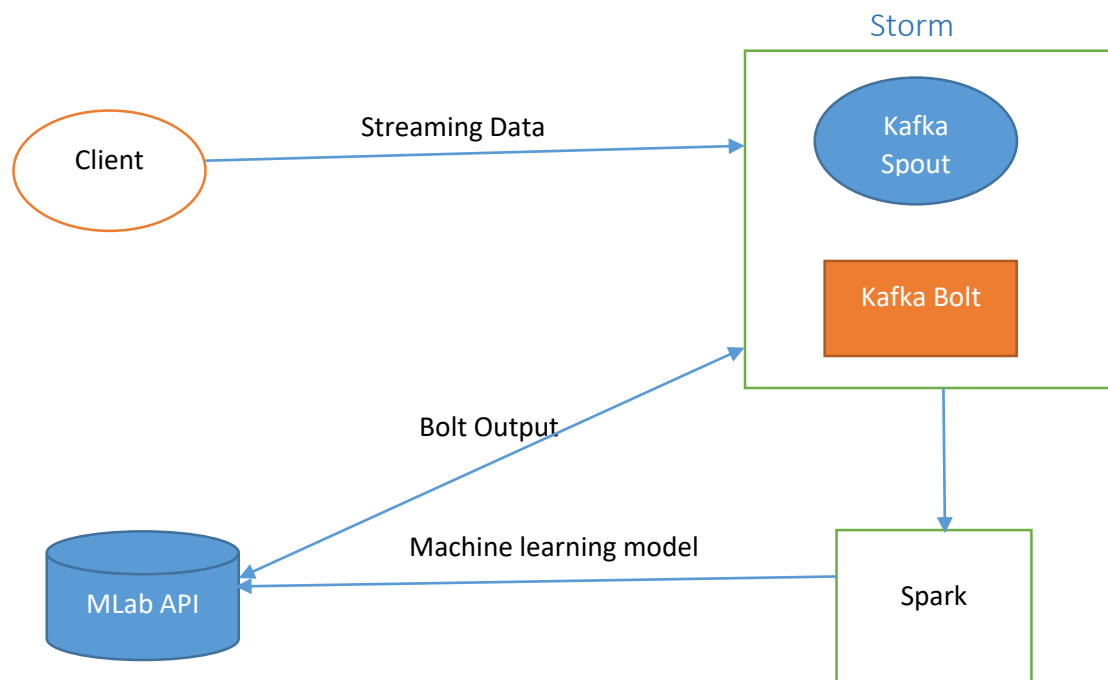
- Running Storm and Kafka
- Running Kafka Consumer
- Integration of Spark Kafka and Storm Kafka
- Implementation of Full Workflow
- Narrow down scope of Domain
- Streaming dataset on Cancer and ECG reports

Project Increment 2

Domain Redefined:

Narrowed down domain to Cancer cells in human body. This project will have ability to process X-ray reports of Brain, Lung and throat cancer, which could be processed by the machine to detect the type and stage of cancer. It has ability to scan through the series of X ray/MRI reports and analyze the pattern to detect the changes occurring in the reports.

We are using SIFT and HOG techniques to extract features and detect the type of cancer. Dimensions can be measured by using the same techniques to detect changes in the stage of cancer by analyzing the shrinkage or expansion of the cells.



Workflow Model

Data Set Available

Cancer Imaging Archive

<http://www.cancerimagingarchive.net/>

TCIA Collections

The image data in The Cancer Imaging Archive (TCIA) is organized into purpose-built collections of subjects. The subjects typically have a cancer type and/or anatomical site (lung, brain, etc.) in common. Each link in the table below contains information concerning the scientific value of a collection, information about how to obtain any supporting non-image data which may be available, and links to view or download the imaging data.

RIDER NEURO MRI

RIDER Neuro MRI contains imaging data on 19 patients with recurrent glioblastoma who underwent repeat imaging sets. These images were obtained approximately 2 days apart. All 19 patients had repeat dynamic contrast-enhanced MRI (DCE-MRI) datasets on the same 1.5T imaging magnet. On the basis of T2-weighted images, technologists chose 16 image locations using 5mm thick contiguous slices for the imaging. For T1 mapping, multi-flip 3D FLASH images were obtained using flip angles of 5, 10, 15, 20, 25 and 30 degrees, TR of 4.43 ms, TE of 2.1 ms, 2 signal averages. Dynamic images were obtained during the intravenous injection of 0.1mmol/kg of Magnevist intravenous at 3ccs/second, started 24 seconds after the scan had begun.

RIDER Lung CT

The RIDER Lung CT collection was constructed as part of a study to evaluate the variability of tumor unidimensional, bidimensional, and volumetric measurements on same-day repeat computed tomographic (CT) scans in patients with non-small cell lung cancer.

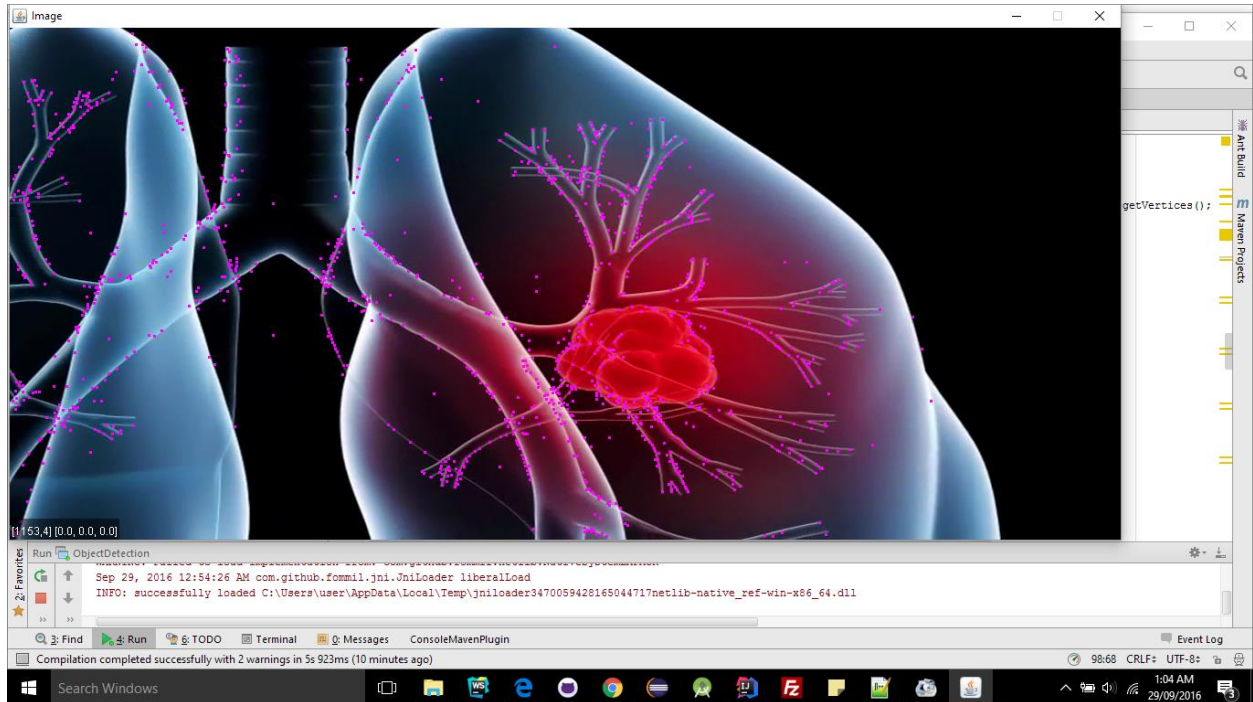
Thirty-two patients with non-small cell lung cancer, each of whom underwent two CT scans of the chest within 15 minutes by using the same imaging protocol, were included in this study. Three radiologists independently measured the two greatest diameters of each lesion on both scans and, during another session, measured the same tumors on the first scan.

Lung Phantom

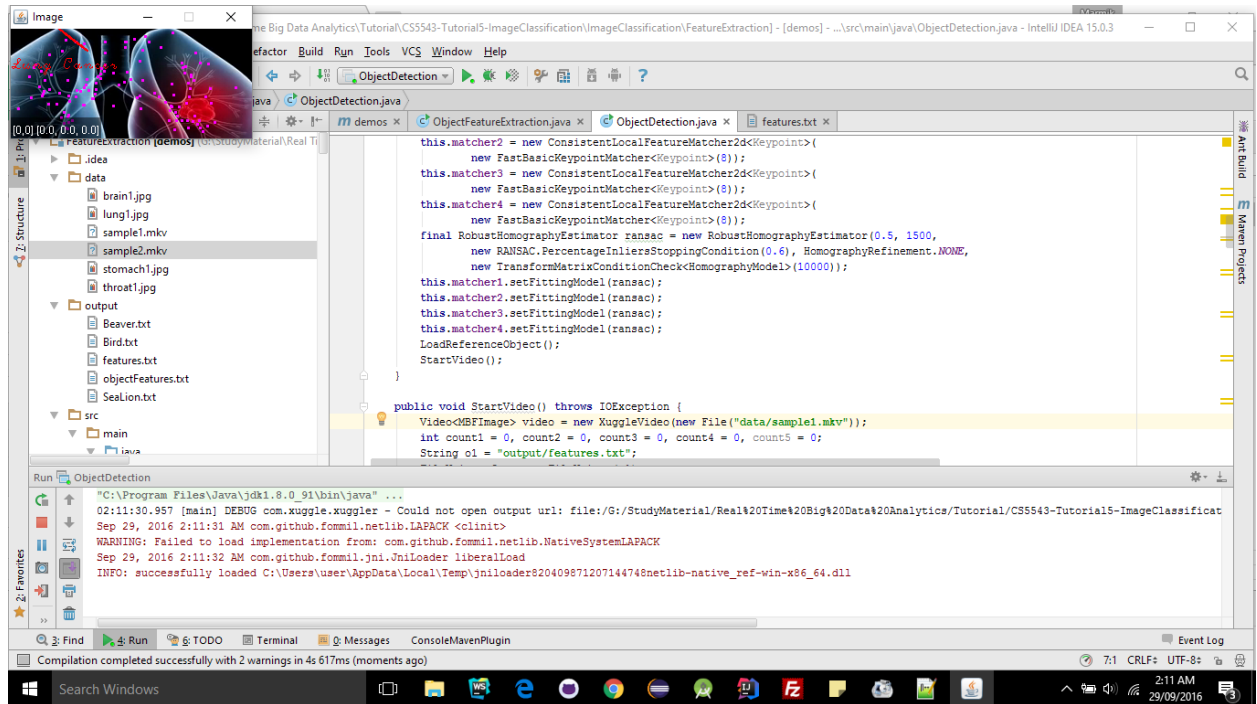
The FDA anthropomorphic thorax phantom with 12 phantom lesions of different sizes (10 and 20 mm in effective diameter), shapes (spherical, elliptical, lobulated, and spiculated), and densities (-630, -10, and +100 HU) was scanned at Columbia University Medical Center on a 64-detector row scanner (LightSpeed VCT, GE Healthcare, Milwaukee, WI). The CT scanning parameters were 120 kVp, 100 mAs, 64x0.625 collimation, and pitch of 1.375. The images were reconstructed with the lung kernel using 1.25 mm slice thickness.

Demonstration and Analysis:

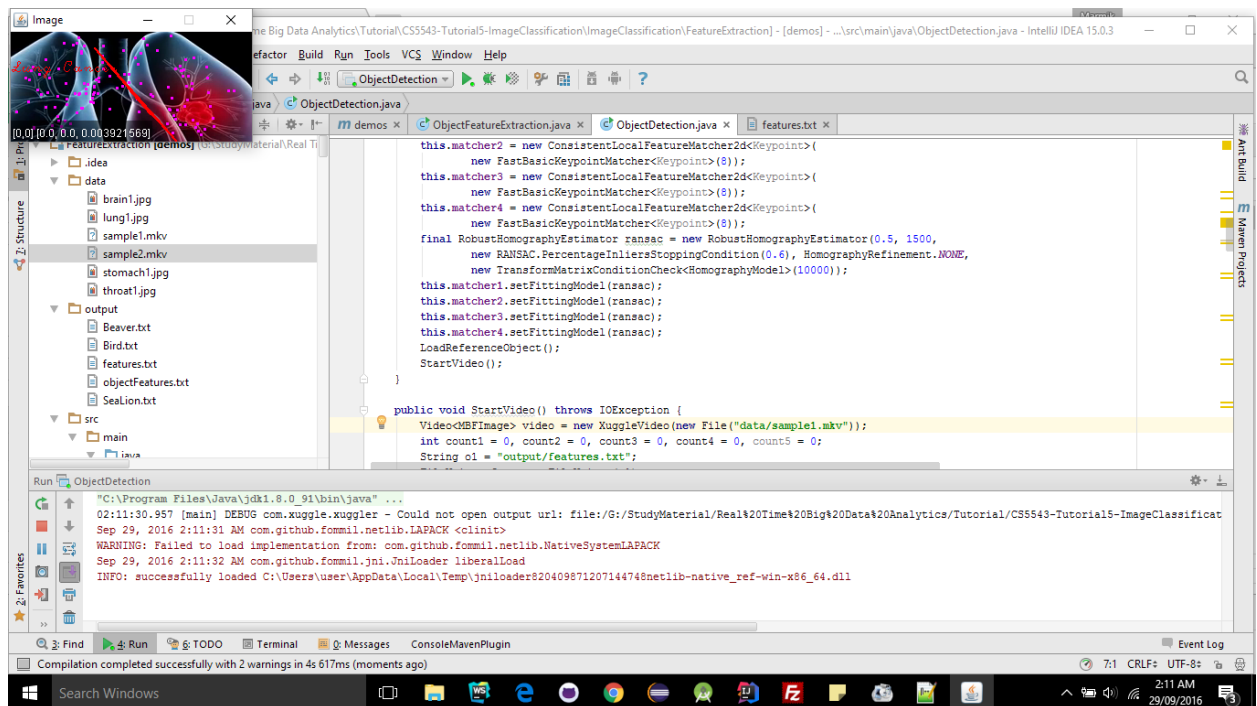
1) Feature detection for Lungs



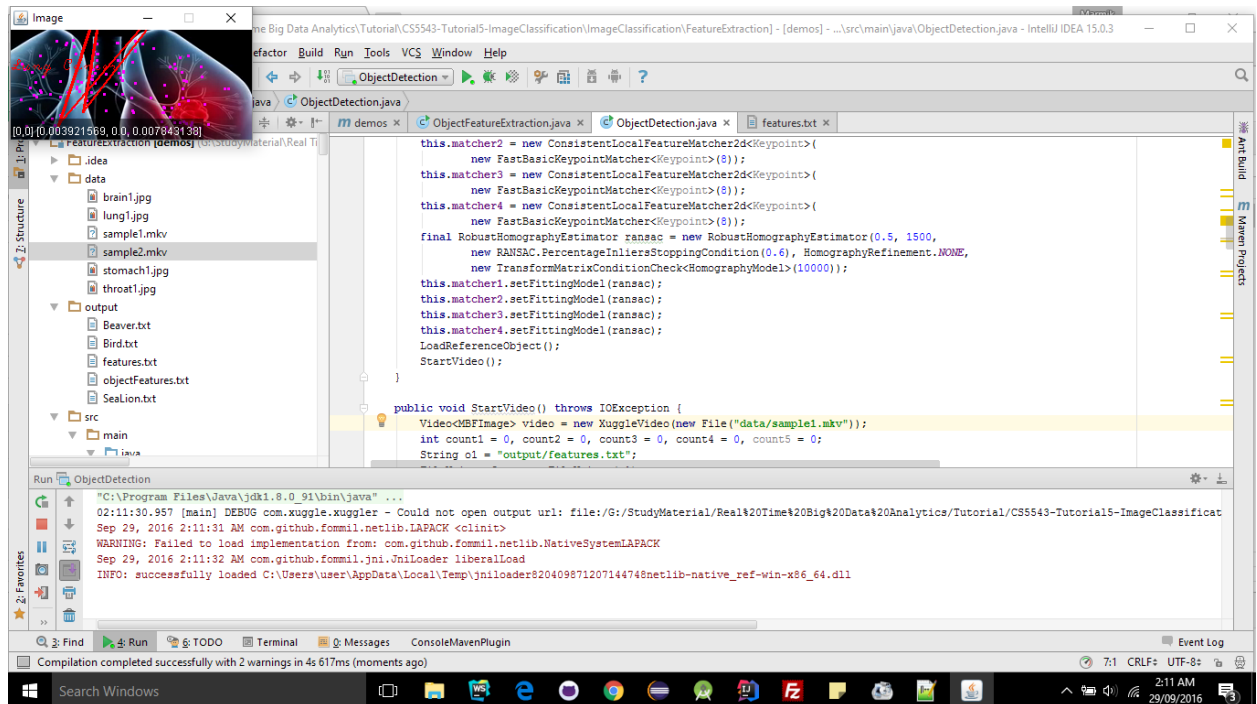
2) Annotation for Lung cancer



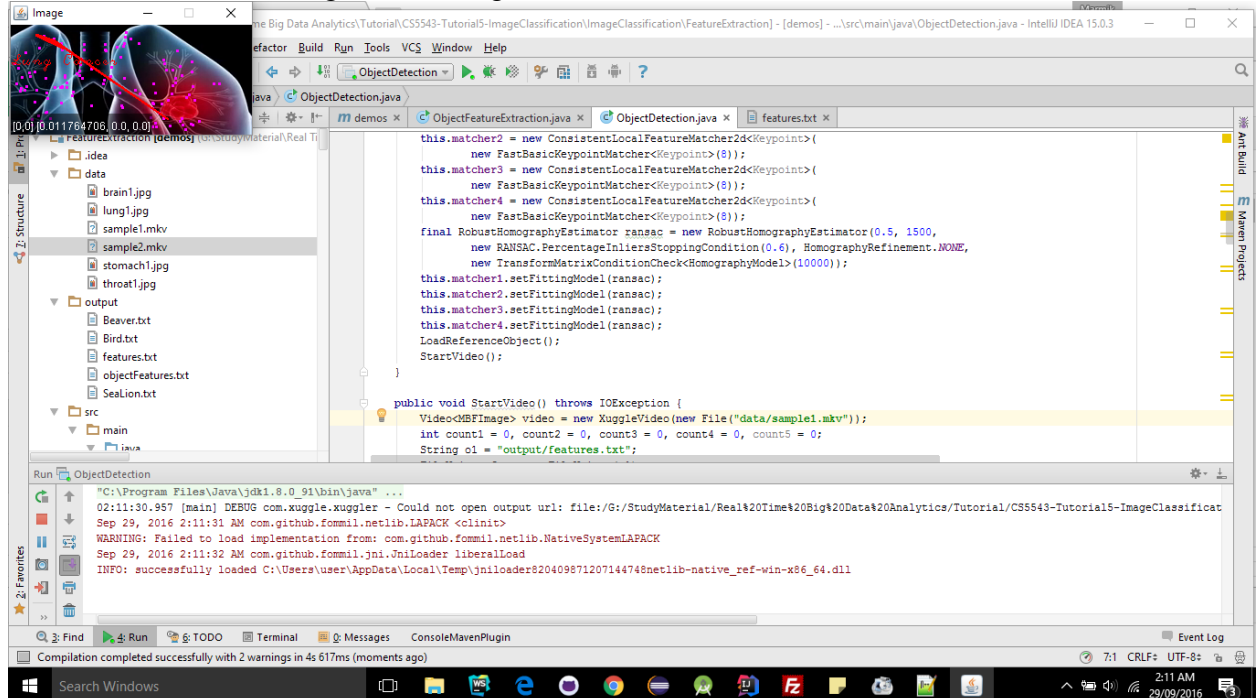
3) Feature Extraction of lung cancer

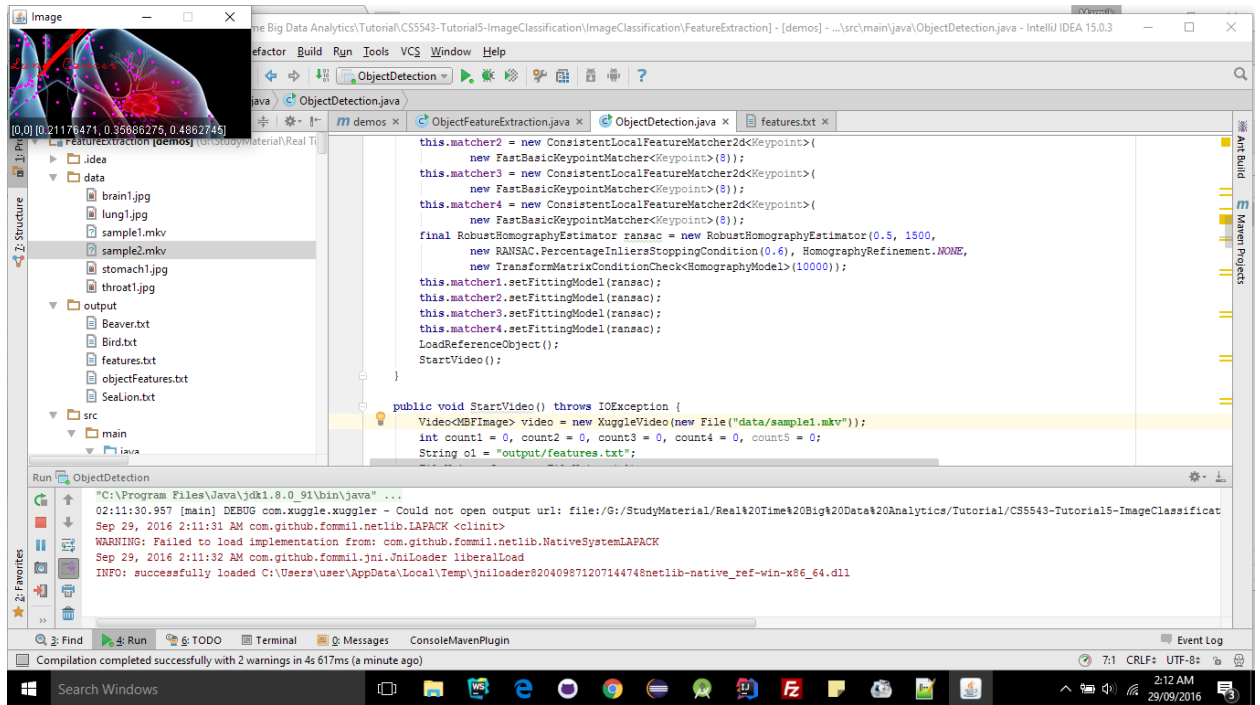


4) Bounding Box for Lung Cancer

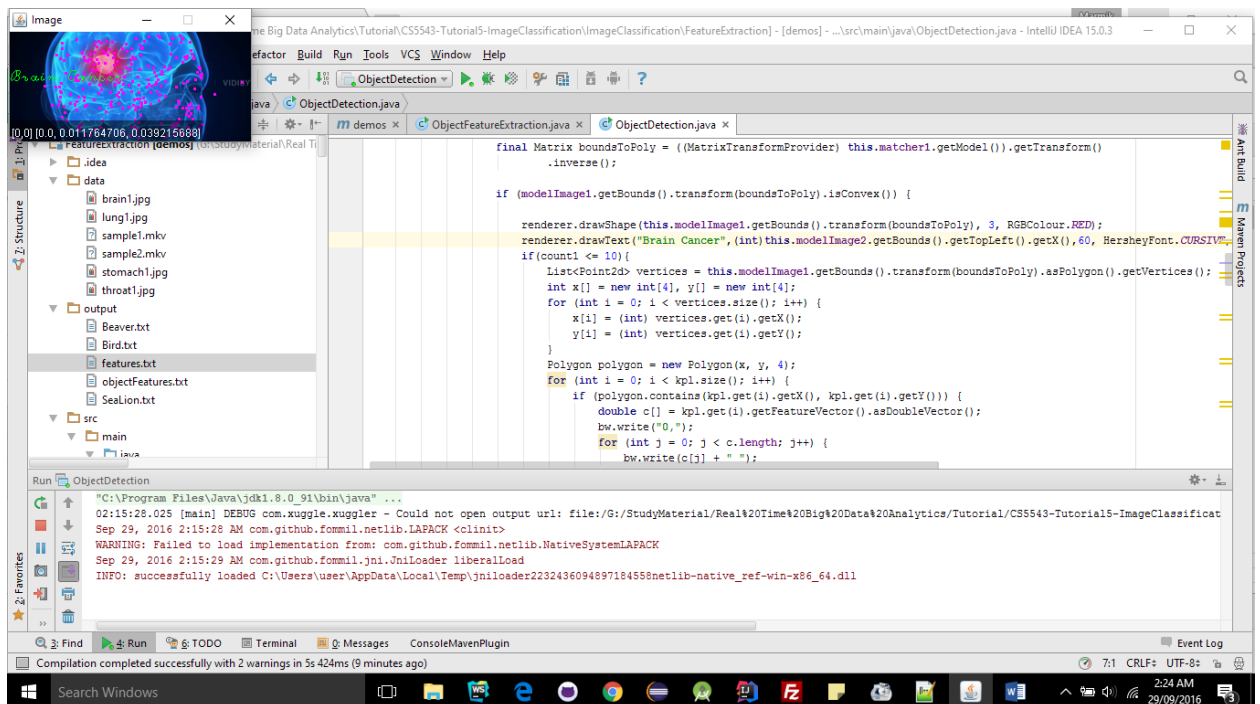


5) Annotation and Bounding box for lung cancer

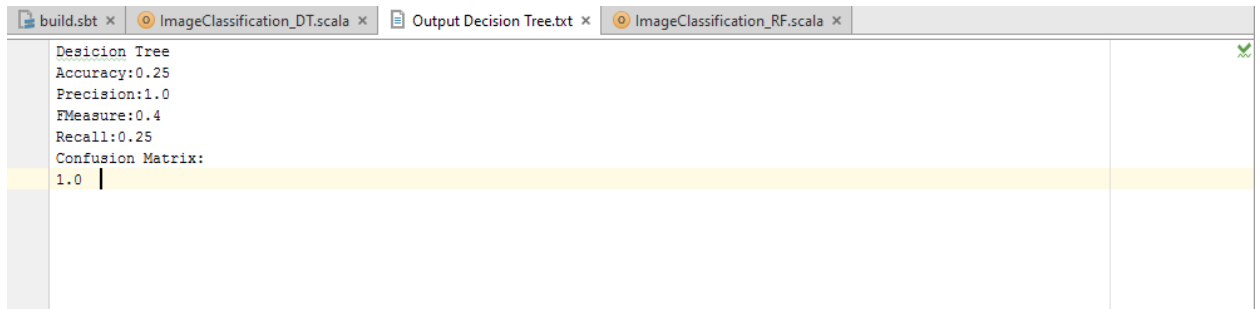




6) Brain Cancer (Annotation, feature extraction, bounding box)

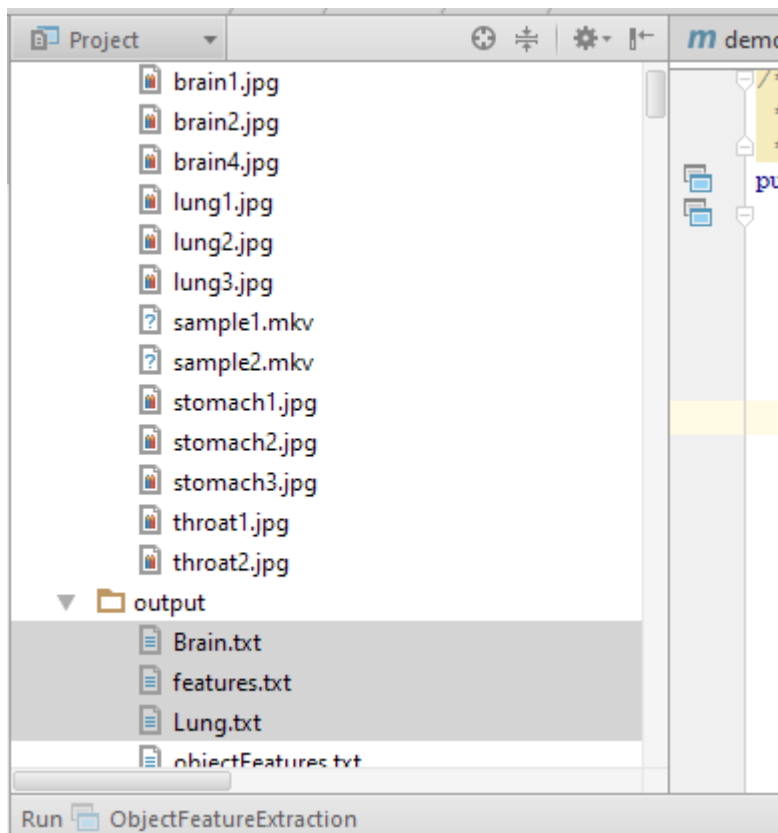


7) Decision tree Accuracy



```
Decision Tree
Accuracy:0.25
Precision:1.0
FMeasure:0.4
Recall:0.25
Confusion Matrix:
1.0
```

Folder Structure



Feature Extraction text

File size exceeds configured limit (2560000). Code insight features are not available.

```
1,-95.0 -127.0 -128.0 -128.0 -104.0 -111.0 -128.0 -128.0 13.0 -115.0 -128.0 -123.0 -118.0 -128.0 -128.0 -124.0 -91.0 -123.0 -128.0 -80.0
1,-58.0 -50.0 -128.0 -128.0 -122.0 -123.0 -128.0 -128.0 -41.0 -35.0 -121.0 -122.0 -118.0 -122.0 -122.0 -111.0 -89.0 -96.0 -91.0 -60.0
1,-94.0 -107.0 -128.0 -128.0 -128.0 -128.0 -128.0 -122.0 -6.0 -92.0 -128.0 -127.0 -128.0 -128.0 -128.0 -92.0 -50.0 -117.0 -120.0 -52.0
1,-123.0 -112.0 -126.0 -128.0 -128.0 -113.0 -101.0 -118.0 -9.0 -101.0 -128.0 -128.0 -128.0 -128.0 -116.0 -64.0 -9.0 -49.0 -110.0 -126.0
1,-12.0 -104.0 -128.0 -128.0 -125.0 -128.0 -128.0 -128.0 -34.0 -112.0 -128.0 -127.0 -45.0 -115.0 -128.0 -128.0 -12.0 -104.0 -128.0 -126.0
1,-127.0 -112.0 -125.0 -128.0 -120.0 -17.0 -66.0 -127.0 -122.0 -49.0 -93.0 -128.0 -122.0 -83.0 -110.0 -114.0 -120.0 -63.0 -37.0 -127.0
1,-128.0 -128.0 -127.0 -126.0 -128.0 -128.0 -122.0 -101.0 -85.0 -123.0 -128.0 -128.0 -128.0 -128.0 -125.0 -9.0 -67.0 -103.0 -127.0 -128.0
1,-128.0 -62.0 -111.0 -128.0 -128.0 -111.0 -125.0 -128.0 -120.0 -88.0 -126.0 -128.0 -128.0 -56.0 -68.0 -123.0 -90.0 -127.0 -128.0 -128.0
1,-62.0 -124.0 -128.0 -124.0 -115.0 -128.0 -128.0 -126.0 -65.0 -125.0 -128.0 -120.0 -104.0 -126.0 -123.0 -104.0 -107.0 -128.0 -128.0 -128.0
1,-101.0 -78.0 -127.0 -128.0 -107.0 -116.0 -128.0 -128.0 -90.0 -70.0 -128.0 -128.0 -98.0 -109.0 -128.0 -128.0 -115.0 -114.0 -128.0 -128.0
1,-32.0 -63.0 -128.0 -128.0 -128.0 -128.0 -128.0 -128.0 -14.0 -37.0 -124.0 -126.0 -116.0 -118.0 -128.0 -127.0 -113.0 -110.0 -119.0 -97.0
1,-126.0 -128.0 -128.0 -126.0 -119.0 -128.0 -128.0 -128.0 32.0 -76.0 -128.0 -128.0 -128.0 -128.0 -128.0 -128.0 32.0 -60.0 -128.0 -128.0
1,-77.0 -37.0 -128.0 -128.0 -128.0 -128.0 -128.0 -128.0 4.0 -38.0 -128.0 -128.0 -128.0 -128.0 -128.0 -127.0 -13.0 -88.0 -110.0 -76.0 -128.0
1,-116.0 -128.0 -128.0 -128.0 -124.0 -87.0 -120.0 -122.0 -109.0 -128.0 -128.0 -128.0 -119.0 -55.0 -99.0 -108.0 42.0 -128.0 -128.0 -128.0
1,-10.0 -105.0 -128.0 -128.0 -128.0 -128.0 -128.0 -117.0 -10.0 -121.0 -120.0 -124.0 -120.0 -128.0 -128.0 -88.0 -114.0 -125.0 -121.0 -118.0
1,-98.0 -128.0 -128.0 -128.0 -128.0 -128.0 -128.0 -127.0 21.0 -128.0 -128.0 -128.0 -128.0 -128.0 -127.0 -85.0 -90.0 -128.0 -128.0 -98.0
1,-117.0 3.0 -124.0 -128.0 -128.0 -128.0 -128.0 -128.0 -77.0 3.0 -128.0 -128.0 -112.0 -47.0 -126.0 -128.0 -123.0 -121.0 -128.0 -127.0
1,-94.0 -126.0 -128.0 -128.0 -128.0 -128.0 -128.0 -126.0 -15.0 -112.0 -128.0 -128.0 -128.0 -128.0 -128.0 -113.0 8.0 -111.0 -128.0 -125.0
1,-125.0 -127.0 -123.0 -125.0 -125.0 -122.0 -126.0 -50.0 -93.0 -127.0 -127.0 -128.0 -128.0 -128.0 -127.0 21.0 -68.0 -116.0 -116.0 -101.0
```

References

- <http://www.michael-noll.com/blog/2014/10/01/kafka-spark-streaming-integration-example-tutorial/>
- <https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/KafkaWordCount.scala>
- https://www.tutorialspoint.com/apache_kafka/apache_kafka_integration_spark.htm
- <http://spark.apache.org/docs/latest/streaming-kafka-integration.html>
- <http://kafka.apache.org/documentation.html>
- <https://kafka.apache.org/090/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html>
- <http://spark.apache.org/docs/latest/streaming-kafka-integration.html>
- https://www.tutorialspoint.com/apache_kafka/apache_kafka_integration_storm.htm
- http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- <http://www.vlfeat.org/overview/hog.html>