Distributed and Scalable Video Analysis Architecture for Human Activity Recognition Using Cloud Services

Cody Wilson Eilar
University of New Mexico

Cody Wilson Eilar

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Marios Pattichis , Chairperson

Dr. Ramiro Jordan

Dr. Wennie Shu

Distributed and Scalable Video Analysis Architecture for Human Activity Recognition Using Cloud Services

by

Cody Wilson Eilar

B.S., University of New Mexico, 2010

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2016

Dedication

To my family and friends who have supported my academic adventures and have inspired me to stay curious.

Acknowledgments

I would like to thank my advisor, Professor Marios Pattichis, for his continuous support in reviewing my source code and for helping me shape my ideas for this thesis. I would also like to thank Elmyra Grelle for helping me stay on track with all the necessary requirements to graduate. Finally, I would like to thank Sandia National Laboratories for supporting me financially in my academic endeavors.

Distributed and Scalable Video Analysis Architecture for Human Activity Recognition Using Cloud Services

by

Cody Wilson Eilar

B.S., University of New Mexico, 2010

M.S., Computer Engineering, University of New Mexico, 2016

Abstract

This thesis proposes an open-source, maintainable system for detecting human activity in large video datasets using scalable hardware architectures. The system is validated by detecting writing and typing activities that were collected as part of the Advancing Out of School Learning in Mathematics and Engineering (AOLME) project. The implementation of the system using Amazon Web Services (AWS) is shown to be both horizontally and vertically scalable. The software associated with the system was designed to be robust so as to facilitate reproducibility and extensibility for future research.

Contents

List of Figures

List of Tables

Glossary

AAS	Amazon's Auto Scaling
AOLME	Advancing Out of School Learning in Mathematics and Engineering
AWS	Amazon Web Services
BoF	Bag of Features
CDF	Cumulative Distribution Function
CI	Continuous Integration
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Value
Docker	Application that enables easy containerization of software packages
EC2	Amazon's Elastic Computer Cloud
ECS	Amazon's EC2 Container Service
Farneback Method	Dense optical flow method
FPGA	Field Programmable Gate Array

Glossary

Git	Application for version controlling software
GPU	Graphics Processing Unit
Hadoop	Software package for running map-reduce jobs
HDF5	Hierarchical Data Format
ivPCL	Image and Video Processing and Communications Lab
KNN	K-nearest Neighbors
Lucas-Kanade Method	Sparse representation of optical flow
OpenCL	Open Computing Language
OpenCV	Open Computer Vision software package
PCA	Principle Component Analysis
REST	Representational State Transfer
S3	Amazon's Simple Storage Service
SDK	Software Development Kit
Spark	Software package for running map-reduce jobs
SQS	Amazon's Simple Queue Service
Storm	Software package for running map-reduce jobs
SVM	Support Vector Machine
TAPI	OpenCV's Transparent API
TDD	Test Driven Development

Glossary

Travis-CI	Software service that enables continuous integration of code
URI	Uniform Resource Identifier

Chapter 1

Introduction

There is strong interest in the development of distributed video analysis systems that can be used to analyze large video databases. Unfortunately, the overwhelming majority of software packages for automated video analysis, are not necessarily designed to scale in order to handle processing on vast video databases.

An example of a large-scale video database is provided by the advancing out of school learning in mathematics and engineering (AOLME) project. AOLME contains over a thousand hours of high quality video data that need to be analyzed so as to understand how middle school students acquire basic programming skills. Currently, most of this analysis is done manually [38] to extract pertinent features for researchers to analyze.

Manual video annotation and transcription is extremely tedious and unsustainable for large datasets. Because of these inhibitory factors, most of these encoded videos are left untouched and unanalyzed, potentially leaving thousands of hours of valuable information about the learning process unexplored and underutilized. Clearly there is a need for a tool to aid researchers in properly analyzing these video datasets efficiently.

1.1 Motivation

Current methods in video analysis systems are extremely application dependent and are inadequate computationally to sufficiently investigate video datasets at such a large scale. As such, there is a propensity for a system that is accurate, scalable and flexible in nature to handle a variety of challenges in automated video analysis.

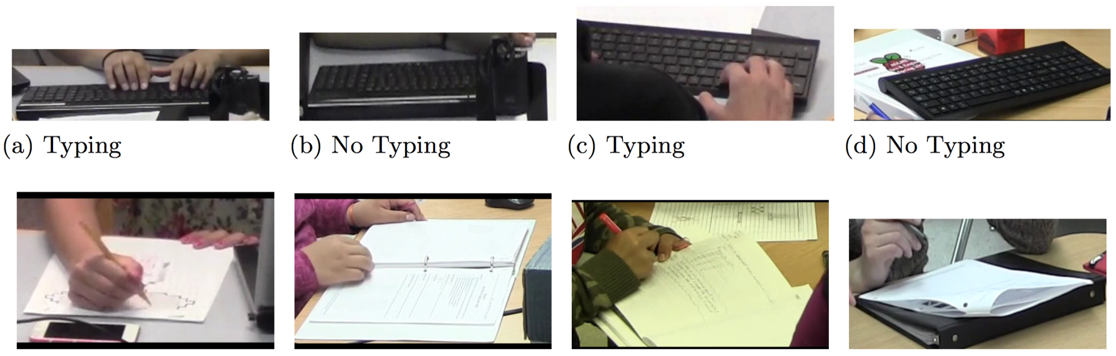
Computationally, there is clearly a need for video analysis methods that can be efficiently implemented in heterogenous compute hardware (such as GPUS and CPUS), and have said hardware function in a distributed environment. Being able to leverage heterogenous computer hardware greatly increases the efficiency and speed of certain, heavily used, video processing algorithms such as 2D convolutions. Furthermore, having this system exist in a distributed environment will greatly speed up ephemeral operations and makes it possible to scale up to address large scale problems. Thus this thesis is motivated by the challenges associated with analyzing large scale video databases.

The human activities that will be considered as part of this thesis are illustrated in Figure 1.1. The thesis explores the problem of classifying typing versus no-typing and writing versus no-writing using optical flow methods.

The proposed research represents a novel extension of prior research undertaken at the image and video processing and communications laboratory. Prior efforts focused on the development AM-FM representations [43] [13] [3] [36] [1] [37] [42] [2] [41] [46] [47] [54] [50] [34] [33] [48] [49] and the development of dynamically reconfigurable architectures [12] [35] [27] [45] [26] [25] [24].

1.2 Thesis Statement

The thesis of this research is that it is possible to scale, accurately classify and process videos using a carefully designed video analysis system for human activity recognition. The basic idea is to distribute video segments and computational tasks among compute nodes so as to enable scalable computation. The focus of the thesis is to identify computationally intensive feature extraction methods that can be pre-computed and have their reduced feature space processed by the master node. Furthermore, the system should be highly scalable so as to support future extensions.



(a) Typing (b) No Typing (c) Typing (d) No Typing

(e) Writing (f) No Writing (g) Writing (h) No Writing

Figure 1.1: Example of features that have been manually extracted from the dataset for training and testing. For the above example, we need to classifiers for each activity to determine if the activity is being performed, or it is not.

1.3 Contributions

This thesis contributes to the computer engineering community by providing both algorithms and an architecture for efficient, scalable and rapid processing of extremely large video datasets in a cloud environment. Additionally, all the code written for this thesis is distributed under the MIT open source license so that the software can be used freely in the community and will thus facilitate reproducibility and extensibility in this field of research.

1.4 Summary

In this thesis, we show that it is possible to reduce the feature set of videos on the order of Megabytes down to tens of Kilobytes, and then accurately classify those features at very high accuracy as a particular human activity. We also create an architecture that is capable of scaling to dozens of compute notes, and potentially hundreds thus making the heavy lifting operations, such as computing optical flow vectors, a trivial task that can be efficiently performed in the AWS cloud, thus enabling researchers to extract germane features from videos in a matter of minutes instead of hours.

Chapter 2

Background

Human activity classification in videos is not only a difficult problem to solve algorithmically but also pushes current computing solutions to the edge of their capability, especially when attempting to keep up with real-time video rates. Many solutions have been proposed for robust activity classification in videos [44] [7] [51] [28]. Methods such as [7] rely on principal component analysis (PCA) and hidden Markov models (HMMs) to attempt to classify motions using trajectories; however, all the datasets study consist of hardware augmentation (a motion tracking device for example), used to track motions of hands etc, and do not rely solely on the video source for classification. Our method requires no other augmentation other than the videos. Other methods such as [44], use similar techniques that are done in this thesis, however they do not address the computational burden of producing a histogram of features to use for classification. We extend parts of their idea and create a distributed that scales both horizontally and vertically. Finally, new deep learning methods such as convolutional neural networks are also gaining popularity in activity classification in videos because there is no need to select the type of features *a-priori*, you let the network select what is important for classification [28]. This technique is computationally expensive, requires huge amounts of data to properly classify activities [28] and also lacks insight into what is or is not functioning properly in the network, essentially rendering the CNN as a black box. The study presented by Laptev et al. [32] showed that it is possible to reduce a video dataset size significantly using optical flow and a technique called bag of features (BoF) which is ultimately used by a machine learning algorithm, such as non-linear support vector machines to classify the activities. The BoF technique, however, lacks scalability and proposes no solution to this problem. Indeed we use a similar method to reduce the feature set, but we augment the bag of features and propose solution that is also scalable.

Much of the work that has been done in the area of activity recognition and video analysis in the cloud is found in Table 2.1. The table lists several studies that are used as the literature review for this thesis. Many of these papers present exceptional work in both human activity classification as well as video analysis using distributed systems. However, none of them fully address the problem presented in this thesis. From Table 2.1, it is evident that human activity classification as well as distributed video analysis is an unsolved problem and many researchers continue to publish in this area. Further more, it is obvious that many of these studies either address activity classification or distributed video systems, but none effectively combine both ideas.

Study	**Features**	**Classification**	**Comments**
Human activity recognition [58]	edge trajectories, optical flow histograms, Fisher Vector	multi-class SVM	Accurate classif. of various human activities using edge trajectories in combination with optical flow
Large-scale video analysis [59]	N/A	N/A	Using google cloud to analyze videos in a secure and robust way
Video segmentation and activity recognition [31]	Fisher Vectors, structured temporal models and Gaussian mixed models	SVM and PCA	Classif. of activities as well as attempting to automatically parse out activities in the videos
Action recognition [10]	Dynamic trajectory and static deep features	Linear SVM	Combining deep learning techniques with trajectory features.
Cloud resource management [29]	N/A	N/A	Scaling resources effectively on the cloud to minimize cost and maximize performance.
Large-scale video classif. [28]	N/A	Convolution Neural Networks	1 million youtube videos with 487 classes, not implemented on a distributed framework
Activity classif. [44]	spatio-temporal synchrony	synchrony autoencoder	Quick motion estimation using local features and high classif. rate.

Table 2.1: Latest work in video analysis in the cloud and human activity classification

In addition to reviewing the studies in Table 2.1, we also summarize many of the datasets used in this field of research in Table 2.2. This table illustrates some of the common datasets that are used in benchmarking how well a given classification algorithm works in comparison with other methods. Our dataset for AOLME is much bigger and is considered to be truly a dataset that is "in the wild". Additionally, the common datasets that are currently being used do not include learning activities that we are interested in automatically analyzing and thus are not a good fit for this thesis. Though for this thesis we use a subset of our entire AOLME video database, we can in the future leverage the nearly terabyte sized dataset as soon as we begin manually classifying and segmenting the videos for validation of our methods.

Title	**Description**	**URL**
UCF101 [55]	A dataset of 101 human actions (13,320 videos)	`http://www.thumos.info/download.html`
KTH	Six types of human activity (2391 sequences)	`http://www.nada.kth.se/cvap/actions/`
Olympic [44]	16 olympic sports gathered from youtube	`http://vision.stanford.edu/Datasets/OlympicSports/`
Toy Assembly [57]	29 sequences of 2-3 minute long sequences of a human assembling a toy from five different bins	`http://www.cc.gatech.edu/~nvo9/sin/`
CMU-MMAC [56]	Database that contains multimodal measures of activities such as cooking and food preparation	`http://kitchen.cs.cmu.edu/main.php`
MPIICooking [52]	Database of 65 cooking activities (8.7GB of AVI formatted video), continuously recorded in a realistic setting	`http://tinyurl.com/nvcoh6w`

Table 2.2: Common datasets used for activity recognition

Thus we have defined some of the important research in this area and where it is lacking. In the following sections, we explore important algorithms that are foundational to the success of our classification and scalable architecture. In section 2.1.1 we review optical flow and how it can be used for classification and then in section 2.2 we explore the current state-of-the-art cloud computing techniques and how they can be utilized to create a flexible and scalable architecture.

2.1 Human Activity Classification in Videos

Human activity classification in video is an unsolved problem and has attracted a variety of different image and video processing techniques. One of the main contributing factors to its difficulty is the sheer size of the data, videos tend to be very large and pose a very interesting problem to the big data community. Another contributing factor in making videos difficult to classify is the scale and frequency in which activities appear in the videos. In other words, if the activity we are searching for is typing, then it is difficult to find that activity when the action could be taking up the entire frame or only tens of pixels. Furthermore, we must also account for the speed of the action. Depending on the frame rate, the actions in the video could appear to be slower in some scenarios than in others, therefore any algorithm must account for time-varying activities.

In current big data processing techniques, a method that is employed to reduce the data size to discover important features about data is known as map-reduce [17]. Many software packages have since been developed to implement the ideas in [17] such as Hadoop, Spark, Storm and many others. However, these frameworks assume that individual pieces of the large dataset are atomic and relatively small. For example, Amazon needs to process information about products that they sell. They can use a relatively small amount of information to represent this each product (especially compared to videos). As a result they are able to distribute these small amounts of data to thousands of clusters to build a product recommender system and associate every product to every other product to create similarities between products. All of this processing can easily be done with Hadoop[60] or Spark. However, passing messages around that contain video segments is not feasible with these systems and is not advised. Since video human activity classification would require such a message passing system, we innovate in this area by proposing a framework that is simple, robust and extremely fast using Amazon Web Services for human activity recognition in videos.

Encoding videos as succinct datasets that can easily be classified and are representative of the underlying features is another area that this thesis attempts to address. In *Learning to encode motion using spatio-temporal synchrony* [30] they attempt to address this need by detecting synchrony between frames in videos. They were robustly able to classify motions in datasets KTH, UCF sports, Hollywood2 and YUPENN in the low to mid 90s classification rate. However, training and testing the system varied between 2-3 minutes all the way up to days. In this thesis, we not only get very good classification rates, but because the system is horizontally scalable, we can train the system in a matter of minutes assuming relatively small video segments (videos on the order of 1-2 MB).

More modern approaches use various compound techniques in an attempt to solve human activity recognition such as [58]. In this paper the authors showed that they can classify many of the existing datasets very well using a combination of optical flow histograms, edge trajectories, and support vector machines for classification. In this thesis, we do not attempt to beat the results in this paper, we strive to provide a method for activity classification that can distributed on a cloud like infrastructure and that can be used in real time. Furthermore, the datasets that they use are not from videos in the wild. Our AOLME dataset contains thousands of hours of video that are not ideal for training and testing activity recognition algorithms. This makes tuning and tweaking the parameters for any algorithm, edge trajectories or not, very difficult. Like them though, we are using optical flow which is well studied and robust for video activity recognition. An other contribution of this thesis that is not provided in [58] is that we openly provide our software as open source so that other studies can be easily conducted.

2.1.1 Optical Flow

There are many varieties of algorithms that aid in the analysis of motions in videos. Farneback, Lucas-Kanade and Horn-Schunck [22] are all well studied and successful algorithms for this analysis. In this paper though, we have chosen to limit the scope to two optical flow methods that are available to a variety of languages. We explain the implementation of the Farneback optical flow algorithm [19], also known as dense optical flow, and the pyramidal, Lucas-Kanade approach to optical flow [9].

2.1.2 Optical Flow Methods

In this thesis we attempt to classify two types of activities in video, typing and writing. Since both of these activities involve motion, i.e. a change of apparent structure position from one video frame to the next, optical flow algorithms are a suitable tool for attempting to extract germane features from the video.

We use both Lucas-Kanade [39] and the Farneback [19] optical flow algorithms to attempt to extract important motion features from the AOLME videos. Both algorithms attempt to solve a common problem known as the *aperture problem*. The problem is defined by assuming that there have been small changes in both the x,

y and time components of a video scene. This is outlined in Equation 2.1 Equation 2.1

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.1}$$

where I is the image, x & y are the column row coordinates respectively, and t is the time between two adjacent image frames. Taking the Taylor series expansion of Equation 2.1 results in Equation 2.2

$$f_x u + f_y v + f_t = 0 \tag{2.2}$$

where

$$f_x = \frac{\partial f}{\partial x} \; ; \; f_y = \frac{\partial f}{\partial y} \qquad\qquad u = \frac{dx}{dt} \; ; \; v = \frac{dy}{dt} \tag{2.3}$$

Equation is 2.3 is known as the Optical Flow equation. The object is to determine what u and v are given that f_x and f_y are the image gradients and f_t is the time gradient. Since in Equation 2.2 we have only two unknowns, we cannot solve the system without additional constraints. This is known as the *aperture problem.* Both the Lucas-Kanade method and the Farneback method attempt to estimate this problem. Lucas-Kande attempts to reframe the problem such that we have an overdetermined system, solving it and then providing motion vectors for only features that move. The Farneback solution, on the other hand, argues that is possible to solve the *aperture problem* by first approximating each neighborhood of both frames with quadratic polynomials, and then estimating the displacement fields between the frames using polynomial expansion. Rather than providing only a few motion vectors, the Farneback method provides dense optical flow. That is to say that there is a motion vector for every pixel between the frames [19].

2.1.3 Lucas-Kanade Method

We leverage a common C++ library that has already implemented a specialized version of the general Lucas-Kanade algorithm which uses pyramids to solve the optical flow at different scales of motion [9]. This algorithm is provided freely in a C++ computer vision library known as OpenCV [23]. Essentially, this algorithm is much like the original paper published by Lucas and Kanade, but it solves the issue of large motions between frames and at different scales. By definition, the Lucas-Kande method assumes that the displacement of features in the image between two frames is small and roughly constant within a pixel neighborhood. This algorithm, then, by definition cannot handle large motions between frames, and hence the algorithm presented in [9] attempts to more robustly solve optical flow. The Lucas-Kanade method in OpenCV is often used with Shi-Tomasi [53] detection points to estimate what are good features to track in a frame. This allows the Lucas-Kanade algorithm to perform calculations on a sparse matrix rather than computing dense optical flow.

2.1.4 Farneback Method

Again, we leverage the C++ OpenCV library to calculate the motion vectors for Farneback optical flow. Unlike the previous method, the Farneback algorithm does not require any track points to estimate motion vectors because it is a dense optical flow algorithm, i.e. 100% of the pixels has an associated optical flow vector. The main idea behind calculating the motion vectors in this method, is to use polynomial expansion for a neighborhood of pixels [19] and then use that estimation to find a global translation between the two frames. This essentially aids with global background movement so that unique movements in the foreground can be accurately measured.

2.2 Cloud Computing

Cloud computing has become a common practice among scientists, engineers and business owners alike for solving computationally expensive problems at a relative low cost. Cloud computing offers services on demand rather than provisioning hardware and software ahead of time. This has opened the door to researchers who require thousands of computers but don't necessarily have the funds or access to a cluster of their own [6]. Cloud computing offers a model that is pay as you go, you only pay for what you use. This contrasts with the traditional way of solving computationally expensive problems by acquiring hardware and software ahead of time to run on dedicated machines. This has the advantage that the owner of the hardware has full control of its resources, but it has the disadvantage of costing a significant amount of money and there is a chance that the originally allocated hardware is either insufficient for the task or is overdone, therefore wasting money and computational resources. At its core, information technology resources are now a programmable resource in the cloud, rather than one that has to be manually setup and configured.

The programming model for cloud computing is fundamentally different than traditional software models. In the traditional infrastructure of computing, applications ran on servers and often times relied on the state of a particular server to perform computations [5]. That model must be broken in order to efficiently develop applications for the cloud. Clearly, not all applications can be stateless, but the keys is to eliminate as many stateful components as possible and replace them with stateless ones. When this paradigm is followed, along with designing software with well defined interfaces, it makes creating a scalable, flexible application easy using cloud service providers [5] and how to make auto deployment a snap for rapid testing and distribution using container technologies such as docker. An example of a well defined interface is a representational state transfer (RESTful) one, which is an interface that provides easy interoperability between compute nodes on a network. Typically an interface like this communicates over http or https.

In the following subsections we explore the current state of cloud computing and what types of applications can leverage a cloud like infrastructure, a high level overview AWS and a few of the services that can benefit a video processing application.

2.2.1 What's in the cloud?

In most of the modern cloud architectures such as Amazon Web Services (AWS), Microsoft's Azure and Google's Cloud Platform, services offered range from compute power to database solutions. However, the resources that are often most important are access to compute resources, highly available networks and redundant and durable storage. With these three resources available, its possible to create almost any kind of application in a cloud architecture. The idea is to eliminate servers, and instead replace them services. Buzzwords that emerge from this context are software as a service (SaaS), hardware as a service (HaaS), and X as a service (XaaS) where X can refer to any number common IT processes.

The compute resources that are provided are simply virtual machines that can be created programmatically, usually through some kind of application program interface (API) provided by the cloud company. These virtual machines can usually be provisioned with almost any kind of operating system that is supported by the cloud company of choice. As a result, developers can easily stand up thousands of virtual instances in the cloud with their operating system of choice with the click of a button, and tear them down just as quickly.

Storage resources are also crucial to effective cloud software development. Making storage resources highly available has the advantage of there being no single point of failure in the data and also facilitates fast access to the data because no one network interface will be throttled on the network. In a distributed computing sense, this is extremely important so that data can be operated upon efficiently over potentially thousands of nodes.

Without a well provisioned network, this could be the ultimate bottleneck for a software package attempting to leverage the cloud. Fortunately, network as a service (NaaS) is also available in most commercially available clouds. Most providers will supply very basic networking for free, but then as with most resources available in the cloud, it is something that can be upgraded, provisioned and tailored to the developer's desire.

2.2.2 Amazon Web Services

Amazon Web Services (AWS) is the most mature cloud solution architecture available to the public. A popular streaming service in the United States, Netflix, recently moved all of its streaming services to AWS [61]. This is important to note because Netflix almost exclusively serves streaming video to hundreds of thousands of customers everyday. It's obvious from this move alone that there is a lot of potential in AWS for processing videos. Netflix cites many reason as to why they moved their monolithic video streaming app to hundreds of micro services that are run on the AWS cloud, but a few that truly stand out are "scalable computing and storage needs", "service availability", and "cost reduction" [61]. All of these reasons are enough for any academic institution to begin leveraging the power of micro services and cloud infrastructure.

AWS provides a number of services that are important to scaling our architecture both horizontally and vertically. Several services that are important to our video processing paradigm are:

- Elastic compute cloud (EC2) - These are virtual machines that can be provisioned quickly and can be configured a number of ways. They are the work horses for operating effectively on the AWS cloud. Furthermore, you can provision one, a thousand, or have them created as you begin to saturate existing instances.
- Simple storage service (S3) - This services provides storage that is enormous, durable and distributed over regions in the united states. Petabytes of data can be easily accessed from anywhere within an Amazon region and have the advantage of being redundant and fault tolerant. That is to say, the data is not simply stored on a hard drive at amazon, but is replicated over hundreds of machines and locations.
- Simple queue service (SQS) - Is a distributed queue that allows asynchronous reads and writes to any node in a cluster that has access. This service allows extremely reliable communication between micro services.
- Amazon Auto Scaling (AAS) - Auto scaling is a service that allows users to scale their EC2 instances automatically based on CPU and memory saturation. It is a key service that allows users to keep costs low yet perform at maximum capability.
- Elastic container service (ECS) - This services provides orchestration for automatic deployment of containers. This is an especially useful service if your application is deployed into a Docker container.

The list provided is by no means meant to be exhaustive, but rather provide a brief summary of the services that are important to creating an application that can be horizontally scaled on the cloud. AWS offers dozens of services, each one can be tailored to the specific needs of the developer and the application requirements.

EC2 instances are the work horses in the Amazon cloud. Not only are they important for horizontal scalability, but they are also important for vertical scalability. There are a number of instance types that can be chosen for customized applications needs. For example, instances can be provisioned with a single CPU of moderate frequency and several hundred MBs of random access memory (RAM) or they can be vertically scaled to contain state-of-the-art GPUs with dozens of CPU cores and hundreds of GBs of RAM. In fact, AWS is one of the few cloud services providers that does provide access to instances with GPUs, which is extremely important for video processing algorithms that leverage efficient GPU algorithms written in CUDA and/or OpenCL. Thus applications can be scaled vertically and horizontally using these EC2 instances.

2.2.3 Activity Recognition Using the Cloud

Until this point, we have only presented the basics of how the cloud can be used to perform computationally heavy tasks. In this subsection, we investigate how other research has leveraged the cloud to perform similar video processing tasks. One of the latest works is "A Cloud-Based Large-Scale Distributed Video Analysis System" [59]. In this paper, the authors attempt to make a robust, scalable and secure platform for performing video analysis using the Google cloud computing infrastructure. Their system proposes video analysis modules that are pluggable and therefore could perform a multitude of tasks and they also detail the data model for processing videos. However, their infrastructure lacks the continuous deployment and ease of deployment that the architecture in this thesis provides and they do not provide the software they wrote as a deliverable as we do. Many of the same concepts are shared between our thesis and this paper in terms of architecture, but they are not automatically orchestrating their services from software development to deployment. Our proposal not only supports a cloud like infrastructure for analyzing videos, but also proposes a delivery system that is not only convenient for the end users of our video analysis package, but is also convenient for developers wishing to plug their particular application into our architecture. Furthermore this thesis provides all the software openly to the public so that experiments can easily be redone and leveraged for future work.

2.2.4 Docker

Over the last two years, Docker has gained significant traction amongst software developers for application deployment. Docker is a software package that allows a variety of *nix type machines to run applications developed on different platforms to be run natively. For example, if an application was developed to run on Ubuntu 14.04, a user who has Docker installed on Cent OS 7 will be able to run that same byte code. This has a significant impact on the way that software is deployed. With Docker, gone are the days when software had to be compiled on multiple architectures or having users install hundreds of third party libraries just to run the release version of the software.

This paradigm has significant advantages for deploying scalable applications in a cloud infrastructure. The first advantage being that as long as a *nix machine is being run on an EC2 instance, you can deploy the package to the instance as fast as it takes to download the docker image. As a result, maintaining packages on the instance is not needed. All the software needed for the application is packaged into the docker image. Secondly, ECS makes it a trivial task to deploy your Docker container to thousands of instances with only a few commands. With ECS you can start and stop services, orchestrate varieties of instances and also perform load balancing to ensure that your application is being run at top performance and not costing exorbitant amounts of money.

Chapter 3

Methods

In this chapter we review the basic implementation details of our distributed video processing cloud architecture. First we cover the overall distributed architecture on the cloud. We delve into details of how all the compute nodes communicate with the primary node and how this can be done at any scale. Another step we take in this thesis, is to cover the development process and release process utilized for this open source package. The details of continuous integration and test driven development are important driving factors in developing reliable, scalable, and useable code. Then we cover how we use Lucas-Kanade and Farneback optical flow algorithms to reduce the feature space to only six 25 bin feature vectors. Finally, with the reduced feature space, we outline how to classify the videos using several well known machine learning algorithms such as K-nearest neighbors and support vector machines (SVM).

3.1 Scalable Architecture

A core principal for a well designed, horizontally scalable application, is to design it such that it does not contain state [5]. When state is required, the software complexity increases substantially and makes it difficult to distribute the system over a scalable amount of nodes. However, if the software was designed such that each service can operate and stand on its own, it is the perfect embarrassingly parallel computing task to tackle. For this thesis we focus on ensuring that our feature extractor, as described in Section 3.5, is completely stateless. This is a design feature that has allowed us the flexibility to scale our system over as many nodes as are available on the AWS cloud. A stateless architecture greatly increases the chances of any program to be turned into a micro service on the cloud. As a result, more and more parts of any application can be horizontally scaled across any number of compute nodes. So unless access to stateful services are necessary, such as database accesses, it is almost always best to design software stateless to create scalable services. This also has the added advantage

3.1.1 Architecture Overview

Our system builds upon AWS to create an easy-to-maintain and easy-to-scale video processing system. We use S3 storage to put small video clips that have been extracted from our AOLME dataset. These clips are made available to to all processing nodes. The processing nodes communicate with the master node using Amazon's simple queue service (SQS). Figure 3.1 illustrates the basic distributed system design and the psuedo-code for the entire system is shown in Figure 3.2 and Figure 3.3.

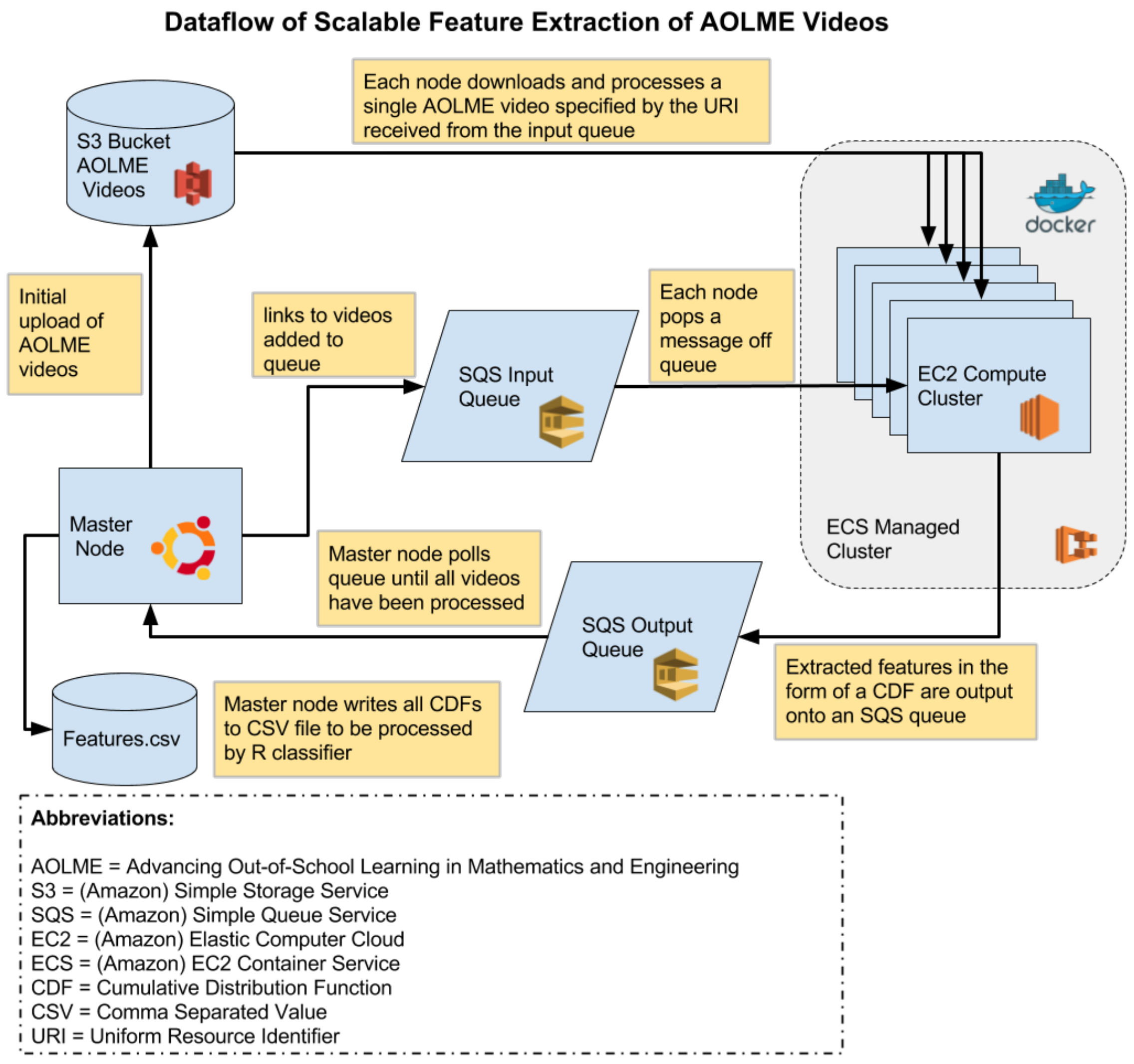

Dataflow of Scalable Feature Extraction of AOLME Videos
S3 Bucket AOLME Videos
Each node downloads and processes a single AOLME video specified by the URI received from the input queue
docker
Initial upload of AOLME videos
links to videos added to queue
SQS Input Queue
Each node pops a message off queue
EC2 Compute Cluster
Master Node
ECS Managed Cluster
Master node polls queue until all videos have been processed
SQS Output Queue
Features.csv
Master node writes all CDFs to CSV file to be processed by R classifier
Extracted features in the form of a CDF are output onto an SQS queue
Abbreviations:
AOLME = Advancing Out-of-School Learning in Mathematics and Engineering
S3 = (Amazon) Simple Storage Service
SQS = (Amazon) Simple Queue Service
EC2 = (Amazon) Elastic Computer Cloud
ECS = (Amazon) EC2 Container Service
CDF = Cumulative Distribution Function
CSV = Comma Separated Value
URI = Uniform Resource Identifier

Figure 3.1: Dataflow of the distributed video system using AWS components

```
1: for each video in pre-segmented videos do
2:     UploadToS3(video)
3: AddVideosToSqsQueue(video_list.txt)
4: while videos in video_list.txt are still in processing do
5:     Sleep(1)
6: output.txt = WriteCdfsToFile(PopAllCdfsFromQueue(output_queue_name))
7: dataframe = ReadCsv(output.txt)
8: for sample_index in 1 to len(dataframe) do        ▷ Leave-one out algorithm for
   training and testing
9:     train_data = dataframe[-sample_index]
10:    test_data = dataframe[sample_index]
11:    knn_result = knn(train_data, test_data, k=3)
12:    svm_model = svm(train_data, kernel=linear, cost = 0.001...10000)
13:    svm_result = predict(svm_model, test_data)
14: WriteExperimentResults(knn_result, svm_result)
```

Figure 3.2: Pseudocode for proposed method in the master node. In the above pseudo-code, indentation indicates the beginning and end of a block of code, in other words when the indentation ends in a for loop, it means that is the end of the for loop. Additionally, all function names represent the same function names that were created in the actual code.

```
$video\_uri$ = PopVideoFromQueue()
$segmented\_video$ = DownloadS3Video($video\_uri$)
$previous\_frame$ = $nullptr$
for each $frame$ in $segmented\_video$ do        ▷ Block extracts features from video frames
    $current\_frame$ = $frame$
    if $previous\_frame$ == nullptr then
        $continue$
    $optical\_flow$ = CalculateOpticalFlow($current\_frame, previous\_frame$)
    $mag$ = $optical\_flow$.GetMagnitude()
    $max\_mag$ = Max($mag$)
    $magnitude = mag > max\_mag \times 0.25$
    $orientation\_vectors$ = $optical\_flow$.GetOrientations()
    $blobs$ = GetBlobs($magnitude$)
    for each $blob$ in $blobs$ do
        $centroid\_x, centroid\_y$ = GetCentroids($blob$)
        $blob\_orientations$ = GetBlobOrientations($blob$)
        $background\_mag$ = GetMotionAround($mag, blob$)
    $num\_bins = 25$
    $pdfs$ = GetPdfs($magnitude, orientation\_vectors$
$centroid\_x, centroid\_y, blob\_orientations, background\_mag, 25$)
$cdf\_features$ = Normalize($pdfs$)
PushToSqsQueue($cdf\_features$, $output\_queue\_name$)
```

Figure 3.3: Pseudocode for proposed method. In the above pseudo-code, indentation indicates the beginning and end of a block of code, in other words when the indentation ends in a for loop, it means that is the end of the for loop. Additionally, all function names represent the same function names that were created in the actual code.

From Figure 3.1 we see that the first step is to upload the videos to S3. We keep the videos very small, because as we show in our experiments section, it takes quite a long time to process large videos therefore there is a significant benefit to keeping the video chunks relatively small so that many machines could potentially work on the feature extraction process. The next step is to place a message on the SQS queue specifying which video to process and what its classification is. For the purposes of this thesis, we manually place messages on the queue so that we can control the flow of messages. In a production system though, we would have the S3 bucket notify the SQS queue that a new video was uploaded and ready for processing. The third step is the processing step. In our setup, we create 20 EC2 instances running our feature extractor application. Each one of these instances polls the SQS queue waiting for a message to arrive. As soon as one does, it downloads the appropriate video from the S3 bucket, processes the video, and then places the results on another SQS queue. At this point, the master node is polling the results queue and collecting the results into a csv file. Finally, the csv file can be used to train the SVM in the R code.

3.1.2 Master Node Configuration

The master node in our system is responsible for sending out jobs to process and then coalescing the results from the calculations performed by the slave nodes. All of these processes are done using the boto3 [4] Python software development kit (SDK). The core implementation of AWS uses a representational state transfer like (RESTful) interface to communicate to all the services that Amazon offers in a programatic way, but they also offer several easy-to-use object oriented libraries written in several languages to make programming easier for the end user. The master node need not be any specific operating system as long as the Python language can be interpreted on it. In this thesis, we use Ubuntu 14.04 to run our master node logic, but it could just as well be OS x or any other flavor of linux.

The master node performs several basic tasks. The first of which is to put messages on the SQS queue. This is orchestrated by reading a csv file that consists of an S3 link to a video segment, the classification of the segment, the SQS queue to which to output the features and finally the optical flow method to use. An example of the file is shown in Table 3.1.

path	**classification**	**sqs_queue**	**algorithm**
aolme/data/typing/seg_1.mp4	1	feature_queue	farneback
aolme/data/notyping/seg_1.mp4	2	feature_queue	farneback

Table 3.1: Example of data file that is used by the master node to place messages on the SQS queue.

From the example data shown in Table 3.1, we can see that the nodes have the ability to switch the algorithm as well as associate a classification from the video. Having the ability to switch method types allows us to easily benchmark using Lucas-Kanade optical flow versus Farneback. We also put the output queue in the message so that the slave nodes know to which queue to place the results of their calculations. This information is also necessary for the master to know which queue to wait on to collect all the results. Additionally, if we need more information to be passed to the slave nodes so that they can effectively do their job, we can easily put that information in the queue with the message trivially.

Once the master node has sent all the messages to the queue, it then polls on the queue it placed the messages on to verify that all the messages have been remove by the slave nodes. This is an important step to validate that the slave nodes are indeed popping messages off the SQS queue and processing the videos that are associated with each message. Once this has been validated, the master node begins to poll on the designated output queue for the results output from each of the slave nodes. Once all the results have been collected, the master node places each of the vectors into a comma separated features file. The pseudo code for the operations performed by the master are shown in Figure 3.4.

As we have shown in this section, very little needs to be configured on the master node other than the ability to run Python and the AWS python utilities. This makes running our software from almost any type of machine very easy with just a few setup steps. The master node plays an important role in sending and receiving the data

```
1: $videos\_to\_process \leftarrow$ ReadInputData(input.csv)
2: for i = 0; i < len(videos_to_process); ++i do
3:     $sqs\_message \leftarrow$ CreateMessage(videos_to_process[i])
4:     $output\_queue\_uri \leftarrow videos\_to\_process[i].output\_queue\_uri$
5:     SendSqsMessage($sqs\_message, sqs\_uri$)
6: while MessagesRemainingInQueue($sqs\_uri$) $\neq 0$ do    ▷ Poll queue every second
7:     Sleep(1)
8: while MessagesRemainingInQueue($output\_queue\_uri$) $\neq 0$ do
9:     $feature\_vectors \leftarrow$ ReceiveSqsMessage(output_queue_uri)
10:    Sleep(1)
11: WriteFeaturesToDisk( $feature\_vectors$ )
```

Figure 3.4: Pseudo code for collecting the features in the master node

that the user wishes to process and is an enabling part of our system. That is to say, it really doesn't matter what software we are running on our slave nodes, as long as the slave nodes fulfill the contract that we have defined in our messaging format. This means that we don't necessarily have to run our C++ extract features program on the slave nodes, but we could be running any flavor of algorithm we wish with no configuration changes on the master node. Not only is this setup scalable, but it's highly flexible because of this idea.

3.1.3 Slave Node Configuration

The next very important piece to our innovative architecture is the algorithm that is run on all the slave nodes. This algorithm simply polls on a single queue, then once a messages is received, it downloads the small S3 video segment, processes it using our feature extraction technique, puts the results on a queue that it has discovered on the incoming message, deletes the video locally and then begins polling on the queue again. Figure 3.5 illustrates this idea clearly.

We can see that the slave node logic is, like the master node, very simple. We

```
while $True$ do
    $sqs\_message \leftarrow$ ReceiveMessages($queue\_name$)                ▷ Blocking call
    $video\_path \leftarrow$ DownloadS3Video($sqs\_message.video\_path$)
    $features\_cdf \leftarrow$ ExtractFeatures($video\_path$)               ▷ Call C++ Code
    SendS3Message($sqs\_message.output\_queue$, $features\_cdf$)
    DeleteSqsMessage($sqs\_message$)                  ▷ Remove message from queue
    Sleep(1)
```

Figure 3.5: Slave Node Implementation Pseudo-Code

simply wait for messages to come in from one queue, process the video, and then output the features onto another queue. However, there is a piece missing from Figure 3.5 that makes the slave nodes a truly innovative part of our overall architecture and that is the orchestration and deployment of our highly refined C++ code.

One of the big hurdles in launching applications that run on a cluster is that all the nodes on the cluster must be running the same libraries, operating system, and versions of the software so that all the answers are returned from the slaves are repeatable and reliable. In traditional systems, this action was typically performed by system admin and as a result, you had to rely on third party packages and libraries that were deployed with the cluster. So if the software under development needed some updates to a package or some bug fixes, you were out of luck. You had to work around those bug fixes and/or write the updates by hand to get similar functionality. This is not so with our system. Using ECS, EC2 and Docker [40], we have developed a system that allows any flavor of linux to be deployed with any version of software that is required to run on the slave nodes seamlessly. For example, we developed our C++ code using OpenCV 3.0 and g++4.8 all on Ubuntu 14.04 and built a Docker container that packaged all that software together. We were then able to deploy our software to an Amazon machine image (AMI) running an Amazon flavor of Linux that was built for running docker and for communicating to an ECS cluster. We did all that without configuring a single Linux instance by hand. And should we choose to roll back to an earlier version of OpenCV or upgrade our compiler to the latest standard, we could do so by changing the configuration of our Docker container and our development environment. This method in no way hinders either vertical or horizontal scalability. With Docker, we can still pass flags that allow the container to take over the host's GPUs so that any code written specifically for the GPUs can still be run in a container.

3.2 Automatic Deployment of Software and the Development Process

An aspect of this thesis that separates us from many of the techniques proposed in the background section, is the way we have developed our system to be readily repeatable, easy to use and flexible to allow to future improvements. This is especially important to UNM's image and video processing and communication lab ivPCL lab so that future graduate students can leverage the work done in this thesis to get a head start on future developments of the proposed feature extraction algorithm. In this section we cover several topics that demonstrate that we have done more than just provide an innovative solution to classifying activities in video, but have also created a system after which other projects can model themselves. Specifically, we cover how the proposed architecture's underlying C++ code is developed and how it is automatically and seamlessly deployed to AWS all the while maintaining a vertically and horizontally scalable architecture.

3.2.1 Vertical Scalability

Since the ivPCL is strongly geared towards vertically scalable solutions using FPGAs and GPUs, it is fitting that we should also make the goal of this thesis to leverage those technologies. To do so, we have developed our software to take advantage of OpenCV's transparent API known as TAPI. The transparent API is an enabling technology to be able to seamlessly switch between GPU, CPU and or any hardware technologies without the software programmer having to select one at compile time or at run time explicitly. TAPI uses Open Computing Language (OpenCL) has its underlying technology to achieve significant improvements over its base algorithm suite. This fundamental technology allows the programmer to write software for a variety of hardware implementations without being burdened with implementing the algorithms by hand. For example, the Farneback optical flow algorithm that is already provided in the OpenCV library can be run easily on either the GPU or the CPU with almost no extra programming on the programmers part. This is also a powerful idea for users who want to distribute software to wide community, but have hardware accelerations that can be added to speed the software up when the hardware is available on the system. So if the programmer has attached an accelerated algorithm for computing the discrete radon transform as demonstrated in [11], it is completely possible to plug that implementation into OpenCL and in turn, program the abstraction layer into the OpenCV library. This also means that the horizontally scalable aspect of this thesis still holds so that we can distribute the same algorithm to nodes with less capable hardware. Although it is optimal to have a cluster of machines with FPGAs hanging from the PCI express bus, implementing the software in OpenCL gives users the option to run the code on almost any compute device. This means that the code can run on machines in AWS or in a local cluster. Furthermore, machines that have the necessary hardware and are accessible in the lab can perform some of the heavy lifting before farming out the rest of the data to the slave nodes on the AWS cloud. So even though the system is designed with horizontal scalability in mind, the option of going vertically scalable for certain computations has been in no way hindered.

3.2.2 Continuous Integration

Its all too often that software is written and completely forgotten about because the build process is too complicated and burdensome on the users of the interface. A goal of this thesis is to maximize reusability and maintainability. We accomplish this using a popular software technique called continuous integration [18]. Continuous integration is the idea that software should be constantly giving developers feedback about the state of their software to ensure robustness and to have an obvious tool for developers to use document the build process of the software. This is an especially important tool for teams of developers so that failures of new commits can easily be observed quickly and effectively. This has the advantage of addressing errors early on in the software development phase so that errors are addressed immediately.

For this thesis, we chose to use an online tool called Travis-CI for our continuous integration. Travis easily hooks into our Github repository and automates the build process of our extract features program. The build of our C++ program consists of compiling and linking against all third party libraries, running all of our google test unit tests, and then deploying the Docker image to AWS to be used for deployment on the ECS cluster. This tool significantly unburdens the developer from having to do these steps manually. Continuous integration also ensures that the unit tests that have been developed for the software work on the deployed environment specifically in the docker image. In other words, we develop the software locally on any flavor of linux, but there is a chance that any new changes that we have made to the software do not work in the deployed system. Travis-CI acts as an automated alert that something might not be right in the software.

The fact that the continuous integration also pushes our Docker image up to our ECS cluster is also huge advantage. This process ensures that our Cluster is always running the latest compiled and unit tested version. So not only have we made deployment of our software easy using a Docker container to package it up, we've also automated the entire deployment process of our software simply by pushing our code to our Github repo.

Making the deployment and build of our software automated has guaranteed us that we will have repeatability in our code and that we have the latest functioning version running on our compute cluster. Furthermore, if mistakes are made at the local development level and they are not caught until the developer has pushed their changes to a Git repo, the automated build system will trigger a failure and not deploy the system. A key contribution with using the automated build system is that anyone from anywhere will be able to repeat the experiments that we have performed in this paper with very little software configuration. We have essentially frozen the code in a working state. Even if AWS goes away, the core Docker image is available to be run on any *nix type machine with Docker installed. So if future users decide that they want to use our extract features algorithm, it is ready to run. Figure 3.6 illustrates the general flow of the continuous integration.

3.2.3 Test Driven Development

For the majority of our C++ development, we used a software engineering technique called test driven development (TDD) [8]. This technique is germane to our method's section because repeatability of our experiments and future extension of our work depends on well developed, robust software. The main idea is that before we ever wrote code to implement a new algorithm, we first design a failing test and then write code to make that test pass. This has several advantages over not writing tests for code:

- It gives developers a predictable way to code. Rather than thinking abstractly about what goals a developer is attempting to accomplish, the developer writes

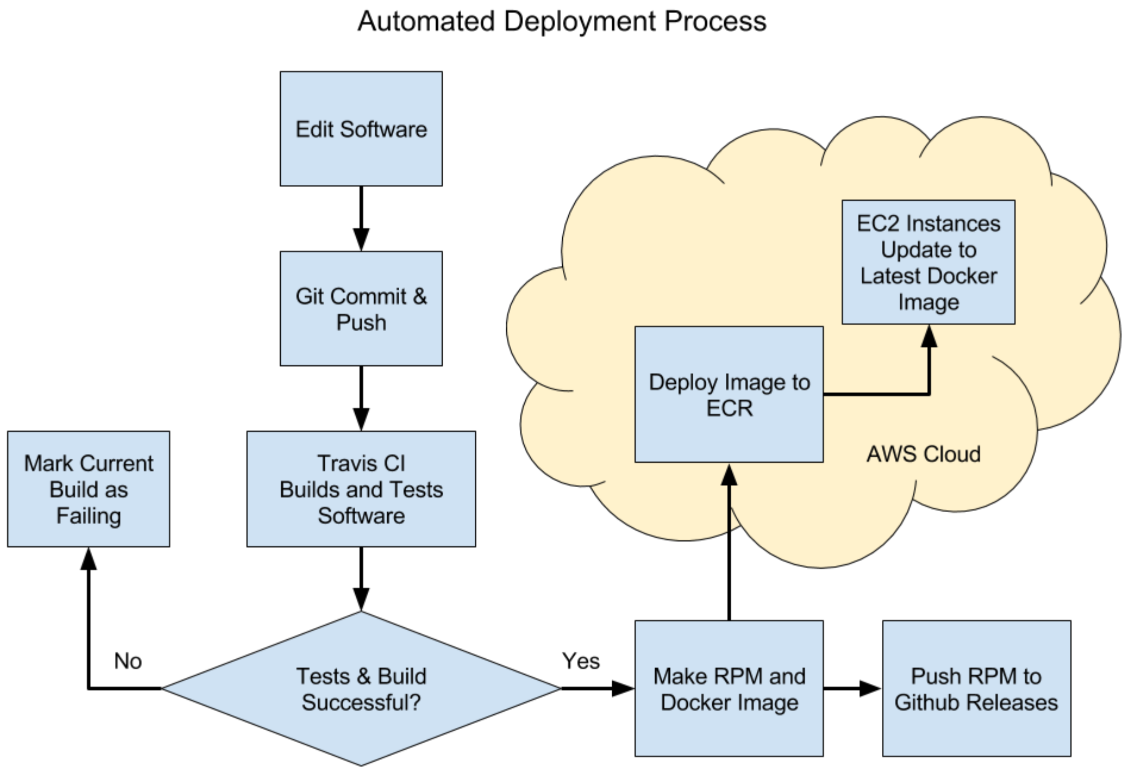

Automated Deployment Process
Edit Software
Git Commit & Push
Travis CI Builds and Tests Software
Mark Current Build as Failing
No
Tests & Build Successful?
Yes
Make RPM and Docker Image
Push RPM to Github Releases
Deploy Image to ECR
EC2 Instances Update to Latest Docker Image
AWS Cloud

Figure 3.6: Automated deployment of the proposed feature extraction method to the AWS cloud using continuous integration

concrete tests that should validate the behavior that is desired.

- It allows for lessons to be learned early on about the implementation details of the software. If a developer delves directly into solving a software problem with a single mindset about how it should be implemented, then the opportunity for a different design pattern to be used is extinguished. The main idea being that if test driven development is used for designing an algorithm then the best design pattern for the problem will be chosen, not just the one the developer is familiar with.

- Test driven development forces developers to be accountable. All too often code is written with the mindset that its okay to commit hundreds or even thousands of lines of code most of which is barely tested. And thus the code becomes riddled with bugs and other developers end up having to fix those issues. With TDD, this is a less frequent occurrence because developers must write a test for every logical unit of code.

For this thesis we use TDD to develop our extract features program and to design several programs that we use for experimental purposes. In order to facilitate the development, we use google test as our C++ testing framework [21]. An example of a test we have written to test receiving a single video frame from a file is shown in Appendix C. This test uses a concept from the TDD community known as a mocked object. The philosophy behind using a mocked video reader as illustrated in the test, is that we don't want to design our tests to be reliant on the state of the current system. If we do so, the second we push our code to the repository, our continuous integration tool will fail. Mocked objects give us the ability to test our algorithms without having to rely on the state of our development machine, the network or other variable elements. This is a powerful concept because it ensures that our tests run fast and also forces us to program to interfaces rather than concrete objects. In order to fully leverage the power of mocked objects, we use a design pattern, as shown Appendix C, known has dependency injection [20]. The idea behind dependency injection is that we can inject our dependencies into an algorithm at runtime or compile time to have our object get the resources it needs to perform a calculation in a variety of ways. For example, in our test in Appendix C, we inject the interface called "Reader". "Reader" is a specific example of high-performance dependency injection because no virtual interfaces are used. This concept can only be used at compile time and the interfaces cannot be swapped out at runtime. High performance dependency injection is achieved by using only templates, which by their nature, can only be determined at compile time and not run time. In order to achieve type inference at runtime, we would need to use a virtual interface instead which has the additional overhead of doing a virtual table lookup. As a result of using this reader interface instead of using OpenCV's video reading capabilities, we give ourselves the flexibility to read from hierarchical data format (HDF5) files, a web interface, or anything as long as we adhere to the contract we have defined in the interface code. And thus we can also write a mocked object, which is to say, an object that we can easily define the inputs and outputs of on the fly so that we can test a variety of scenarios in our motion estimation algorithm without having to rely on any external resources. Figure 3.7 illustrates how dependency injection looks for our MotionEstimation class.

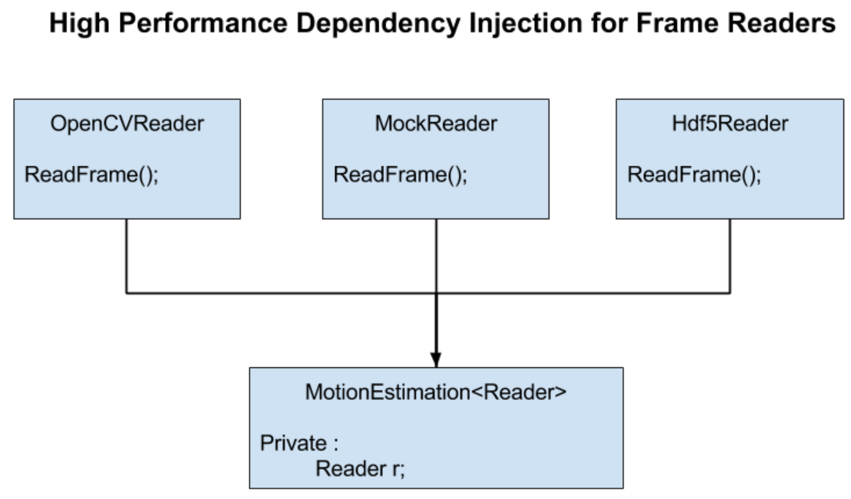
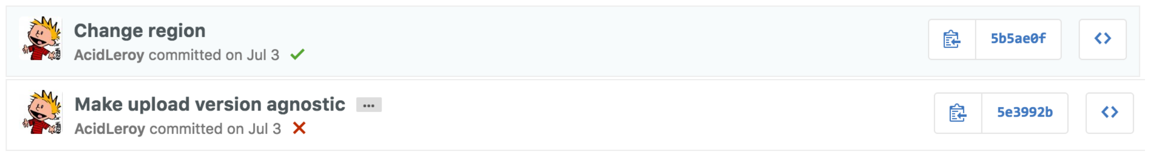


Figure 3.7: High performance dependency injection using templates to implement the motion estimation class with different types of video readers.

The appendix example demonstrates how we now have a failing test, and we can then begin implementing the code that causes test to pass. As a result, if we push our failing test to Github, our current build will not be deployed to the cluster until we have our tests passing. Figure 3.8 illustrates what a commit looks like when it is passing on our Github project and Figure 3.8 shows what it looks like when the build is failing because of a test, compilation issues, or problems pushing to the AWS cloud.



Change region
AcidLeroy committed on Jul 3 ✓
5b5ae0f

Make upload version agnostic
AcidLeroy committed on Jul 3 ✗
5e3992b

Figure 3.8: An example of two commits that were pushed to our repository. The top figure shows how the build is marked as passing with a green check mark, and the bottom shows the build failing with a red x.

3.3 Implementing Optical Flow

In our software, we use two OpenCV library calls, `goodFeaturesToTrack` and `calcOpticalFlowPyrLK` to implement the Lucase-Kanade Pyrmidal optical flow. The first function is used to find features that can be easily tracked from one frame to the other using the Shi-Tomasi algorithm [53]. The next method then calculates the optical flow between the good points using the pyramidal implementation of the Lucas-Kanade algorithm [9]. Figure 3.9 outlines the general program flow for calculating motion vectors in the proposed feature extraction method.

The algorithm used in our proposed architecture is similar to Figure 3.9 but contains fewer steps since there is no need to get good features to track. In the C++ software, we also implemented Farneback method. The Farneback implementation is shown in Figure 3.10.

As can be seen in 3.10, we don't need any good features to track because we

```
1: procedure CalculateVectors($frame1$, $frame2$)
2:     if $track\_points\_initialized$ then
3:         $opticalflow \leftarrow$ calcOpticalFlowPyrLK($track\_points, frame1, frame2$)
4:     else
5:         $track\_points \leftarrow$ goodFeaturesToTrack($frame1$)
6:         $track\_points\_initialized \leftarrow True$
7:         $optical\_flow \leftarrow$ CalculateVectors($frame1, frame2$)
       return $optical\_flow$
```

Figure 3.9: Calculating Lucas-Optical Flow from Videos

```
1: procedure CalculateVectors($frame1$, $frame2$)
2:     $optical\_flow \leftarrow$ calcOpticalFlowFarneback($frame1, frame2$)
3: return $optical\_flow$
```

Figure 3.10: Calculating Farneback Flow from Videos

are calculating the optical flow globally between frames, rather than selecting a few features. This has the advantage of tracking optical flow objects that may fail the Shi-Tomasi method for tracking, but because it is no discriminant in the features, the resulting motion vectors are dense.

3.4 Comparison of Methods

We implemented the Lucas-Kanade method first in our research because in general, performance is a concern and, as long as not too many features or too few features are detected, the Lucas-Kanade algorithm will be faster [16]. Despite this fact, we found that our classifier did not perform as well on features extracted from the Lucas-Kanade method, as it did using the Farneback method. Hence most of the results in this thesis have been calculated with Farneback optical flow unless otherwise specified.

3.5 Feature Extraction from Optical Flow

The first step that needs to be done in our software is to open a video file. All of the videos that are used in this thesis are compressed before reaching an S3 bucket, and then are later decompressed within each compute node. This is important to note because bandwidth is a limiting factor when transferring videos from an S3 bucket to the compute node. Therefore the first step is to have OpenCV read the file in and decompress it before reducing the feature space. Figure 3.11 illustrates how a video is first read in to extracting the features from each of the cropped videos producced. The `CalculateVectors` function in both Figure 3.9 and 3.10 returns several dense matrices that represent the features that we can extract from the optical flow output. These features are magnitude, orientation, x direction and y direction of the optical flow features. These are ultimately the features that we use to train and classify using an SVM. However, if we had two $N \times M$ video frames as the input, we now have $4 \times N \times M$ features. Clearly we have not yet reduced the input feature space. Thus, based on information that we know *a-priori*, we can reduce our feature space significantly.

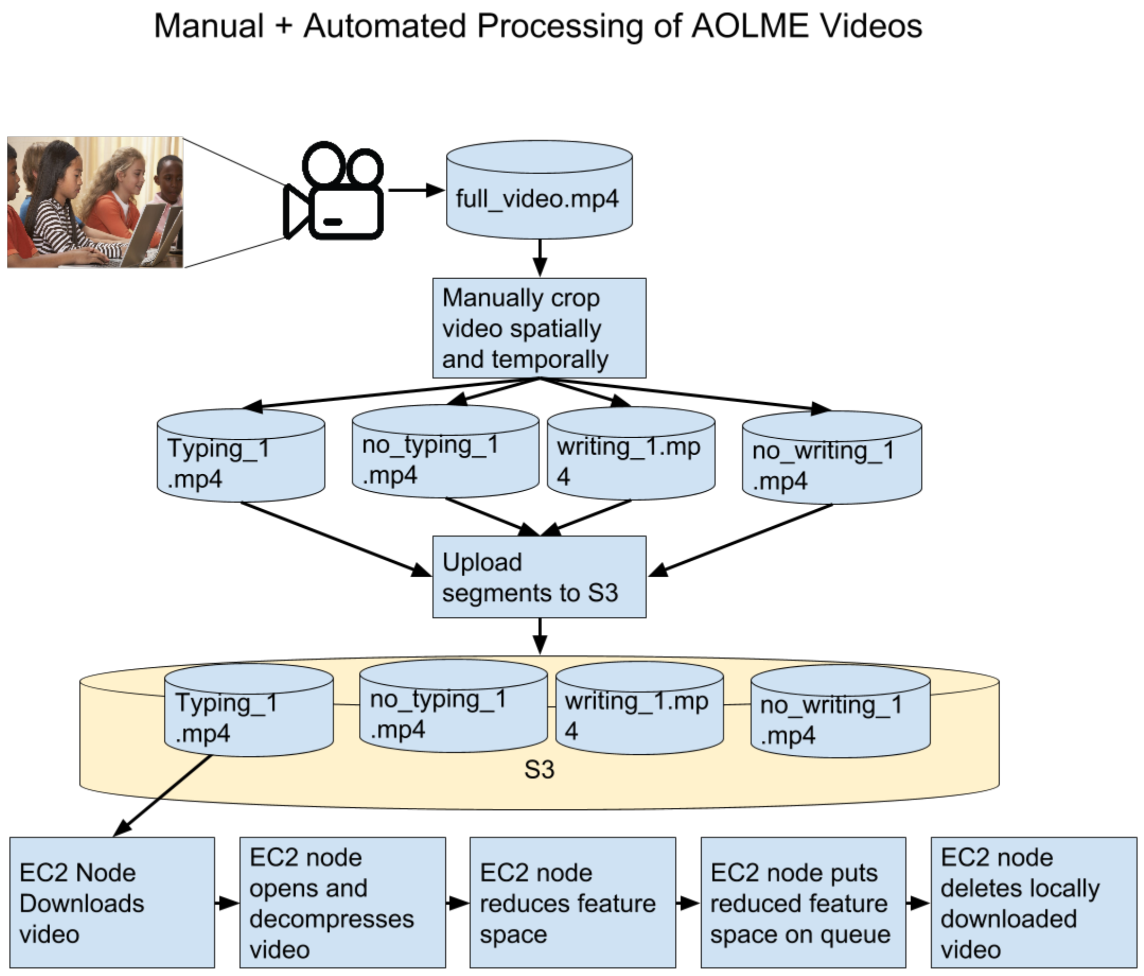

Manual + Automated Processing of AOLME Videos
full_video.mp4
Manually crop video spatially and temporally
Typing_1 .mp4
no_typing_1 .mp4
writing_1.mp 4
no_writing_1 .mp4
Upload segments to S3
Typing_1 .mp4
no_typing_1 .mp4
writing_1.mp 4
no_writing_1 .mp4
S3
EC2 Node Downloads video
EC2 node opens and decompresses video
EC2 node reduces feature space
EC2 node puts reduced feature space on queue
EC2 node deletes locally downloaded video

Figure 3.11: Processing of a single raw video to feature extraction as implemented in the proposed architecture.

In the case of typing and writing, we know we can expect there to be motion from one frame to the next. We don't know by how much, but we do know that it is not zero. Using this knowledge, we can then threshold the optical flow vectors that we get back from Figure 3.9 and 3.10. The threshold value used was empirically calculated from doing multiple runs on the AOLME videos. We found we got the best results by only retrieving optical flow vectors with a magnitude greater than 75% of the max value. The set of Equations in 3.1 illustrate this idea.

$$\begin{aligned} \mathbf{V} &= (\mathbf{V_x}, \mathbf{V_y}) \\ \mathbf{V_m} &= \begin{cases} 1, & \text{if } \|\mathbf{V}\| \geq \max(\|\mathbf{V}\|) \times 0.25 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

where $\mathbf{V_x}$ and $\mathbf{V_y}$ are the optical flow vectors in the x and y directions respectively. $\mathbf{V_m}$ is the bit mask that is then used to extract the subset of data from each of the dense matrices.

$$\begin{aligned} &\text{Let:} \\ &\|\mathbf{V\prime}\| = \|\mathbf{V}\| \circ \mathbf{V_m} \\ &\mathbf{V_x\prime} = \mathbf{V_x} \circ \mathbf{V_m} \\ &\mathbf{V_y\prime} = \mathbf{V_y} \circ \mathbf{V_m} \\ &\mathbf{\Phi\prime} = \mathbf{\Phi} \circ \mathbf{V_m} \end{aligned} \tag{3.2}$$

Using the optical flow bitmask, $\mathbf{V_m}$, we can then extract features from each one of our dense matrices using the Hadamard product as shown in Equation 3.2, where $\|\mathbf{V\prime}\|, \mathbf{V_x\prime}, \mathbf{V_y\prime}, \mathbf{\Phi\prime}$ are subset matrices for the magnitude, x and y direction and orientation respectively. We have now reduced the feature space somewhat, but depending on the size of the video and the amount of entropy per frame pair, we could still have a significant amount of data to process for classification, this idea is especially true for Farneback optical flow.

In addition to extracting generic vectors from the video, we also add geometrical centroids, blob orientation and background motion around the blobs to the optical flow statistics being used for classification. We implement these methods to attempt to leverage information that could be useful during classification. In order to calculate the geometrical centroids and orientations of each blob, we use some well known algorithms available in OpenCV, `connectedComponentsWithStats` and `findContours` [23]. `connectedComponentsWithStats` is a function that allows us to compute the centroid for each blob of connected pixels. The input to this function is our binary mask image, $\mathbf{V_m}$. Once we have all the connected blobs, we can then calculate the orientation of each one of those blobs using `findContours` in combination with with `fitEllipse`. The full implementation of this algorithm is outlined in Appendix A. The final step is to then dilate each blob, and then retrieve the magnitude of the optical flow in this region. Appendix B gives the C++ code that was used for these calculations. Figure 3.12 illustrates the idea of acquiring the centroid and orientation of the blobs from $\mathbf{V_m}$.

When the previous optical flow features have been generated, their values are then organized into a probability density function (PDF) with 25 bins. That is to say that each frame pair generates a PDF and that PDF is accumulated for every subsequent frame in the video sequence. When our software reaches the end of the video file, a normalized, cumulative distribution function (CDF) is calculated and output for each vector. So for each input video there will be one CDF with 25 bins for blob orientation, blob centroid x and y, motion vector magnitude, motion vector orientation and background motion vector magnitude. Figure 3.13 clearly illustrates this concept.

Ultimately, these are the features that are then accumulated for multiple AOLME

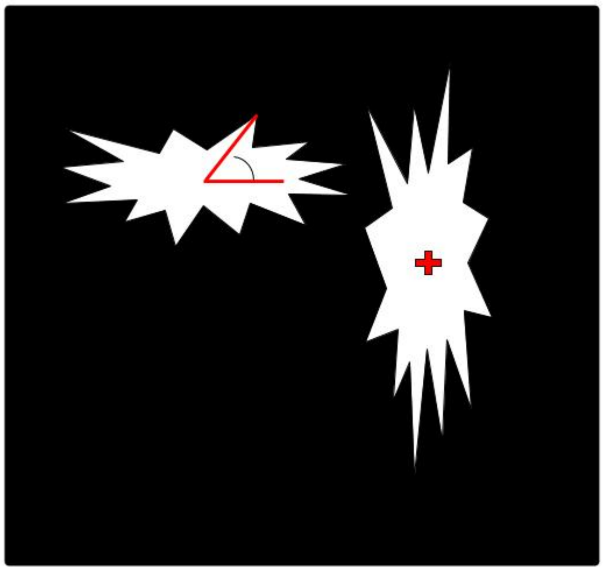


Figure 3.12: Example of orientation measurement on the left, and the centroid calculation on the right.

videos and used for classification.

3.6 Classifying the Reduced Feature Space

At this point we now have accumulated a bag of features for videos. The features that are collected are stored in a comma separated file (csv) that can be read in by the any of the popular machine learning packages such as those provided by the R language or Python's SciKit-Learn. The file contains labels that have filename, centroid x CDF, centroid Y CDF, background motion CDF, motion magnitude CDF, motion orientation CDF and classification. We can then use an SVM to classify the features. To validate our results, we use leave-one-out cross validation to ensure that we have not overfit the data.

This thesis uses the SVM software that is included in the R language for accurate

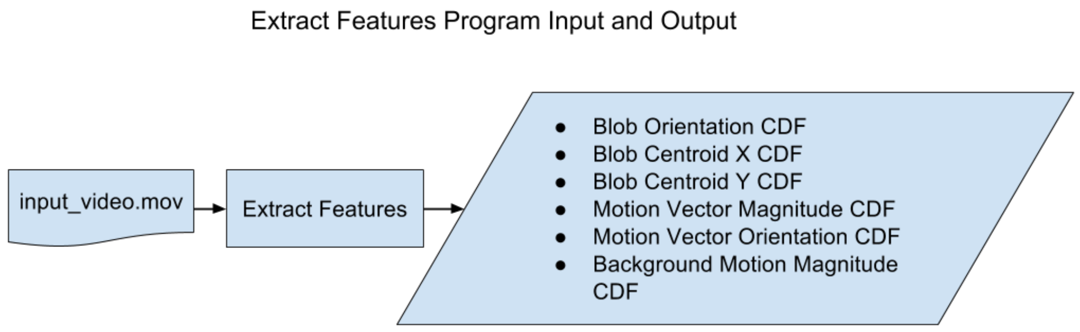

Extract Features Program Input and Output
input_video.mov
Extract Features
Blob Orientation CDF
Blob Centroid X CDF
Blob Centroid Y CDF
Motion Vector Magnitude CDF
Motion Vector Orientation CDF
Background Motion Magnitude CDF

Figure 3.13: Flow of the extract features program. For every input video, it will return a CDF with 25 bins for each of the extracted features from the motion vectors

classification. The algorithm is based off the original paper written by Vapnik [15] but was then much improved for computational efficiency by Chang & Lin in 2011 [14] with their award winning software package known as LIBSVM. This library was originally written in C, but many fans of the algorithm have created software bindings for multiple languages, including R.

The classification of our feature vectors is very simple since the majority of the hard work has already been implemented in the machine learning algorithms we use to do the classification. The process is as follows

- Load features from CSV file into an R data frame
- Plot statistics about the features
- Loop over data frame using leave one out cross correlation
- Select most accurate results between K nearest neighbors and tuned, non-linear support vector machine.

The R code used to do the classification is shown in Appendix D.

Chapter 4

Results & Discussion

4.1 The AOLME Dataset

The AOLME dataset is an enormous repository of over 900 hours of video recordings of students. The videos contain students interacting with facilitators, their peers and computers to write code in Python on the Raspberry Pi. The dataset is wealth of information but difficult to exploit in its current state. The data used for this thesis is a subset of the entire AOLME dataset. By hand, we have selected several videos and extracted typing and writing clips from the original dataset and are using these as ground truth for measuring the accuracy of our methods.

As Figure 1.1 suggests, we are only using a cropped version of the video. The reason for this is that we are not attempting to solve the tracking problem in this thesis, only the classification problem. Hence, we assume that the videos entering into our software have already been clipped and cropped with the target activities inside of them and the corresponding lack of the activity. Our subset of the AOLME database consists of the following:

- Twenty videos of typing
- Twenty videos of no typing
- Twenty videos of writing
- Twenty Videos of no writing

4.2 Accuracy of Classification

In this section we explore how well our results are for both the classification of typing and writing videos using the techniques described in methods chapter.

For our first set of results, we ran to classifying typing motions in videos. The input messages into the cluster are shown in Table 4.1. The original dataset, however, contains 10-20 for each of the training classifications, we have left them out in this table for brevity.

Using our R code, we then plot some statistics about the vectors that have come back from the cluster.

path	**classification**	**sqs_queue**	**algorithm**
aolme/data/typing/seg_1.mp4	1	feature_queue	farneback
aolme/data/typing/seg_2.mp4	1	feature_queue	farneback
aolme/data/typing/seg_3.mp4	1	feature_queue	farneback
aolme/data/typing/seg_4.mp4	1	feature_queue	farneback
aolme/data/typing/seg_5.mp4	1	feature_queue	farneback
aolme/data/typing/seg_6.mp4	1	feature_queue	farneback
aolme/data/typing/seg_7.mp4	1	feature_queue	farneback
aolme/data/typing/seg_8.mp4	1	feature_queue	farneback
aolme/data/typing/seg_9.mp4	1	feature_queue	farneback
aolme/data/typing/seg_10.mp4	1	feature_queue	farneback
...	...	...	...
aolme/data/notyping/seg_1.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_2.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_3.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_4.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_5.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_6.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_7.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_8.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_9.mp4	2	feature_queue	farneback
aolme/data/notyping/seg_10.mp4	2	feature_queue	farneback

Table 4.1: Data that is sent to the SQS for calculation on the cluster. The original dataset includes 10-20 for both classifications

With those feature vectors, we found that we were able to get the confusion matrix shown in Table 4.2

	typing	**no typing**
typing	19	1
no typing	3	17

Table 4.2: Confusion matrix for classification accuracy for typing

From Table 4.2 we can see that we get 90% accuracy for classifying typing motions on the keyboard. We had difficulty classifying videos that had significant motion in them, but the motion was not typing and we also found that we had trouble classifying the videos where there is was not much typing in the videos that were classified as typing. But overall, when the scene clearly had typing and when it clearly did not, we found that we had 90% accuracy.

Our results for determining writing, however, were not as good as our results for classifying typing. Our CDFs for typing are shown in figure

	writing	**no writing**
writing	18	2
no writing	11	9

Table 4.3: Confusion matrix for classification accuracy for typing

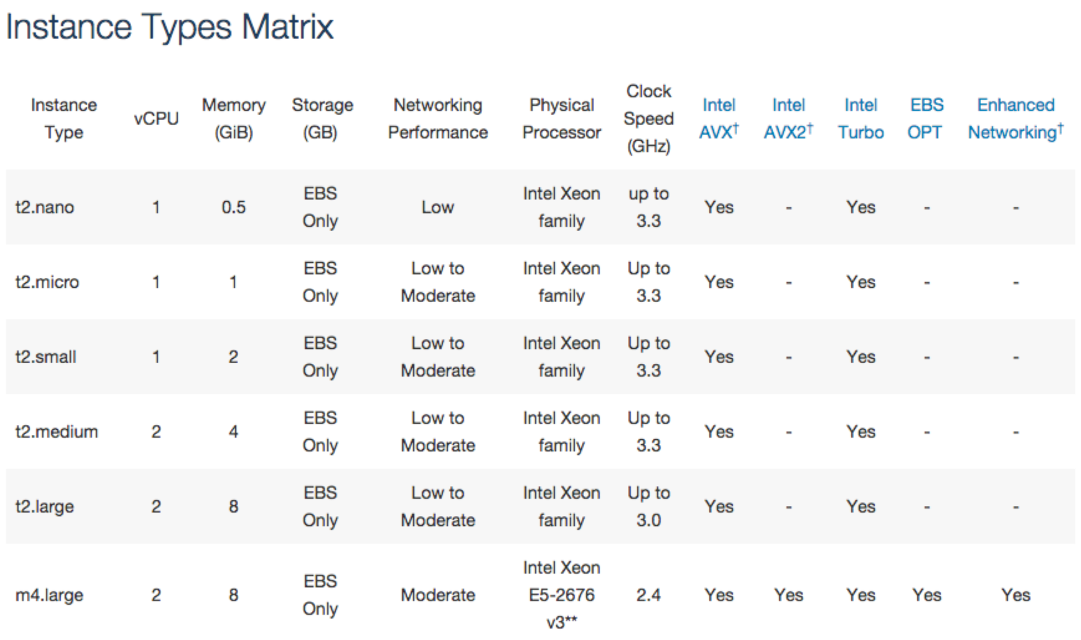
As can be seen in Table 4.3, we did not get as good as results for writing, only about 65% classification. In these results we find that our algorithm struggled more with classifying videos that had no writing in them as having writing in them. This may mean that the motion vectors we are extracting are highly dependent on the type of scene we are looking at. Furthermore, the original algorithm was developed in Matlab and then ported to C++ for this thesis. We saw differences in the feature vectors between the two implementations; however, classification results proved to be very similar.

4.3 Proof of Scalability

In order to show that our system is scalable, we record the time it takes for the cluster to perform certain repetitive tasks. For the first experiment, we have the cluster operate using only a single EC2 instance, and then scale the experiment by one instance and compare how long it takes to calculate 10 2.1MB videos. For this experiment, we used Amazon's t2.micro instance which contains 1 virtual CPU running on a high frequency Intel Xeon processor with turbo up to 3.3GHz and contains 1GB of memory. Because t2 instances are designed to have burstable performance, Amazon does not list any specific processor on their webpage as can be seen in Figure 4.3. Amazon designed these instances like this so that the user is not consistently charged for using high performance CPUS, but rather only when they need the performance are they charged for those compute cycles. As a result, T2 instances can take advantage of Intel capabilities such as native instructions for AES encryption (AES-NI) Advanced Vector Extensions (AVX) for floating point and Turbo Boost where the CPU core can be made to run faster. So for this reason, we cannot give an exact chip type that was used when performing these computations because it is possible that the virtual CPU that we were initially assigned, is not the same CPU that we received later on in the experiment. This is the instance that is considered to be part of the free tier program. Our instance runs a special Amazon Machine image The results from this experiment are show in Figure 4.3

From Figure 4.3, we can see that as long as the number of instances that we have divides evenly into the number of videos that calculate, then we get a linear increase in speed. In other words, we get 10x speed up when using 10 instances rather than just using a single instance.

The size of the videos does matter. The smaller that the videos are, the less the overhead is when running the cluster. The reason for this is because not only does it take longer to transfer larger videos, but it also takes more time to process them. So from our experiments, it looks like keeping the videos under 2MB is optimal for distributing and processing the videos. We tested how well the t2.micro instances performed against our MacBook Pro 15 inch with an Intel Core i7 clocked at 2.7 GHz and has 16GB of RAM. It should also be noted that the Macbook Pro contains an AMD Radeon R9 M370X GPU with 2048 GB of memory. This allows us to take advantage of the TAPI programming model that we leverage as described in



Instance Types Matrix

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)	Intel AVX†	Intel AVX2†	Intel Turbo	EBS OPT	Enhanced Networking†
t2.nano	1	0.5	EBS Only	Low	Intel Xeon family	up to 3.3	Yes	-	Yes	-	-
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.small	1	2	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.medium	2	4	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.large	2	8	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.0	Yes	-	Yes	-	-
m4.large	2	8	EBS Only	Moderate	Intel Xeon E5-2676 v3**	2.4	Yes	Yes	Yes	Yes	Yes

Figure 4.1: A subset of instances that can be used for processing on the cloud. Notice how t2 instances are not associated with any specific processor, only the processor family.

the Methods section. As shown in figure 4.3, we can see that even though we have significant processing power locally, we still must receive the message from the queue and then process the videos; therefore there is actually little gained in terms of performance even though we are able to leverage the onboard GPUs. Figure 4.3 also demonstrates that 10 instances run at exactly the same speed as a single instance. So from this graph we can also infer that even though we have a higher power CPU technically on the t2.micro instance, we are able to leverage OpenCV's transparent API and take advantage of the local GPU on the Macbook Pro.

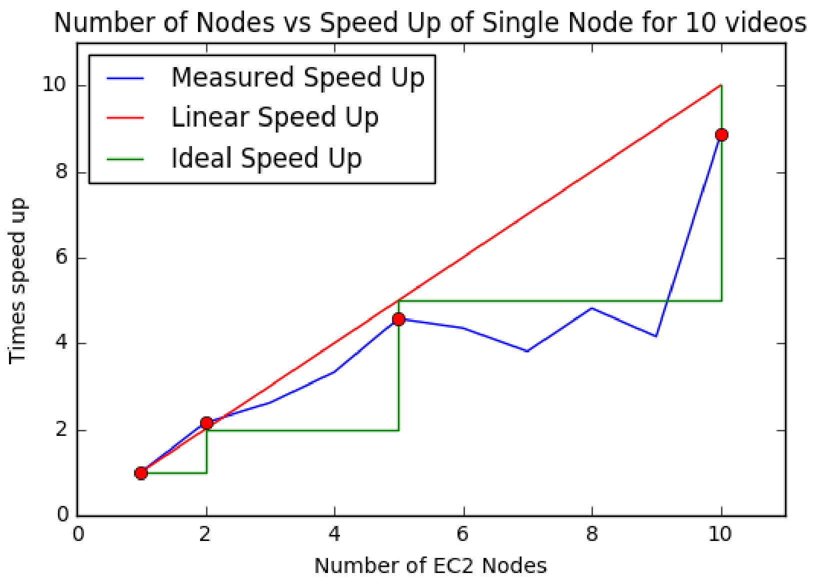


Figure 4.2: Leaving the number of videos to process the same, we increase the number of EC2 instances to illustrate the speed up.

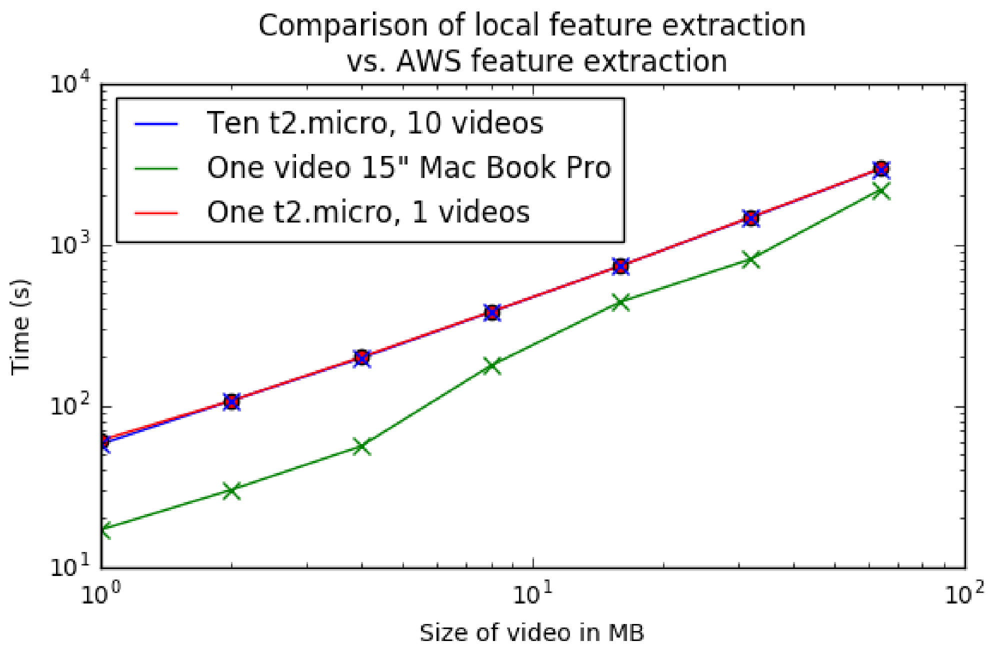


Figure 4.3: Comparison of the time it takes for a single node to process 1 video vs the time it takes a cluster to process 10 videos. Videos vary in size to test the efficacy of choosing to send smaller vs larger videos to the cluster. A single t2.mirco instance was included to show that a single instance takes just as long as 10 instances.

Additionally, we can see that the performance of S3 has a linear download and upload speed from an EC2 instance. We have collected data that calculates the mean of 10 sample download-upload pairs of a given file size. The results of this are shown in Figures 4.4 and 4.5. So as expected, we can rely on S3 to give us a linearly predictable download and upload rate. Because though, it can take a significant time to download, process, and upload videos, it is more beneficial for the distributed system to break videos into smaller pieces so that no one node is occupied for a long period of time. If this notion is followed, it is much easier to scale the feature extractions horizontally.

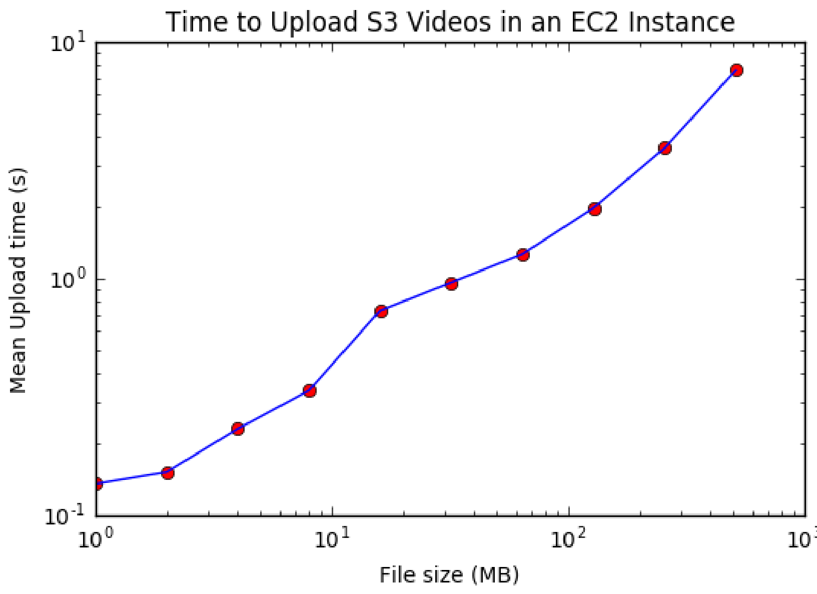


Figure 4.4: Average time to upload varying file sizes in S3 using an EC2 instance.

The upload times can be seen in Table 4.4

The download times can be seen in Table 4.5

When we consolidate the numbers above, we find that we end up with an average upload speed of 37.0486495118 MB/s and an average download speed of 40.149124154 MB/s.

File Size	Seconds
Upload 1MB (s)	0.135685
Upload 2MB (s)	0.152184
Upload 4MB (s)	0.231676
Upload 8MB (s)	0.336037
Upload 16MB (s)	0.732164
Upload 32MB (s)	0.963004
Upload 64MB (s)	1.269231
Upload 128MB (s)	1.989168
Upload 256MB (s)	3.567965
Upload 512MB (s)	7.608010

Table 4.4: Average upload speeds in seconds on an EC2 instance to an S3 bucket

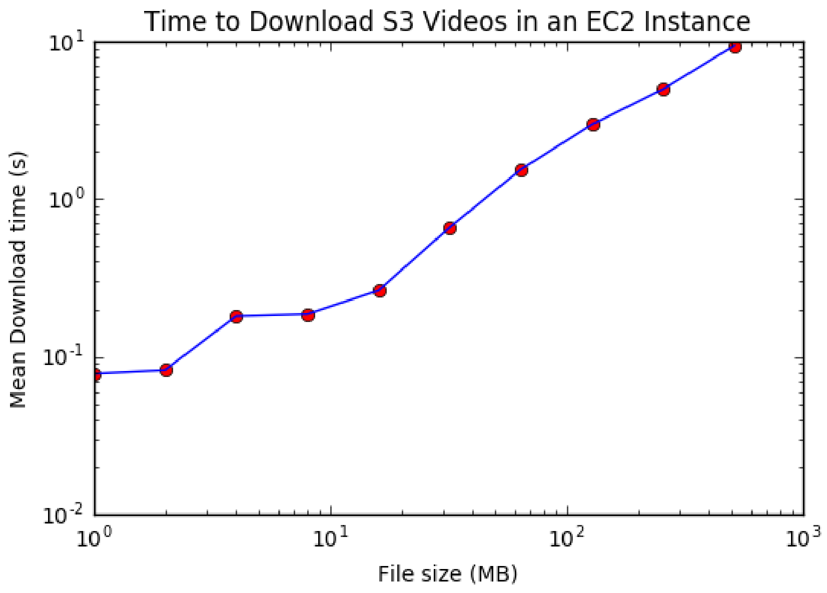


Figure 4.5: Average time to download varying file sizes in S3 using an EC2 instance.

File Size	Seconds
Download 1MB (s)	0.078182
Download 2MB (s)	0.082242
Download 4MB (s)	0.181033
Download 8MB (s)	0.186510
Download 16MB (s)	0.263108
Download 32MB (s)	0.661547
Download 64MB (s)	1.542783
Download 128MB (s)	2.976941
Download 256MB (s)	4.990264
Download 512MB (s)	9.406564

Table 4.5: Average Download speeds in seconds on an EC2 instance from S3

4.4 Discussion

In the previous sections we showed that we can accurately classify typing in small segments of video and demonstrated how to we can horizontally and vertically scale our system in the cloud. In order to achieve this, we had to find the optimal points to break the system apart so that the computationally expensive aspects of the system could be handled by the compute cluster, and the quick computations could be handled by the master node, or a client node. The quick computations in the master or client node are made possible by the cluster of computers reducing the video files down to only a few significant CDFs. Thus, given a set of CDFs that are only tens of kilobytes in size, we were able to quickly train a machine learning algorithm using some ground truth videos, and then could quickly classify features being output from the system in microseconds.

Based on the plot in Figure 4.3, we showed that the system we have proposed in this thesis is horizontally scalable to at least 10 nodes. More nodes could have easily been selected, but in an attempt to keep the cost of this thesis low, we decided to only use what is available by default in the AWS cloud. However, based on the success of other applications, such as Netflix that have used cloud technologies to scale their system to hundreds or even thousands of nodes, there is no reason to believe that this system would not scale to the same order of magnitude, though more care would probably need to be taken to scale the nodes strategically on the AWS network services.

Finally we showed that is possible to create a greatly reduced feature space from videos and accurately classify when students are typing at nearly 90%. Because the feature space is so significantly reduced after processing the videos on the cloud, classification of videos becomes a task that takes only milliseconds once the system has been trained. Furthermore, even training, once the feature space has been reduced, only takes a matter of minutes for very large datasets. As we also showed, we did not get very statistically significant results for classifying writing. It is unclear why the motion vectors for this activity were not as significant as they were for typing.

4.4.1 Limitations

One of the limitations of this research was retrieving sufficient ground truth data for the videos that are analyzed. Many of the datasets that were used in other research in this area have large datasets with ground truth associated with them. Since the dataset we use in this paper is novel, we didn't have the time nor the resources to generate a dataset with hundreds of samples with ground truth.

Additionally, we were resource bound financially for this research. If we had more money to conduct the research, we could have tested the system at a larger scale to prove that it would work beyond just ten nodes.

Since most of the focus on this paper is primarily to investigate how to efficiently distribute and analyze videos in the AWS cloud, there was less time for investigation in determining the best ways to classify the human activity. One of the short comings of this research was that it didn't address trying to classify typing against writing. In other words, the paper doesn't investigate classification of typing from writing, it only investigates whether we classify a video as having typing in it, or not having typing in it.

Chapter 5

Future Work & Conclusion

5.1 Future Work

This thesis investigates the basic idea how to efficiently distribute and classify human activity in the AOLME dataset and as a result there a few areas where the research can be greatly improved. The first suggestion to improve the in the field of this research is to greatly increase the size of truth data for more statistically significant data. The research done in this paper consists of 40 video subsets, many of which contained clips from the same footage but at varying times. Increasing this database to several hundred would improve research here.

We also didn't investigate whether we could accurately classify typing vs writing. This would be an extension of this research that would be important so that we could investigate if our algorithm could determine the difference between the two activities.

In other aspect that would be worth investigating for extending this thesis, is to attempt to add an interactive aspect to training and testing. In other words, it would be interesting to precompute and train the machine learning algorithms on multiple human actions and then interactively searching through the feature space to then observe the videos that matched the classification. Work in this area would greatly aid with the manual analysis of the AOLME videos that is currently being done.

Finally, we found that we didn't get very good results for classifying writing. Investigation into this would be interesting to see why the results here varied so much from the results we obtained from typing classification. It's still unknown why the bag of features failed to classify writing accurately as it did for typing. As a suspicion, with no supporting evidence, it may have something to do that general hand movement around paper is similar to that of writing, therefore the algorithm may struggle for this reason.

5.2 Conclusion

We presented a novel video processing architecture and algorithm for human activity classification in large video databases such as the AOLME dataset. Our method is both horizontally and vertically scalable thanks to enabling technologies in the cloud as well as convenient APIs provided by OpenCV for easy switching between CPU and GPU implementations of Lucas-Kanade and Farneback optical flow methods. In addition to the scalability of our system, we also presented an accurate method for detecting typing in video segments extracted from the AOLME dataset. Our algorithm greatly reduced the original feature space of gigabytes down to just a few kilobytes, which makes the bandwidth limit on the cloud very manageable. Because the output feature space is quite small compared with the original size of the videos input into the system, training and testing can be done very rapidly and in turn automatic classification can be done once the system has been trained at near real-time rates. Finally, all of the source code can be retrieved and used from our Github Repo located at https://github.com/AcidLeroy/OpticalFlow.

Appendices

Appendix A

Retrieving Centroids and Orientations from Blobs

This is a snippet of code that illustrates how we retrieve orientations and centroids from blobs of grouped motion vectors. This code is used to extract additional features from each video frame pair as described in the methods section.

```
void UpdateCentroidAndOrientation(const cv::Mat& thresholded_image,
                                  cv::Mat* orientations, cv::Mat* centroids) {
  cv::Mat labels, stats, current_centroids;
  cv::connectedComponentsWithStats(thzresholded_image, labels, stats,
                                   current_centroids);
  // Don't care about background centroid, hence the range.
  if (centroids->empty()) {
    current_centroids(cv::Range(1, current_centroids.rows),
                      cv::Range(0, current_centroids.cols)).copyTo(*centroids);
  } else {
    cv::vconcat(*centroids,
              current_centroids(cv::Range(1, current_centroids.rows),
                                cv::Range(0, current_centroids.cols)),
              *centroids);
  }

  std::vector<std::vector<cv::Point>> contours;
  cv::findContours(thresholded_image, contours, cv::RETR_LIST,
                   cv::CHAIN_APPROX_NONE);
  for (size_t i = 0; i < contours.size(); ++i) {
    // Can only fit an ellipse with 5 points, skip others
    if (contours[i].size() >= 5) {
      cv::RotatedRect result = cv::fitEllipse(contours[i]);
      orientations->push_back(result.angle);
    }
  }
}
```

Appendix B

Retrieving Statistics Around Motion Blob

This is a snippet of code that relates to extract features as described in the Methods chapter.

```
/**
 * Get the histogram for the image around the motion.
 */
template <typename T>
void GetHistoAround(const T& thresholded_motion, int disk_size,
                    const T& gray_scale_image, T* bg_histogram) {
  T dialated;
  // Get disk
  T disk = cv::getStructuringElement(cv::MORPH_ELLIPSE,
                                     cv::Size(disk_size, disk_size));
  // Dilate motion to get pixels around motion
  cv::dilate(thresholded_motion, dialated, disk);
  T diff_image = dialated - thresholded_motion;
  diff_image.convertTo(diff_image, CV_8U);
  T background;
  gray_scale_image.copyTo(background, diff_image);
  // Get histogram of background intensity
  constexpr int num_bins = 25;
  constexpr float range[] = {0, 256};  // the upper boundary is exclusive
  const float* hist_range = {range};
  bool uniform = true;
  bool accumulate = false;

  T hist;
  cv::calcHist(&background, 1, 0, T(), hist, 1, &num_bins, &hist_range, uniform,
               accumulate);
  if (bg_histogram->empty()) {
    hist.copyTo(*bg_histogram);
  } else {
    (*bg_histogram) = (*bg_histogram) + hist;
  }
}
```

Appendix C

Example of Google Test

This snippet of code demonstrates how unit tests were performed on pieces of the code that seemingly depended on reading a real file from the file system. This is described in the Methods section.

```
#include "gtest/gtest.h"
#include "gmock/gmock.h"
#include "mock_reader.h"
#include "motion_estimation.h"

TEST(MotionEstimation, OnlyOneFrameInImageSequence) {
  std::shared_ptr<MockReader> mock(new MockReader{"some_file.mov"});
  MotionEstimation<MockReader, cv::Mat> me(mock);
  std::shared_ptr<cv::Mat> a_ =
      std::make_shared<cv::Mat>(cv::Mat(256, 256, CV_8U));
  cv::randu(*a_, 0, 256);
  std::shared_ptr<Image<>> frame1{new Image<>(a_)};
  EXPECT_CALL(*mock, ReadFrameMat())
```

```
      .WillOnce(Return(frame1))
      .WillOnce(Return(nullptr));

  std::shared_ptr<MockFlow> mflow{new MockFlow()};
  ASSERT_THROW(me.EstimateMotion<MockFlow>(mflow), MotionEstimationException);
}
```

Appendix D

R Code for SVM Classification

This sample code is the same code that was used in the master node for classification of the features. It uses the leave one out strategy for training and testing the optimal solution. This is referenced from the Methods chapter in this thesis.

```
ClassifyFeatures <- function(VideoHists){
  NoOfSamples <- length(VideoHists$Classification)

  # Build a factor of the correct classification:
  All_cl <- unlist(VideoHists$Classification);

  # Store 1 for wrong classification and 0 for correct.
  knnResult <- rep(1, times=NoOfSamples);
  svmResult <- rep(1, times=NoOfSamples);

  # Remove classification
  no_use = c("Filename", "Classification")
  features = GetAllExcept(VideoHists, no_use)
```

```
    # Create a leave one out classification approach
    for(i in 1:NoOfSamples)
    {
      # Set up training and testing data:
      trainData = lapply(features, function(x) x[-i]) # Remove i.
      testData  = lapply(features, function(x) x[i]) # One left out.

      #Combine data
      trainData = t(CombineFeatures(trainData, names(trainData)))
      testData = t(CombineFeatures(testData, names(testData)))

      # Prepare the labels for the training set:
      #  Optimal: k=1
      knnResult[i] <- knn (trainData, testData, All_cl[-i], k=3); # 3

      #**** With tuning ****#
      tune.out=tune(svm, trainData, All_cl[-i], , kernel="linear", ranges=list(cos
      svmResult[i] <- predict(tune.out$best.model, testData);



      #***** Without tuning *****#
  #    model <- svm(All_cl[-i] ~ ., data=trainData, scale=FALSE);
  #    svmResult[i] <- predict(model, testData);
      cat("SVM result =  ", round(svmResult), "\n");
      cat("KNN result =  ", knnResult, "\n")
    }
```

References

[1] Agurto, C., Barriga, S., Murray, V., Nemeth, S., Crammer, R., Bauman, W., Zamora, G., Pattichis, M., and Soliz, P. Automatic detection of diabetic retinopathy and age-related macular degeneration in digital fundus images. *Investigative Ophthalmology and Visual Science 52*, 8 (2011), 5862–5871.

[2] Agurto, C., Murray, V., Barriga, S., Murillo, S., Pattichis, M., Davis, H., Russell, S., Abramoff, M., and Soliz, P. Multiscale am-fm methods for diabetic retinopathy lesion detection. *IEEE Transactions on Medical Imaging 29*, 2 (Feb 2010), 502–512.

[3] Agurto, C., Murray, V., Yu, H., Wigdahl, J., Pattichis, M., Nemeth, S., Barriga, S., and Soliz, P. A multiscale optimization approach to detect exudates in the macula. *IEEE Journal of Biomedical and Health Informatics 18*, 4 (July 2014), 1328–1336.

[4] Amazon. Boto 3 amazon web services sdk for python. `https://boto3.readthedocs.io/en/latest/`, 2014.

[5] Amazon. Architecting for the cloud. `https://d0.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf`, 2016.

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., et al. Above the clouds: A berkeley view of cloud computing.

[7] Bashir, F. I., Khokhar, A. A., and Schonfeld, D. Object trajectory-based activity classification and recognition using hidden markov models. *IEEE transactions on Image Processing 16*, 7 (2007), 1912–1919.

[8] Beck, K. *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[9] Bouguet, J.-Y. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation 5*, 1-10 (2001), 4.

[10] Cai, J., Yu, J., Imai, F., and Tian, Q. Towards temporal adaptive representation for video action recognition. In *2016 IEEE International Conference on Image Processing (ICIP)* (2016), IEEE, pp. 4155–4159.

[11] Carranza, C., Llamocca, D., and Pattichis, M. The fast discrete periodic radon transform for prime sized images: Algorithm, architecture, and vlsi/fpga implementation. In *IEEE Southwest Symposium on Image Analysis and Interpretation* (april 2014), pp. 169–172.

[12] Carranza, C., Llamocca, D., and Pattichis, M. Fast and scalable computation of the forward and inverse discrete periodic radon transform. *IEEE Transactions on Image Processing 25*, 1 (Jan 2016), 119–133.

[13] Carranza, C., Murray, V., Pattichis, M., and Barriga, S. Multiscale am-fm decompositions with gpu acceleration for diabetic retinopathy screening. In *IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)* (April 2012), pp. 121–124.

[14] Chang, C.-C., and Lin, C.-J. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST) 2*, 3 (2011), 27.

[15] Cortes, C., and Vapnik, V. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[16] de Boer, J., and Kalksma, M. Choosing between optical flow algorithms for uav position change measurement.

[17] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[18] Duvall, P. M. *Continuous Integration.* Pearson Education India, 2007.

[19] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis* (2003), Springer, pp. 363–370.

[20] Gamma, E. *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.

[21] Google. Google test. `https://github.com/google/googletest`, 2008.

[22] Horn, B. K., and Schunck, B. G. Determining optical flow. *Artificial intelligence 17*, 1-3 (1981), 185–203.

[23] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[24] Jiang, Y., Esakki, G., and Pattichis, M. Dynamically reconfigurable architecture system for time-varying image constraints (drastic) for hevc intra encoding. In *Asilomar Conference on Signals, Systems and Computers* (Nov 2013), pp. 1112–116.

[25] Jiang, Y., Llamocca, D., Pattichis, M., and Esakki, G. A unified and pipelined hardware architecture for implementing intra prediction in hevc. In *IEEE Southwest Symposium on Image Analysis and Interpretation* (April 2014), pp. 29–32.

[26] Jiang, Y., and Pattichis, M. A dynamically reconfigurable architecture system for time-varying image constraints (drastic) for motion jpeg. *Journal of Real-Time Image Processing* (2014), 1–17.

[27] Jiang, Y., Zong, C., and Pattichis, M. Scalable hevc intra frame complexity control subject to quality and bitrate constraints. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (Dec 2015), pp. 290–294.

[28] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2014), pp. 1725–1732.

[29] Kaseb, A. S., Mohan, A., and Lu, Y.-H. Cloud resource management for image and video analysis of big data from network cameras. In *2015 International Conference on Cloud Computing and Big Data (CCBD)* (2015), IEEE, pp. 287–294.

[30] Konda, K. R., Memisevic, R., and Michalski, V. Learning to encode motion using spatio-temporal synchrony. *arXiv preprint arXiv:1306.3162* (2013).

[31] Kuehne, H., Gall, J., and Serre, T. An end-to-end generative framework for video segmentation and recognition. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2016), IEEE, pp. 1–8.

[32] Laptev, I., Marszalek, M., Schmid, C., and Rozenfeld, B. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (2008), IEEE, pp. 1–8.

[33] Lee, S., Pattichis, M., and Bovik, A. Foveated video compression with optimal rate control. *IEEE Transactions on Image Processing 10*, 7 (Jul 2001), 977–992.

[34] Lee, S., Pattichis, M., and Bovik, A. Foveated video quality assessment. *IEEE Transactions on Multimedia 4*, 1 (Mar 2002), 129–132.

[35] Llamocca, D., and Pattichis, M. Dynamic energy, performance, and accuracy optimization and management using automatically generated constraints for separable 2d fir filtering for digital video processing. *ACM Trans. Reconfigurable Technol. Syst. 7*, 4 (dec 2014), 31:1–31:30.

[36] Loizou, C., Murray, V., Pattichis, M., Pantziaris, M., Nicolaides, A., and Pattichis, C. Despeckle filtering for multiscale amplitude-modulation frequency-modulation (am-fm) texture analysis of ultrasound images of the intima-media complex. *International journal of biomedical imaging 2014* (2014).

[37] Loizou, C., Murray, V., Pattichis, M., Pantziaris, M., and Pattichis, C. Multiscale amplitude-modulation frequency-modulation (am-fm) analysis of ultrasound images of the intima and media layers of the carotid artery. *IEEE Transactions on Information Technology in Biomedicine 15*, 2 (March 2011), 178–188.

[38] LopezLevia, C., Celedon-Pattichis, S., and Pattichis, M. Integrating mathematics, engineering and technology through mathematics modeling and video representations. In *13th International Congress on Mathematical Education* (Hamburg, Germany, 2016).

[39] Lucas, B. D., Kanade, T., et al. An iterative image registration technique with an application to stereo vision. In *IJCAI* (1981), vol. 81, pp. 674–679.

[40] Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J. 2014*, 239 (Mar. 2014).

[41] Murray, V., Pattichis, M., Barriga, S., and Soliz, P. Recent multiscale am-fm methods in emerging applications in medical imaging. *EURASIP Journal on Advances in Signal Processing 2012*, 1 (2012).

[42] Murray, V., Pattichis, M., Davis, H., Barriga, S., and Soliz, P. Multiscale am-fm analysis of pneumoconiosis x-ray images. In *16th IEEE International Conference on Image Processing* (Nov 2009), pp. 4201–4204.

[43] Murray, V., Rodriguez, P., and Pattichis, M. Multi-scale am-fm demodulation and image reconstruction methods with improved accuracy. *IEEE Transactions on Image Processing 19*, 5 (May 2010), 1138–1152.

[44] Niebles, J. C., Chen, C.-W., and Fei-Fei, L. Modeling temporal structure of decomposable motion segments for activity classification. In *European conference on computer vision* (2010), Springer, pp. 392–405.

[45] Panayides, A., Constantinides, A., Pattichis, M., Kyriacou, E., and Pattichis, C. Adaptive real-time hevc encoding of emergency scenery video. In *EAI 4th International Conference on Wireless Mobile Communication and Healthcare (Mobihealth)* (Nov 2014), pp. 217–217.

[46] Pattichis, M., and Bovik, A. Analyzing image structure by multidimensional frequency modulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 5 (May 2007), 753–766.

[47] Pattichis, M., Bovik, A., Havlicek, J., and Sidiropoulos, N. Multidimensional orthogonal fm transforms. *IEEE Transactions on Image Processing 10*, 3 (Mar 2001), 448–464.

[48] Pattichis, M., Panayi, G., Bovik, A., and Hsu, S.-P. Fingerprint classification using an am-fm model. *IEEE Transactions on Image Processing 10*, 6 (Jun 2001), 951–954.

[49] Pattichis, M., Pattichis, C., Avraam, M., Bovik, A., and Kyriacou, K. Am-fm texture segmentation in electron microscopic muscle imaging. *IEEE Transactions on Medical Imaging 19*, 12 (Dec 2000), 1253–1258.

[50] Ramachandran, J., Pattichis, M., Scuderi, L., and Baba, J. Tree image growth analysis using instantaneous phase modulation. *EURASIP Journal on Advances in Signal Processing 2011* (2011).

[51] Ribeiro, P. C., and Santos-Victor, J. Human activity recognition from video: modeling, feature selection and classification architecture. In *Proceedings of International Workshop on Human Activity Recognition and Modelling* (2005), Citeseer, pp. 61–78.

[52] Rohrbach, M., Amin, S., Andriluka, M., and Schiele, B. A database for fine grained activity detection of cooking activities. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (2012), IEEE, pp. 1194–1201.

[53] Shi, J., and Tomasi, C. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on* (1994), IEEE, pp. 593–600.

[54] Sidiropoulos, N., Pattichis, M., Bovik, A., and Havlicek, J. Coperm: Transform-domain energy compaction by optimal permutation. *IEEE Transactions on Signal Processing 47*, 6 (Jun 1999), 1679–1688.

[55] Soomro, K., Zamir, A. R., and Shah, M. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* (2012).

[56] Spriggs, E. H., De La Torre, F., and Hebert, M. Temporal segmentation and activity classification from first-person sensing. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (2009), IEEE, pp. 17–24.

[57] Vo, N. N., and Bobick, A. F. From stochastic grammar to bayes network: Probabilistic parsing of complex activity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 2641–2648.

[58] Wang, X., and Qi, C. Action recognition using edge trajectories and motion acceleration descriptor. *Machine Vision and Applications* (2016), 1–15.

[59] Wang, Y., Chen, W.-T., Wu, H., Kokaram, A., and Schaeffer, J. A cloud-based large-scale distributed video analysis system. In *2016 IEEE International Conference on Image Processing (ICIP)* (2016), IEEE, pp. 1499–1503.

[60] White, T. *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[61] Yury Izrailevsky, S. V. . R. M. Completing the netflix cloud migration. `https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration`, January 2016. [Online; posted 11-January-2016].