The Pennsylvania State University
The Graduate School
College of Engineering

INFORMATION RETRIEVAL FROM IMAGES AND VIDEOS IN MOBILE NETWORKS

A Dissertation in
Computer Science and Engineering
by
Noor Felemban

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2022

The dissertation of Noor Felemban was reviewed and approved by the following:

Thomas F. La Porta
Professor, School of Electrical Engineering and Computer Science
Dissertation Advisor, Chair of Committee

Ting He
Associate Professor, School of Electrical Engineering and Computer Science

Vishal Monga
Professor, School of Electrical Engineering and Computer Science

Soundar Kumara
Professor, Industrial Engineering

C. Lee Giles
Professor, College of Information Sciences and Technology

Chita R. Das
Professor, School of Electrical Engineering and Computer Science
Head of the department of Computer Science and Engineering

Abstract

The widespread usage of mobile devices has led to the generation and storage of a large amount of videos and images. These images and videos are a rich source of information, and searching them for specific occurrences of actions and objects aids in various applications. Currently mobile devices have limited computational power to process some simple machine learning (ML) algorithms. Cellular and WiFi networking has enabled resource-constrained mobile devices to access powerful servers aided with graphical processing units (GPUs), and permitted offloading videos and images to be processed on the cloud. Various queries can be issued in mobile networks, and responding to such queries requires the consideration of the limited network bandwidth, and the computational and energy constraints on the devices.

This thesis explores and investigates different techniques and mechanisms to address various queries in mobile networks. First, we address the problem of collecting images, stored on mobile devices, containing a queried object in a mobile network. The goal is to minimize the query response time given an energy constraint. In order to achieve this goal we divide the process into a pipeline with different stages. Each stage makes offloading decisions based on the network conditions, the cloud backlog and the hit-rate. A filtering stage is designed to run on the mobile device that uses a smaller less accurate but faster CNN to filter out images not containing the object of interest. Following that images that are positively classified in the filtering stage move to the selection stage where the more accurate CNN is used to classify the image and search for the queried object. At any stage, if the mobile device makes the decision to offload, the image is transmitted to the video cloud to be processed there. At the end, all the images that are positively classified should be uploaded to the server, where the requester can have access to them.

Next, we investigate a different problem where we have a mobile network with limited bandwidth and a requester broadcasts a query to a set of mobile devices where the requester is searching for images containing novel objects that the CNNs loaded in the network are not trained to classify. By utilizing the previously loaded and pre-trained models and using the extracted features from the CNNs along with a distance measure, we are able to respond to these queries in a manner that saves on both network bandwidth and mobile device energy.

Finally, we explore a more complex media and study video retrieval in mobile networks. The goal is to identify video clips containing a queried action in a fast manner. In order to achieve this goal, various audio (sound and speech) and visual (frames and video clips) data are used. The idea of dividing the processing task into stages is also used

in this work. We design a pipeline that utilizes various machine learning techniques (audio classification, speech-to-text, object classification and action recognition) in each stage in order to quickly discard false positives from earlier stages. By utilizing various algorithms and machine learning models and properly distributing the processing among the mobile device and the server GPUs we are able to minimize the query response time.

Table of Contents

List of Figures viii

List of Tables x

Acknowledgments xii

Chapter 1
Introduction 1
1.1 Motivation and Challenges . . . 2
1.2 Contributions and Dissertation Outline . . . 3

Chapter 2
Preliminary Concepts and Related Works 7
2.1 Overview of Deep Learning and Image and Video Processing . . . 7
2.1.1 Convolutional Neural Networks . . . 8
2.1.2 Deep Learning Frameworks . . . 10
2.1.2.1 Caffe . . . 10
2.1.2.2 Tensorflow . . . 10
2.1.2.3 Pytorch . . . 11
2.1.3 Datasets . . . 11
2.1.4 CNN Classification Models . . . 12
2.2 Image and Video Processing in Crowdsourcing Applications . . . 15

Chapter 3
Fast Image Search on Distributed Mobile Networks 17
3.1 Introduction . . . 17
3.2 Overview of Image Processing . . . 19
3.3 PicSys Overview . . . 20
3.4 Experiments and Design Decisions . . . 22
3.4.1 **Image pre-processing** . . . 23
3.4.2 **Choosing the deep learning models and model partitioning** 24
3.4.3 **Choosing the k_1 and k_2 values** . . . 25
3.5 PicSys Optimization . . . 26
3.5.1 Mathematical Formulation: no-energy constraints . . . 26

3.5.1.1 **Components of Delay** . . . 26
3.5.1.2 **Formulation** . . . 28
3.5.2 Proposed Heuristic: no-energy constraints . . . 31
3.5.2.1 **Estimating the cloud completion time** . . . 31
3.5.2.2 **Estimating the device completion time** . . . 31
3.5.2.3 **Heuristic** . . . 32
3.5.3 **Cloud tokens** . . . 33
3.5.4 Complexity . . . 35
3.6 Experimental Evaluation . . . 35
3.6.1 Baselines . . . 35
3.6.2 Simulation Results . . . 36
3.6.2.1 **Comparison to optimal** . . . 36
3.6.2.2 **Testbed Implementation** . . . 37
3.6.2.3 **Larger systems** . . . 39
3.6.2.4 **Heterogeneous systems** . . . 41
3.6.2.5 **Comparison to no-Filtering stage** . . . 41
3.7 Related Work . . . 43
3.8 Conclusion . . . 45

Chapter 4
Energy-Efficient Fast Image Search on Distributed Mobile Networks 46
4.1 Introduction . . . 46
4.2 PicSys Optimization . . . 47
4.2.1 Mathematical Formulation: energy constraints . . . 47
4.2.2 Proposed Heuristic: energy constraints . . . 49
4.2.3 Design Considerations . . . 51
4.2.3.1 **Energy Probing** . . . 51
4.2.4 Complexity . . . 51
4.3 Experimental Evaluation . . . 51
4.3.1 Simulation Results - Energy . . . 52
4.3.2 Related Work . . . 53
4.3.3 Conclusion . . . 54

Chapter 5
EDIR: Efficient Distributed Image Retrieval of Novel Objects in Mobile Networks 55
5.1 Introduction . . . 55
5.2 Overview of Image Processing and Definitions . . . 57
5.3 System Overview . . . 59
5.4 EDIR Model . . . 61
5.4.1 Server . . . 62
5.4.1.1 **Calculating closest base classes** . . . 62
5.4.1.2 **Choosing k and c** . . . 63
5.4.1.3 **Calculating θ** . . . 69

5.4.1.4 **Server algorithm** . . . 70
5.4.2 Edge devices . . . 70
5.5 Design Considerations . . . 71
5.5.1 Closest base classes and thresholds . . . 71
5.6 Performance evaluation . . . 74
5.6.1 System Implementation . . . 74
5.6.2 Choosing parameters c and k . . . 75
5.6.2.1 **Full Training Dataset** . . . 75
5.6.2.2 **Small Number of Samples** . . . 78
5.6.3 Comparison to Mahalanobis-distance algorithm . . . 79
5.6.3.1 **Bandwidth and Energy savings** . . . 80
5.6.4 Varying the number of novel samples . . . 81
5.7 Related Work . . . 82
5.8 Conclusion . . . 85

Chapter 6
VidQ **86**
6.1 Introduction . . . 86
6.2 System Overview . . . 88
6.3 Experimentation . . . 89
6.3.1 Performance Metrics . . . 90
6.3.2 Processing Stages . . . 90
6.3.3 Query Types . . . 94
6.3.3.1 **Long Videos Query** . . . 95
6.3.3.2 **Short Videos Query** . . . 97
6.4 VidQ . . . 99
6.4.1 Optimization Formulation . . . 101
6.5 VidQ Heuristic . . . 105
6.6 Performance Evaluation . . . 108
6.6.1 Comparison to Optimal . . . 108
6.6.2 Experimental Evaluations . . . 109
6.6.3 Sensitivity Analysis . . . 113
6.7 Related Work . . . 115
6.8 Conclusion . . . 116

Chapter 7
Conclusion and Future Work **118**

Bibliography **121**

List of Figures

2.1 Low-level filters from AlexNet [34]. 9

2.2 Inception module - GoogLeNet. 13

2.3 Fire module - SqueezeNet. 14

2.4 Residual block. 14

3.1 Overview of the system on the mobile device. 21

3.2 Scaling time using FFmpeg. 23

3.3 Processing time for images. 23

3.4 Recall-ILSVRC2012. 24

3.5 Recall- per classifier. 24

3.6 Aggregate recall on VOC2012. 25

3.7 Implementation results. 38

3.8 Simulation results: time-heuristic. 39

3.9 Three vs. two stage. 42

3.10 Comparison of PicSys to [25]. 43

4.1 Energy-heuristic: varying a_m. 52

4.2 Decision stage distribution. 53

5.1 Overview of the system. 60

5.2 Server process overview. 61

5.3 Edge device process overview. 61

5.4 Classification histogram. 64

5.5 Distance to MAV_{tank}: left of threshold is classified as 'tank', right of threshold is classified as 'ship'. 73

5.6 Distance to MAV_{apple}. 73

5.7 Classification of 'ship' samples ($k = 1$). 74

5.8 Error in chosen parameters, full training set. 76

5.9 Error in chosen parameters, 10 samples. 78

5.10 Number of uploads (TP and FP) for different novel classes. 79

5.11 F1 score for different novel classes. 80

5.12 Varying the number of samples. 82

6.1 Overview of the system. 89

6.2 Mobile device processing times for full video divided in clips of various sizes: Audio classification. 92

6.3 Mobile device and server processing times: 1-minute clip. 94

6.4 Query response time with different approaches. 108

6.5 NFL example: query response time, amount of uploads and accuracy. . . 112

6.6 Varying hit-ratio. 113

6.7 Varying server processing time (β). 114

6.8 Varying transmission rate (r). 114

List of Tables

3.1 Nomenclature and Notation 27

3.2 Completion time [sec] 36

3.3 Decision trace optimal 37

3.4 Decision trace heuristic 37

3.5 Heterogeneous system 41

4.1 Nomenclature and Notation 48

5.1 Definitions and Notation 59

5.2 Closest base classes to the novel class 72

5.3 Estimation results when varying c and k for the full training set 77

5.4 Experimental results: top-10 F1 scores 77

5.5 Amount of data sent from the server to a device 81

6.1 Video and audio file sizes for various clip lengths 91

6.2 NFL: Confusion matrix for various stages 96

6.3 NFL: Evaluation metrics for various stages 96

6.4 Confusion matrix for each of the two stages 98

6.5 Evaluation metrics for the complete query 99

Acknowledgments

I would like to firstly thank my parents for raising me and constantly encouraging me to pursue my dreams and excel in every aspect of my life. Because of them, I was able to pursue my graduate school journey, and without them I would not have been able to complete it. I thank them for all of the opportunities they gave me, and for all the sacrifices they made. I would also like to thank my parents-in-law, my brother, and my sisters, for their constant support and for always being there for me.

Next, I would like to thank my advisor, Professor Thomas La Porta, for being a great supervisor and mentor. I am honored and thankful that I had the opportunity to be guided by him throughout both my masters and PhD studies. Along with the technical knowledge he enriched me with, he helped me develop people skills and professional skills that I will always carry with me.

I would like to thank my collaborators Zongqing Lu, Kevin Chan, Heesung Kwon, Hana Khamfaroush, and Fidan Mehmeti, for working closely with me and sharing their knowledge and expertise with me. I would also like to thank my dissertation committee members Dr.Ting He, Dr.Vishal Monga, Dr. Soundar Kumara, and Dr. C. Lee Giles, for taking the time to review my work and sharing their valuable input.

I would also like to thank the awesome people I had the pleasure of meeting throughout my stay at Penn State, specifically my labmates: Diman Zad Tootaghaj, Stefan Achleitner, Nicolas Papernot, Injung Kim, Vajiheh Farhadi, Berkay Celik, Kristina Sorensen, and Mingming Chen. Thank you all for listening to me talk about random things at odd times!

I am grateful to my friends in Saudi Arabia and the US for always being there for me, supporting me, and believing in me. A special thanks to Al-Hanouf, Raneem, Rasha, Al-Maha, Sara, Jawaher, Shaikha, and Haya.

A very special thanks to my husband Gohar, who I met during this journey, for walking alongside me and supporting me until the end. And to my daughter Linah, thank you for bringing so much joy and love into my life.

Funding acknowledgments Finally, I would like to acknowledge Imam Abdulrahman Bin Faisal University and the Ministry of Education in Saudi Arabia, for providing me with my scholarship to complete my studies.

Part of this research was sponsored by the Army Research Laboratory and accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and

Dedication

To my family, for their endless love and continuous support and encouragement.

Chapter 1
Introduction

With the increasing number of smartphone users [1], and the installment of millions of surveillance cameras [2], the number of generated videos and images is very large. Images and videos are a rich source of information. Processing their content and extracting features to aid in query responses and data collection and retrieval is an important area of research. Due to the large number of available data the task of searching these sources becomes expensive. Applications for distributing processing among network resources, and optimizing processing time and information collection delay are needed.

Crowdsourcing can be used to respond to queries and search and collect data. Mobile devices, such as smartphones are equipped with enough computational power to run simple classification and inference algorithms, and they also have sufficient storage to store models for local processing. Machine learning techniques have been developed to process images and videos and perform tasks such as object detection and classification [3], action recognition [4], audio classification [5] and speech-to-text [6].

Various queries that request images and videos are issued. In cases such as a missing person, suspicious activities, and acts of terrorism, investigators are required to collect as much useful information as possible in a fast manner. An example of an information query may be: "Search for a person carrying a backpack at the park around noon time". Devices can filter images and videos that have been captured based on location and timestamp. The next step is identifying objects such as a person or a backpack. In order to detect such objects, classification algorithms need to be used.

A different type of query would be searching a large amount of videos for clips that contain a queried action. An example would be searching a library of stored football games for field goals. In order to respond to such query, both audio cues and visual cues can be used. An example for an audio cue that can be used is searching for a kicker's name, or for crowd cheering. An example for a visual cue that can be used is the act of

kicking.

To enable responding to the queries we design systems that perform distributed processing and enable dividing the processing in the system based on the available resources. We also study how to deploy machine learning models to edge devices and how to efficiently use the energy of mobile devices to run such algorithms. Our work focuses on optimizing for different objectives such as completion time, mobile device energy and network bandwidth.

1.1 Motivation and Challenges

Research in the area of deep learning remains an active topic [7], [8], and [9], and crowdsoursing information using edge devices is an active area of study [10] and [11]. Responding to queries in a wireless mobile system, and collecting images and videos that contain objects of interest becomes a more interesting and challenging topic as it requires (i) training the classification or detection models, (ii) deploying these trained models to the edge devices (iii) utilizing the limited computational and memory resources on the edge devices, and (iv) adapting to the network constraints.

Training deep learning models requires significant computational power; hence the use of CPU clusters or high performance Graphical Processing Units (GPUs). Given the nature of the computational intensive task, training CNNs is done on servers, and not on the edge devices. Training is time consuming. For example training a well known model called GoogLeNet from scratch takes a week on high-end GPUs [12], and transfer learning using a pre-trained model on a new small set such as CIFAR-10 [13] takes about three hours [14].

Another major challenge in training these models is the need for large datasets that include labeled samples. Different datasets such as ImageNet [15], MS COCO [16], PASCAL VOC [17], Sports-1M [18], UCF100 [19], and UrbanSound [20] are used for various tasks such as object detection, object classification, action recognition and audio classification. These data sets are widely used in the machine learning community, specially for bench-marking; they do not form an exhaustive list of all the real-world objects. As different applications are developed new samples need to be gathered and labeled to perform accurate training.

In order to perform inference after training the models, weights and parameters for the models have to be stored. Hence, storage and memory need to be considered, as well as the amount of bandwidth needed to transfer theses models. For example, in PyTorch [21]

GoogLeNet trained on ImageNet is 52.1 MB, and ResNet is 46.8 MB. As more inference is done on mobile devices, new models have been introduced that are smaller in size and less accurate, for example MobileNet [22] is 14.2 MB and SqueezeNet [23] is 5 MB. These models still remain significantly large so that we need to address memory and transmission issues when deploying the models to edge devices and downloading them in real-time.

As the computational power on mobile devices advance, videos and images are able to be processed. Despite these advances, mobile devices still remain slower than servers and inference using deep learning models on the edge is still considered slow. In [24] it is shown that the processing times on mobile GPUs such as NVIDIA Jetson TX1 compared to high performance GPUs such as NVIDIA Titan X Pascal, are significantly higher. Mobile devices such as hand held cell phones are even slower than mobile GPUs. In Chapter 3 we show that image classification on a mobile device is 40 times slower than the server.

Various applications such as intelligence operations, surveillance systems, forensics and emergency responses such as missing persons, query for information retrieved from images and videos. These systems and networks are highly dynamic and constrained, and require adaption to the environment. In our work we study how to respond to these queries while taking into consideration the previously mentioned challenges. In the next section we highlight our technical contributions and how we solve these problems.

1.2 Contributions and Dissertation Outline

In this work we focus on studying how to respond to information queries by adapting to the network constraints and resource availability. In our solutions we accommodate networks with limited bandwidth in the uplink and downlink. That limitation means we need to transfer data in an efficient manner. As processing on mobile devices is energy intensive and time consuming, reducing processing on devices is preferred to save on both battery and time. However, moving all the processing on the server may create a large backlog causing the server to become the bottleneck, as we will show in Chapter 3. In crowdsourcing, with the presence of mobile devices we are able to exploit the devices to reduce the burden on the server. Therefore, there are various trade-offs that we look into when responding to such queries. Specifically, we focus on four main goals: (1) responding to an image query in a wireless system by optimizing response time; (2) responding to an image query in an energy constrained network while also optimizing for response time;

(3) responding to an image query in a bandwidth constrained wireless network where devices are not trained to search for the queried object; and (4) responding to a video query in a wireless system by optimizing response time.

The following presents an overview of the flow of this dissertation and outlines the research problems we investigate.

- In Chapter 2, we discuss preliminary concepts and related works. We present an overview of image and video processing and how deep learning models such as convolutional neural networks (CNN) work. In addition, we discuss various crowdsourcing applications that have been previously implemented.

- In Chapter 3, we formalize the problem of data collection of a queried object in a distributed mobile network. The work aims to optimize for response time such that after a query is issued images that contain the object should be collected in a centralized server as fast as possible. We prove that the problem is NP-hard and propose a heuristic to solve it. We propose early exits from pre-loaded CNNs, or the usage of smaller less complex but less accurate CNNs as a filtering stage. The filtering stage is used in order to filter out images that have a high probability of not containing the object of interest; therefore saving processing time on devices. We show how devices make the appropriate choice of either processing images locally using early exits, or offload images to the server to be processed, or process images locally using the complete more accurate but costly CNN. These decisions are made dynamically as the devices make them based on the network conditions, and the estimation of the completion time of the device that it calculates locally based on the number of images stored on the device, the estimated processing times, and hit-rate. We compare the performance of the proposed heuristic to several baselines: processing all the images on the device locally then uploading the positively classified images to the server; all-offload where all the images are offloaded to the server and processed there; and a greedy algorithm that makes the choice to offload an image or process it locally based on the device's estimated completion time only, without considering any delay on the server side. We then compare our heuristic to a state of the art algorithm that was proposed for optimising completion time for labeling images [25]. However, that algorithm does not consider collecting the images that contain the object of interest, and only considers labeling. We show that our algorithm outperforms the aforementioned algorithms.

- In Chapter 4, we expand the problem formulated in Chapter 3 to include an energy

constraint. Each device dedicates a specific percentage of its battery to respond to the query. Devices make the decision to offload images to the server or locally process them based on the remaining energy in the device along with the other factors mentioned previously. We evaluate our algorithm on both LTE and WiFi and compare it to the time-heuristic proposed in Chapter 3 that does not take into account energy.

- In Chapter 5, we tackle a different type of query where the issuer is querying for a novel object that the pre-loaded models, on the mobile devices, are not trained to classify. In this scenario we need to consider different factors. First, we are not able to train new models due to the time required for training and the limited availability of samples of the new queried object. Second, the network has limited bandwidth which requires decreasing the amount of data that is transmitted.

  We propose a cosine-distance based algorithm to solve the novelty object image retrieval problem. The server calculates the closest base classes to the queried class based on the small number of available samples of the novel class. Base classes are the classes that the local models on the devices are previously trained to classify. After the server does the analysis, the query is sent to the devices indicating the closest base classes along with a threshold for each class, and other parameters. The thresholds enable the devices to classify an image as including an object that belongs to the novel queried class or the closest base class. We show in detail how the parameters are chosen on the server once a query is issued, and detail the data that needs to be transmitted in the network. We implement the system and show an in depth analysis and evaluation. We compare the algorithm to another algorithm that uses Mahalanobis-distance [26] and show that we transmit less data with a small F1 score trade-off. Our algorithm also enables the users to tune in different parameters to adapt to the network constraints and achieve a required accuracy.

- In Chapter 6, we address a third type of query where the requester is searching a large set of videos, stored locally on a mobile device, for a clip or video containing a specific action. Both visual and audio data are used to respond to this query. The mobile device has access to a video cloud where videos or audio segments are able to be offloaded for faster processing. The goal is to optimize for response time. We propose dividing the processing into different stages, where earlier stages act as filters and quickly discard clips before moving to complex stages -in terms of

both time and energy-. We show that the problem is NP hard, and we propose a heuristic. We demonstrate that the heuristic outperforms different baselines and state-of-the-art algorithms.

- Finally, in Chapter 7 we conclude the thesis and discuss future work.

Chapter 2
Preliminary Concepts and Related Works

In this chapter we start with a brief overview of deep learning, in particular the usage of convolutional neural networks in image and video processing. Following that we discuss different crowdsourcing systems that use vision applications and distributed processing over mobile networks.

2.1 Overview of Deep Learning and Image and Video Processing

Deep learning (DL), based largely on Artificial Neural Networks (ANNs) [9], is a type of Machine Learning (ML), and an approach to artificial intelligence (AI). The existence of multiple layers (depth) allows the machine to learn in multiple steps [7]. In [27] deep learning is defined as "A sub-field within machine learning that is based on algorithms for learning multiple levels of representation in order to model complex relationships among data. Higher-level features and concepts are thus defined in terms of lower-level ones, and such a hierarchy of features is called a deep architecture. Most of these models are based on unsupervised learning of representations".

Machine learning, as opposed to traditional programming methods, relies on a training framework where a system requires a large amount of training data from which the system learns. Training a neural network consists of of two phases: a forward pass and a backwards pass. In the forward pass a loss function is computed. In the backward pass the gradients of the learnable parameters are learned.

In our work we mainly use a class of deep neural networks known as Convolutional

Neural Networks (CNNs), commonly used for image and video applications. In the remaining of this section we will discuss CNNs in more detail, specifically for image and video processing, as those are the main applications we will use in our technical chapters.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a method of DL. By using a large amount of training data and high computational resources CNNs are able to achieve high accuracy in many computer vision applications such as image classification, object detection and semantic segmentation [9]. As a trade-off these models require more computational time and resources compared to traditional computer vision techniques. Traditional methods include feature extraction, such as SIFT [28], and Hough transforms [29], followed by machine learning algorithms such as Support Vector Machines (SVMs) [30] and K-Nearest Neighbours [31].

Originally proposed in [32] by Yann LeCun, CNNs gained more momentum in image classification when AlexNet [3] was published in 2012 and won the ImageNet [15] Large-Scale Visual Recognition Challenge (ILSVRC). In general CNNs consists of multiple layers that an image passes through before generating the final output. A classical CNN consists of convolutional layers, followed by an activation layer, pooling layers and finally a fully connected layer at the end. We will briefly describe the functionality of these layers. For an in depth description and more details on CNNs we refer the reader to Stanford's Convolutional Neural Networks for Visual Recognition [33].

There exists a number of filters (also known as kernels) that are used in each **convolutional layer**. The filters aim to detect specific features in the image, resulting in a feature (activation) map. The idea behind the usage of filters is that neighboring pixels in images are more correlated, and objects or features can appear in different locations in the image. Hence the idea of sliding the filters across the image/feature maps, and performing convolutions. For example Figure, 2.1 from [34], shows visualizations from filters in the first convolutional layer of AlexNet. These filters detect low-level features such as edges and color constrictive information. As we go deeper in the CNN higher level features are detected. For an RGB image with three channels, given a set of filters of size $w \times h \times d$, where w, h and d are the width, height and depth respectively ($d = 3$ for RGB images); the output of the convolutional layer is of size $w \times h \times f$ where f is the number of filters. Similarly, later convolutional layers in the network take the output of previous layers (activation maps) as input. We note that these filters are learned.

Activation functions are used to introduce non-linearity. The Rectified Linear

Figure 2.1. Low-level filters from AlexNet [34].

Unit (ReLU) is a commonly used activation function and is a threshold function at zero, $f(x) = max(0, x)$. **Pooling layers** are used for down-sampling and reducing the dimensions of feature maps. Max-pooling and average-pooling are commonly used.

Fully-connected layers take the results from the previous layer, and flatten them. The are called fully connected as all the inputs from the previous layer are connected to every activation unit in the next layer. This layer checks which high-level features correlate more strongly to a specific class. Fully-connected layers are usually present as the last layers in the CNN to form the final output. For classification we use an activation function after the fully connected layer such as *softmax* or *sigmoid*.

Deep 3-dimensional convolutional neural networks (C3D) were introduced in [35]. 3D CNNs retain temporal information for video sequences, making these features especially useful for action and activity recognition. Similar to 2D CNNs, C3D uses convolutional, pooling and fully connected layers, with an additional dimension representing the number of video frames used.

As our work deals more with applications related to image and video processing, for the remainder of the chapter we will focus more on object and action classification and recognition.

2.1.2 Deep Learning Frameworks

Due to the wide use of deep learning, and the increased research in the field, various open-source frameworks have been developed such as Caffe [36], TensorFlow [37], and Pytorch [38]. These frameworks aim to provide a level of abstraction and simplification for implementing complex deep learning models; they also provide easy integration and use of GPUs. In this section we discuss the three frameworks used for the systems proposed in this dissertation.

2.1.2.1 Caffe

Caffe is an open-source deep learning framework developed by the Berkeley Vision and Learning Center (BVLC) [36]. Caffe provides binding for Python and MATLAB, and can also be used on the command line. Caffe models are defined in plaintext protocol buffer (prototxt), making it easy for beginners to understand but requiring more lines of code to describe complex CNNs [39]. A Caffe model consist of Blobs, Layers and Nets. Blobs are N-dimensional structures that carry data and flow through the network. A Net is a set of connected layers that perform Forward/Backward computations. Layers are the building blocks of the models, and Caffe provides a catalogue of layers from state-of-the-art architectures.

Caffe is used by researchers in various applications such as semantic segmentation [40], object classification [41], and action recognition [42] and [35].

2.1.2.2 Tensorflow

Tenserflow is an end-to-end open-source platform for machine learning developed by Google [37]. Tenserflow provides APIs for Python and C++. Tensors are the building block for both Tenserflow and PyTorch. Initially, it is required to build a computational graph that consist of nodes and operations. The nodes take tensors as inputs, and produce tensors as outputs. After defining the graph and the tensors they are run in a session. However, eager execution, where operations are evaluated immediately without building graphs, was introduced in Tenserflow 2.0.

Tensorflow models are developed for various applications such as image classification [43], action recognition [44], and audio classification [45].

Tenserflow Lite [46] was developed to run some machine learning models on mobile and edge devices. After a model is trained using a GPU or CPU, they are converted and compressed using a converter before being deployed to the mobile or embedded device.

2.1.2.3 Pytorch

Pytorch is a Python version of the deep learning library Torch [38]. It is commonly used for its seamless integration with Python. Two kinds of data abstractions exist in Pytorch; variables and tensors. Tensors are the building block of PyTorch and can be easily moved between GPU and CPU [47]. Pytorch uses a data structure called computation graph. The nodes of the graph are variables and mathematical operations, and the edges are feeding input. Pytorch differs from other frameworks such as Caffe and Tensorflow in that its computation graphs are dynamic. In the general case, layers are usually expressed as Python classes, where forward methods are designed, and models are represented as classes, with multiple layers.

A large number of researchers use Pytorch for various computer vision applications such as image classification [48] and [49], object detection [50], semantic segmentation [51], and video classification [52] and [53].

2.1.3 Datasets

Since data is the main component of success for deep learning, various datasets have been collected and carefully labeled for various deep learning applications. In this section we will briefly highlight the datasets we use in our work.

CIFAR-10 and **CIFAR-100** [13] are very commonly used datasets for object classification. The CIFAR-10 dataset consists of $60,000$ labeled color images belonging to 10 classes. Each class has 6000 images. The data set is devided into training and testing sets. The training set consists of 5000 images per class, and the test set has 1000 images per class. On the other hand as the name suggests CIFAR-100 has 100 classes with 600 images in each class, divided into 500 images for training and 100 images for testing.

PASCAL-VOC (VOC2012) [17] initially introduced in 2005, is another popular dataset for image classification, object detection and segmentation. The dataset consists of 11540 annotated images belonging to 20 classes. VOC2012 dataset is divided into training and validation sets. The distribution of images among classes are not equal; and can be found in the documentation [54]. The last PASCAL VOC competition was in 2012 [55], but the dataset is still used as a benchmark.

ImageNet [15] is one of the most popular benchmark datasets for image classification and object detection. ImageNet is a large-scale dataset that consist of 1.3 million images belonging to 1000 categories. The large-scale of the dataset pushed the need for better deep learning models and solved the overfitting problem. Published in 2009, the dataset,

the year after, was used in an annual competition, ImageNet Large-Scale Visual Challenge (ILSVC), that ended in 2017 and resulted in various ground breaking CNN models such as AlexNet [3] and GoogLeNet [12]. However, even after the conclusion of the competition ImageNet remains as one of the main benchmarks for object classification and detection.

Kinetics-400 [56] is a video dataset that consists of 400 different human actions. The distribution of samples among classes is not equal. Each class has at least 400 samples, and each sample is approximately 10 seconds in length. The clips are taken from Youtube videos. The dataset is divided into training, validation, and testing sets. The training set contains 250 to 1000 clips from each class, the validation set is 50 clips per class, and the test set is 100 clips per class.

2.1.4 CNN Classification Models

In 2012, AlexNet [3] won the ImageNet Large-Scale Visual Challenge (2012 ILSVRC) and caused a boom in the deep learning field, as it significantly outperformed other methods by using a neural network that consists of five convolutional layers, maxpooling layers, dropout layers and three fully connected layers. Compared to today's CNN this model is relatively simple. However, the introduction of ReLU as a nonlinerity function was shown to reduce training time compared to tanh that was the standard at the time. The authors also used two GPUs, which meant a larger model can be trained with a decrease in training time. They also used two methods to reduce overfitting; data augmentation and dropout layers. The data augmentation techniques used are image translations, horizontal reflections and patch extractions. Overall, AlexNet was a game changer for the computer vision community as it paved the way for various more complex and accurate CNNs that followed in the years after. The Caffe implementation [57] reports a 57.1% top-1 accuracy, and a 80.2% top-5 accuracy on the validation set.

Following is a brief summary of the CNNs that we use in our research; the reported accuracy for the image classification CNNs is on the ImageNet dataset.

GoogLeNet [12] was the winner of ILSVRC 2014, with a 68.7% top-1 accuracy (31.3% error) and 88.9% top-5 accuracy (11.1% error) [58]. As apposed to previous CNNs, that stacked layers sequentially, GoogLeNet introduced the Inception Model. The inception model introduces parallelism, as shown in Figure 2.2 from [12]. The model introduced multiple filter sizes running in parallel, each extracting a different level of detail from the input. The addition of the 1x1 convolution reduced the depth of the volume. GoogLeNet is built from 9 inception models, with a total of 100 layers. They replace fully connected layers with average pooling, therefore saving on the number of

parameters. The size of the GoogLeNet Caffe model is 51.1 MB.

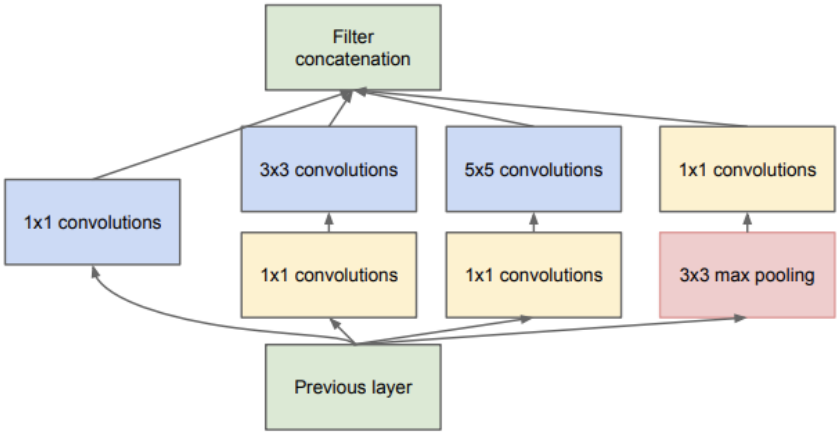


Figure 2.2. Inception module - GoogLeNet.

As researchers strived to develop CNNs with higher accuracy, models began to get deeper and more layers were added. This depth led to having large models that need more resources for both training and prediction. As these models recorded high accuracy, there was a need for smaller models for use on memory limited devices that are easily deployed on devices in networks.

SqueezeNet [23], introduced in 2016, was among the CNNs targeting devices with limited memory, and used on edge devices in networks. Given the need for fewer parameters, the design relied on making the majority of filters (1x1) convolution filters. It also introduced the squeeze layers, shown in Figure 2.3 from [23], to decrease the number of input channels. In order to increase accuracy given a smaller CNN, SqueezeNet delayed downsampling in order for convolution layers to have large activation maps. Squeezenet is constructed using a convolution layer followed by eight Fire modules, as shown in Figure 2.3, then a final convolution layer. Fire models are made of a squeeze layer followed by an expand layer. The number of 1x1 filters and 3x3 filters in those two layers are tunable, with the number of filters in the squeeze layer being less than the number of filters in the expand layer. By reducing the number of filters in the squeeze layer, the number of parameters are reduced as the volume entering the expand layer reduces. SqueezeNet has an accuracy of 57.5% in top-1 and 80.3% in top-5. The size of the SqueezeNet Caffe model is 5 MB [59].

ResNet [60], is a group of models that follow the concept of going deeper, and adding layers. However, it addresses the problem of vanishing gradients using ResNet blocks. The building block of ResNet models is the ResNet block, shown in Figure 2.4. The idea

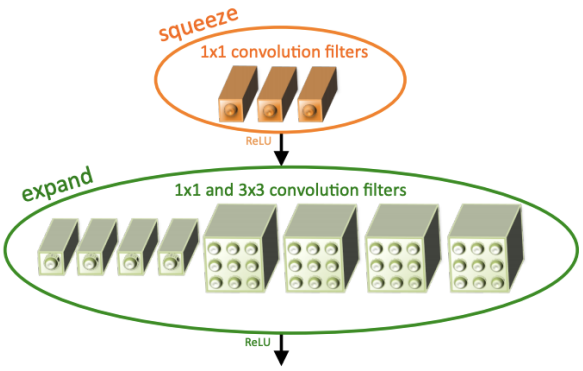


Figure 2.3. Fire module - SqueezeNet.

is that instead of the training trying to fit the underlying mapping, they fit a residual mapping instead. That is done using shortcut connections, such that each layer feeds into the next layer and directly into the layers 2-3 hops away, skipping one or more layers. Given an input x and a desired learned distribution $H(x)$, the residual $R(x)$ is defined as $R(x) = H(x) - x$. Rearranging that we get $H(x) = R(x) + x$. Given the shortcut connection we see that the layers are learning the residual. That enables the propagation of gradients to initial layers in a fast manner. Different variations of ResNets exist each with different depths, such as ResNet-32, ResNet-50, and ResNet-101. The accuracy of ResNet-50 using Caffe is 92.2% on top-5, with a model size of 102.5 MB [61].

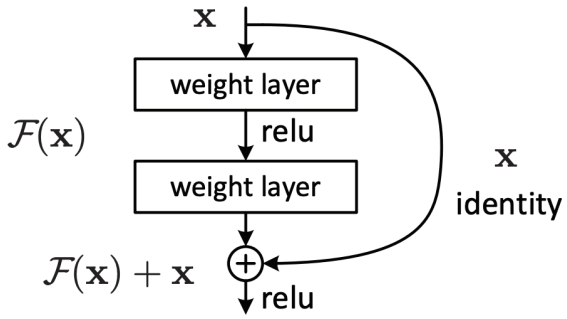


Figure 2.4. Residual block.

Two-Stream Inflated 3D ConvNet (I3D) [4], is used for action recognition from video streams. Trained on Kinetics dataset it reaches up to 74.2% top-1 accuracy and 91.3% top-5 accuracy. The idea in this architecture is inflating deep image classification networks (2D architectures) by using 3D filters and pooling kernels, making the filters cubic instead of squares. The two streams used are RGB stream and optical-flow stream. The two networks are trained separately, and at test time the predictions of the two

networks are averaged. The Tensorflow model size is 52.8 MB.

2.2 Image and Video Processing in Crowdsourcing Applications

With the large amount of data captured on mobile devices, researchers introduced Mobile Crowd Sensing, which is defined as, *"a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices and aggregates and fuses the data in the cloud for crowd intelligence extraction and human-centric service delivery [62]"*. For example in [63], a real-time tracking and surveillance system was proposed for analyzing images from various cameras. They partition processing between edge nodes and upload non-redundant videos, therefor utilizing network bandwidth.

Following the interest in utilizing mobile devices in using their data, researchers proposed moving processing to these devices. Deep learning models for mobile devices such as [23], [22] and [64] were created. Mobile CNNs take into account the limited memory and computational resources on mobile devices.

As mobile devices gained more power with advanced hardware and software, the use of more powerful servers to aid these devices in complex processing is still needed. Various techniques and systems for computational offload were introduced. Mobile Cloud Computing as defined in [65] is, the *"integration of cloud computing technology with mobile devices to make the mobile devices resource-full in terms of computational power, memory, storage, energy, and context awareness"*. In [66], an offload framework is proposed to enable devices to offload to multiple servers optimizing for energy consumption. CrowdVision [10], and NetVision [67] both propose cloud offloading in aiding mobile devices in processing videos. While [10] optimizes for a device's time and energy, [67] aims to minimize the system's query response time.

Lu et al. [67] aim to label all the videos in the system using deep CNNs. The objective is to minimize the maximum completion time of the nodes. After proving the problem is NP-hard they propose two algorithms to solve it. There is an option of either processing the video locally on the mobile device, or offloading it to a video cloud for processing. The first algorithm is a greedy algorithm that offloads a video from the edge device to a cloud in a way that minimizes the increasing time in the chosen cloud. In the online algorithm, the dynamic change in the transmission rates are considered. In the first part of our work we look at similar problems but rather than considering labeling only, we

consider collecting the positive data at a centralized server.

Chapter 3
Fast Image Search on Distributed Mobile Networks

Mobile devices collect a large amount of visual data that are useful for many applications. Searching for an object of interest over a network of mobile devices can aid human analysts in a variety of situations. However, processing the information on these devices is a challenge owing to the high computational complexity of the state-of-the-art computer vision algorithms that primarily rely on Convolutional Neural Networks (CNNs). Thus, we build PicSys, a system that enables answering visual search queries on a mobile network. The objective of the system is to minimize the maximum completion time over all devices while taking into account the energy consumption of mobile devices as well. In this chapter we discuss optimizing completion time. Chapter 4 discusses an energy-efficient system. First, PicSys carefully divides the computation into multiple filtering stages, such that only a small percentage of images need to run the entire CNN pipeline. Splitting such CNN computation into multiple stages requires understanding the intermediate CNN features and systematically trading off accuracy for the computation speed. Second, PicSys determines where to run each of the stages of the multi-stage pipeline to fully utilize the available resources. Finally, through extensive experimentation, system implementation, and simulation, we show that PicSys performance is close to optimal and significantly outperforms other standard algorithms [68].

3.1 Introduction

The wide spread use of camera-enabled mobile devices has led to the generation of large amounts of images and videos that are rich sources of information for critical applications like surveillance and monitoring [11], [69]. The computer vision community has made

headway in developing algorithms to extract information from visual data. Deep Neural Networks (DNN), specifically Multi-layer CNNs, have been shown to achieve high accuracy in visual analytics tasks such as image classification, detection, segmentation and activity detection [3], [40], [35]. However, this comes at a high computational complexity – often these algorithms push the limits of what can be achieved on current generation of mobile devices.

In this chapter, we focus on the specific problem of searching for objects of interest on a distributed network of mobile devices and a server in the cloud, in response to a query, which is issued in a centralized fashion. Images that contain the object of interest are gathered in a central server[1]. Responding to such queries requires: (i) identifying all mobile devices that are likely to contain the images of interest, (ii) sending queries, (iii) searching for the objects within the images taken by these devices and finally, (iv) uploading the images of interest to the central server. In this chapter, we propose PicSys to address these steps. [2]

PicSys consists of a number of mobile devices that have stored images, and a more powerful server machine, which we call the *cloud*, to which the mobile devices communicate over a wireless link. When a query is issued, it is expected that the issuer (analyst) will receive all the positively identified images that match the query description. The mobile device makes decisions as to where the image processing should take place, on the device or the cloud, in a distributed fashion. The goal of the system is to process all images as fast as possible, including uploading all images that contain the object of interest to the central server.

To help mitigate the low processing power and highly constrained energy resources of mobile devices, we introduce a processing pipeline on the mobile device which includes what we call an *accelerated* CNN. The accelerated CNN is a simple CNN that is faster, requires less energy, and is less accurate than a more complex CNN. It is used by the mobile device as a filter to remove images that likely do not have the object of interest from the processing pipeline. This reduces the burden on the mobile device of either offloading images for processing, or processing the full complex CNN locally.

The main contributions of our work are:

- We design the PicSys system which has the structure, via its multi-stage CNN

[1]The public has shown an increasing willingness to share images and videos with law enforcement in circumstances such as terrorist attacks and missing people. We assume voluntary participation, with mobile devices taking appropriate privacy measures [70].

[2]In our system, we consider only processing images from mobile devices, i.e., those images that are not already stored in the cloud.

pipeline, to support the design of algorithms that minimize query completion times.

- We perform extensive experimentation to determine processing times, and performance for various CNNs and pre-processing tasks to select the best set to meet our objectives.
- We formulate and solve an optimization problem to decide where to run each of the stages of the computation, so that we utilize the available computational resources properly.
- We design a heuristic to solve the problem.
- We present a detailed implementation of our system in a real testbed and illustrate experimental and simulation results for both a small case network as well as on large networks.

After implementing our system and using the task-level measurements to run a large scale simulation, the obtained results show that PicSys outperforms existing schemes, and that using the filtering stage in the mobile devices yields a 35% in time savings as opposed to when no-filtering stage is available.

The remaining part of the chapter is organized as follows. Section 3.2 provides an overview of image processing. Section 3.3 gives an overview of the model. In Section 3.4 we present experimental results that help in designing and evaluating the system performance. The optimization and algorithm are presented in Section 3.5, followed by the performance evaluation relying on both simulation and experimental results in Section 3.6. Section 3.7 reviews some related work. Finally, the chapter is concluded in Section 3.8.

3.2 Overview of Image Processing

In this section we present some background on image processing as it is required to understand this chapter. A full description and more details on image processing is shown in Chapter 2, we present some highlights in this chapter for readability.

Object classification refers to recognizing the objects appearing within images. Specifically, object classification algorithms return a probability distribution of certain objects being contained within an image. The higher the probability for an object, the more likely that object is in the image. The objects that may be searched for in an image are predefined by training the object classifier to recognize features of the objects of interest.

There are two main performance metrics considered with object classification: **recall** and **false positives**. Recall measures the number of images containing an object of interest that are returned considering the ground truth of total number of images that contain the object. Formally, it is expressed as $R = \frac{TP}{FN+TP}$, where TP and FN are true positives and false negatives of the object classification, respectively. False positives are the number of images that are positively classified that in fact do not contain the object of interest. The rate at which true positives are returned is also called the hit-rate.

We refer to the CNN as a classifier. It consists of a set of layers, starting with an input layer and ending with an output layer. Layers vary depending on the CNN architecture; some common layers are convolutions, max pooling, dropout, data augmentation, ReLU activations, etc. We mainly use GoogLeNet [12] and Squeezenet [23] in our experimentation. We refer the interested reader to [12] and [23] for more details about each layer in those architectures.

Because the classifier only returns a probability that an object is within an image, a threshold k is often defined which is used to label whether an image contains an object, or not. If the probability that an object is present in the image is kth highest or better out of all the objects detected, the image is labeled as containing the object. By setting a small value of k, only images that contain the object with high certainty will be labeled. This may miss some images that contain the object and will result in a lower recall and a lower false positive. By setting a large value of k, the results will be more inclusive and will lead to a higher recall, but also to a higher false positive.

State-of-the-art classification algorithms are based on deep CNNs, such as Alexnet [3], GoogLeNet [12], VGGNet [71] and ResNet [60]. Many CNNs, designed for classification, end with a Softmax layer. The Softmax layer assigns a probability for each class the CNN was trained on (with total sum being equal to 1). That probability is the likelihood of an image containing an object that belongs to that particular class. We use the term top-k to refer to the top k predictions that have the highest probabilities. In general, the best results are achieved when the full CNN is processed, but tradeoffs can be made to allow lower classification performance with lower processing times by processing fewer layers of the CNN.

3.3 PicSys Overview

In PicSys, operation starts when mobile devices receive a query from a centralized source requesting images containing an object of interest. This can be done in a similar way to

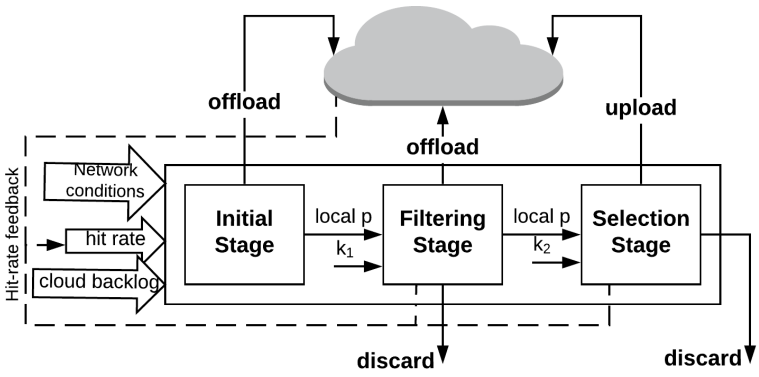


Figure 3.1. Overview of the system on the mobile device.

subscription alert services.

Images can be processed on mobile devices or the cloud. The cloud has high computational power and a persistent power supply, and as such enables the use of more accurate models that complete processing in a shorter time period. Mobile devices are constrained by their computational power. PicSys uses a *multi-stage decision technique* to reduce the overall processing time while meeting a specific recall requirement.

Fig. 3.1 illustrates the mobile device's three-stage decision process. In the first stage, the *Initial Stage*, the mobile device makes a decision to either (i) process the image locally, or (ii) offload it for processing on the cloud. That decision is based on the network conditions, cloud backlog and the hit-rate estimate. The hit-rate estimate is continuously updated based on recent historical data. If the hit-rate is very high, it may be better to offload the picture to the cloud for processing as, ultimately, images of interest need to be uploaded anyway However, note that *even if the hit-rate is high, not all images will be uploaded as that may overwhelm the cloud which serves multiple mobile devices simultaneously.* If images are uploaded to the cloud, the remainder of the processing takes place there.

If local processing is chosen in the first stage, then the *Filtering Stage* is executed on the mobile device. The Filtering Stage involves a computationally efficient, but less accurate algorithm, which we refer to as the *accelerated* CNN. It extracts coarse-grained features from the images, based on which a decision is made as to if the image likely contains the object of interest. depending on whether the top-k_1 requirement is met or not. If not, the image is discarded. If so, the mobile device decides to (i) complete processing locally by executing the *Selection Stage*, or (ii) offload the image to the cloud for further accurate processing.

In the Selection Stage on the mobile device, a more complex and accurate CNN is

executed which achieves more accurate results albeit at a higher cost. After processing, one of two decisions is made: (i) upload the image to the requester if the image is likely to contain the object, or (ii) discard the image.

Further, the system enables the interaction between mobile devices and the cloud. In order to avoid increased communication complexity, there is no interaction between the mobile devices.

Most crowdsourcing applications, such as law enforcement or intelligence gathering require a person to inspect a photo, and virtually all military applications require a person to verify an image, so a full high quality image is needed. We choose to upload the image with its original size for the requester to have access to it in case further processing or analysis needs to be performed. When offloading the image to the cloud for processing we also choose to send the original sized image to avoid the need to re transmit the original image if the object of interest ends up being present. Offloading the scaled version of the image and then re-transmitting it would result in a different problem setup, and is beyond the scope of this work.

It should be mentioned that PicSys tunes the parameters of the different stages of the pipeline based on the recall requirement of the application. In PicSys we have to pick two k values - namely k_1 and k_2 (as explained in Section 3.4.3) for Filtering and Selection Stages, respectively.

3.4 Experiments and Design Decisions

This section elaborates the PicSys design choices backed by extensive experimentation. We use the experimental results to drive the simulations. In the following experiments we use *Caffe* [36], a deep learning framework, to train and test our CNNs. The *gpu flag* is used to run the models on NVIDIA GTX TITAN X 12 GB, modeling the cloud. For implementation on mobile devices we use the *caffe-android-lib* [72] on Samsung Galaxy S9. The two labeled data sets we use for classification are ImageNet [15], ILSVRC2012, with 1000 classes, and VOC2012 [17], with 20 classes.

To get the processing times, we used a small set of 100 images, captured from a Galaxy S9. For the ImageNet data set we used the trained models from Model Zoo [73]. For VOC2012, we fine-tune the CNN to classify the 20 classes.

3.4.1 Image pre-processing

To use Caffe, images must be scaled to the particular dimensions required by the CNN. This scaling may be done internally by Caffe, or by pre-processing. We find that pre-processing the image for scaling using FFmpeg [74] instead of Caffe has two benefits: (i) the scaling time is a linear function of image size as shown in Fig. 3.2, and the processing time after scaling is constant, as shown in Fig. 3.3; (ii) more importantly, scaling using FFmpeg and then processing the image is faster than pre-scaling on Caffe, as shown in Fig. 3.3.

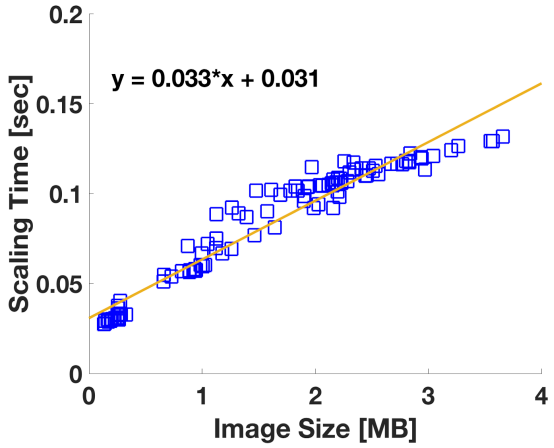


Figure 3.2. Scaling time using FFmpeg.

These realizations help make the offloading decisions in Section 3.5, and serve as the argument to choose pre-processing the images using FFmpeg.

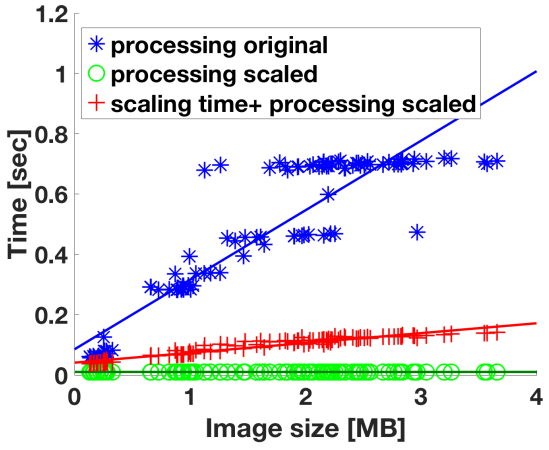


Figure 3.3. Processing time for images.

3.4.2 Choosing the deep learning models and model partitioning

Three factors need to be considered when choosing the deep learning models: recall, processing time and energy consumption. The requirement for the accelerated CNN is to have a faster processing time with lower energy expenditure than the expensive CNN. In this section we study the three factors in more detail as we explore our two choices for the accelerated CNN: a truncated version of the expensive CNN, i.e., terminate the processing after only a subset of layers are complete, or a separate, simpler full CNN such as SqueezeNet.

In the first set of experiments, we compare the recall of GoogLeNet, SqueezeNet and a compressed version of GoogLeNet. We use the publicly available Caffe models of GoogLeNet and SqeezeNet trained on ILSVRC2012 [73]. We leverage the original design of GoogLeNet and use the existing Softmax layer available in an early stage after a layer called *inception_4a*; we use this subset of the network and call it 'GoogLeNet_i4'.

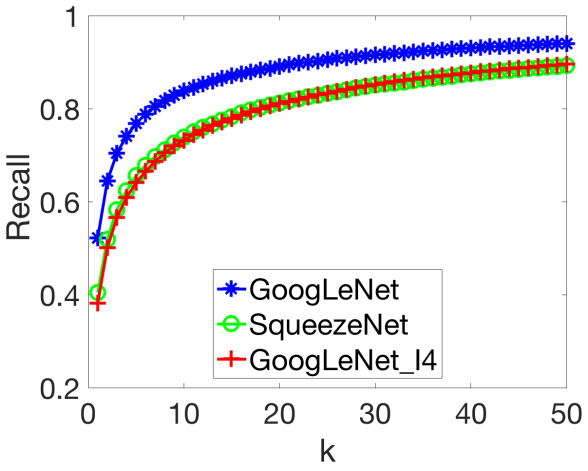


Figure 3.4. Recall-ILSVRC2012.

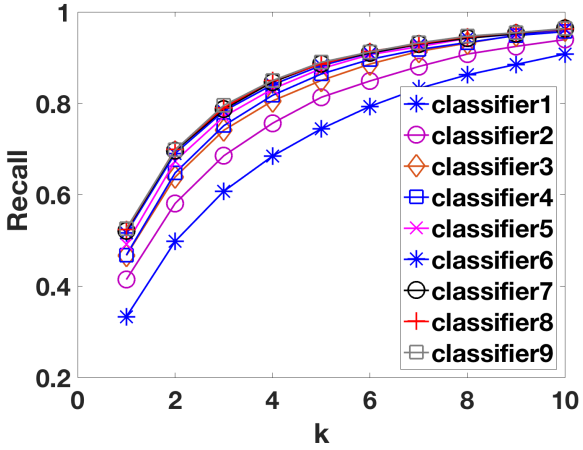


Figure 3.5. Recall- per classifier.

Fig. 3.4 shows the recall vs. top-k, where top-k is the highest k probabilities (classes) from the classifier, as explained in Section 3.2. The results are obtained using 25k images from the ILSVRC2012 validation set. We see that running the complete GoogLeNet model yields higher recall than running a subset of it or a more compressed model, such as SqueezeNet. We also observe that choosing either SqueezeNet or GoogLeNet_i4 will give approximately the same recall.

Note that with the recent development in compressed CNN models and the large amount of research on deploying these models on mobile devices [22], [64], [75], the accelerated model and the expensive model can be exchanged with other models as the

user sees fit. If using a deep learning framework that enables dynamic computation on CNNs, choosing the accelerated CNN as a subset of the expensive CNN is beneficial, and enables the download of a single model on the device.

In the second set of experiments we show more details on partitioning complex CNNs. The work in [76] shows processing times for full CNNs as well as for individual layers. For our work, in addition to the processing time for each layer, we require the addition of a classifier after the layer where partitioning takes place. To get more insights, we fine-tuned GoogLeNet on VOC2012 by adding various classifiers after inception layers. We add *Pooling, Dropout, Classifier*, and *Softmax* layers after each inception *"Concat"* layer, summing up to nine classifiers.

Fig. 3.5 shows the recall after each classifier. The processing time on the mobile device increases from 0.3 to 0.4 sec as the number of layers increases. The higher the number of layers used in the forward pass, the higher the recall. However, the increase is not linear and the difference gets smaller the further into the network we process. The choice of the classifier is based on a tradeoff between accuracy and computation cost.

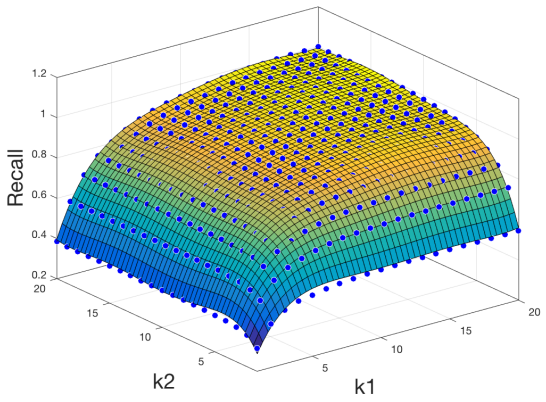


Figure 3.6. Aggregate recall on VOC2012.

3.4.3 Choosing the k_1 and k_2 values

The two processing phases may be filtered using different k values. k_1 is used for the accelerated CNN and k_2 is used for the expensive CNN. The expensive CNN only receives and processes images that pass the accelerated CNN's top-k_1 requirement. Thus, the loss in recall from the Filtering Stage cannot be regained regardless of how high k_2 is set. However, setting the value of k_1 too high may result in unnecessary uploads or expensive CNN processing for the mobile device because the accelerated CNN will not

filter out some of the images that do not contain the object. Fig. 3.6 shows the recall on VOC2012 with 20 classes. We see that a higher recall is reached as the values of k_1 and k_2 increase. Using Fig. 3.6, we pick k_1 and k_2 values based on the recall requirement of the application. In Section 3.6 we choose $k_1 = 5$ and $k_2 = 3$.

3.5 PicSys Optimization

In this section we present the formal optimization problem for allocating resources in PicSys, and a heuristic algorithm that come close to achieving optimal performance. Ultimately, the goal is to process all images and have those with objects of interest uploaded to the cloud as fast as possible.

We first show that the problem is a nonlinear optimization. We then propose heuristics to solve the problem, which achieve running times close to optimal.

3.5.1 Mathematical Formulation: no-energy constraints

We assume that initially every image is stored in one of the devices in the system. Devices process images in series, i.e., they perform scaling and local processing consecutively. However, scaling and local processing can take place while other images are being transmitted or are waiting in the transmission queue. In the following, we use M to represent the set of mobile devices in the system ($m \in M$), and I to denote the set of images in the system, i.e., $I = \{I_1, I_2, \ldots, I_{|M|}\}$.

We assume that the transmission rate r_m, and the top-k values are given, and that k_1 and k_2 are set to meet the recall requirement. We also assume that the mobile devices may be heterogeneous, and after an image is scaled, the processing time for that image on any mobile device is $c^p_{i,m}$, whereas the processing time on the cloud is c^p_u.

Let q be the queried class; we define two binary variables: $z^{k_1}_{i,m}$ and $z^{k_2}_{i,m}$. $z^{k_1}_{i,m}$ is 1 if q is in the top-k_1 classes after accelerated processing, and 0 otherwise. Similarly, $z^{k_2}_{i,m}$ is 1 if q is in the top-k_2 classes after expensive processing, and 0 otherwise. Table 5.1 summarizes the notation used throughout this chapter.

3.5.1.1 Components of Delay

In our problem formulation, we are accounting for the following delay components:
Scaling delay, $c^s_{i,m}$, is the time required for the input image i on device m to be scaled to the input size of the CNN.

Table 3.1. Nomenclature and Notation

$c^s_{i,m}$	scaling time of image i on device m
c^p_u	processing delay of an image on the cloud
$c^p_{i,m}$	processing delay of image i on device m
$c^t_{i,m}$	transmission delay of image i on device m
α_m	processing time for accelerated CNN on device m
β_m	processing time for expensive CNN on device m
r_m	transmission rate for device m
$B_{i,m}$	size of image i on device m
$z^{k_1}_{i,m}$	truth value of image i, stored on device m, after accelerated processing
$z^{k_2}_{i,m}$	truth value of image i, stored on device m, after expensive processing
$cost1_{i,m}$	cost of direct offload
$cost2_{i,m}$	cost of accelerated processing followed by offload
$cost3_{i,m}$	cost of accelerated processing followed by discard
$cost4_{i,m}$	cost of accelerated processing, then expensive processing followed by upload
$cost5_{i,m}$	cost of accelerated processing, then expensive processing followed by a discard
T_{max}	completion time for all images (makespan)
$t^{completion}_{i,m}$	completion time of image i that is on device m
$t^{start}_{i,m}$	start time of image i on device m
$x1_{i,m}, \ldots, x5_{i,m}$	decision variables
$l^t_{i,m}$	transmission queueing delay of image i on device m
$l^q_{i,m}$	cloud queueing delay for image i from device m

Processing delay, $c^p_{i,m}$, is the sum of processing times for image i that take place on device m. Since we have two stages where local processing can take place, namely, accelerated CNN and expensive CNN, we need to account for both delays when considering processing delay. Therefore, $c^p_{i,m} = a\alpha_m + b\beta_m$, where $a \in \{0, 1\}$, $b \in \{0, 1\}$, whereas α_m is the processing time for the accelerated CNN on device m, and β_m is the processing time for the expensive CNN on the same device (m).

Note. An image that is uploaded to the cloud can only start processing after it has been completely uploaded, and when a server is free. Both the cloud and mobile devices can start processing while sending or receiving images.

Transmission delay, $c^t_{i,m}$, is the time for the image to complete transmission. For simplicity, we assume that each mobile device has a dedicated channel, hence devices are able to transmit simultaneously. This assumption is realistic in many situations where

participants are geographically dispersed or access the network via Wifi, which is likely in cases in which a large number of images would be uploaded. The study of the impact of congestion on the wireless links is left for future work. The cloud receives images from many sources, as multiple devices may be uploading images concurrently. However, a single mobile device transmits images sequentially. Thus, the transmission delay of image i from device m to the cloud is calculated as $c^t_{i,m} = \frac{B_{i,m}}{r_m}$, where r_m and $B_{i,m}$ represent the transmission rate and image size, respectively.

Transmission queueing delay, $l^t_{i,m}$, is the time image i spends on device's m transmission queue after it is scheduled to be offloaded/uploaded until it starts transmission.

Cloud queueing delay, $l^q_{i,m}$, is the time the offloaded image i (originating from device m) spends in the cloud queue from the moment it arrives until a server is available to process it. Note that both $l^t_{i,m}$ and $l^q_{i,m}$ depend on the decisions that have been made at previous time slots, and on the number of images in the queue.

3.5.1.2 Formulation

The goal is to minimize the system completion time given a minimum recall requirement. A system is said to have completed its tasks when all the images in it have been processed, and all images that are positively classified by the CNNs reach the cloud. Therefore, *the completion time of the system is the maximum of completion times of all the images in the system.* Thus, we have the following problem:

minimize T_{max}

subject to:

$$T_{max} \geq t^{completion}_{i,m}, \qquad \forall i \in I, \forall m \in M, \tag{3.1}$$

$$x2_{i,m} + x3_{i,m} + x4_{i,m} + x5_{i,m} \leq 1, \qquad \forall i \in I, \forall m \in M, \tag{3.2}$$

$$\sum_{j=2}^{5} xj_{i,m} - x1_{i,m} = 0, \qquad \forall i \in I, \forall m \in M, \tag{3.3}$$

$$x2_{i,m} - z^{k_1}_{i,m} x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{3.4}$$

$$x3_{i,m} - (1 - z^{k_1}_{i,m}) x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{3.5}$$

$$x_{4i,m} - z^{k_2}_{i,m} x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{3.6}$$

$$x5_{i,m} - z^{k_1}_{i,m}(1 - z^{k_2}_{i,m}) x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{3.7}$$

$$xj_{i,m} \in \{0, 1\}, \qquad \forall j \in \{1, 2, 3, 4, 5\}, \forall i \in I, \forall m \in M, \tag{3.8}$$

where $t_{i,m}^{completion}$ is the completion time of image i that is on device m, and is a function of the decision variables. It can be calculated as $t_{i,m}^{completion} = t_{i,m}^{start} + cost_{i,m}$. In this equation, $t_{i,m}^{start}$ and $cost_{i,m}$ are start time for image i on device m and the total cost for processing image i, respectively. We have

$$t_{i,m}^{start} = t_{i-1,m}^{start} + x1_{i-1,m}(c_{i-1,m}^{s} + \alpha_m) + x4_{i-1,m}\beta_m + x5_{i-1,m}\beta_m, \tag{3.9}$$

and

$$cost_{i,m} = (1 - x1_{i,m})cost1_{i,m} + x2_{i,m}cost2_{i,m} + x3_{i,m}cost3_{i,m} + x4_{i,m}cost4_{i,m} + x5_{i,m}cost5_{i,m}. \tag{3.10}$$

The binary decision variables $x1_{i,m}$, $x2_{i,m}$, $x3_{i,m}$, $x4_{i,m}$, $x5_{i,m}$ are defined as follows. $x1_{i,m}$ is equal to 0 if image i on device m is offloaded in the first stage, and 1 otherwise. $x2_{i,m}$ is equal to 1 if image i on device m is passing through accelerated CNN and then offloaded to the cloud, and 0 otherwise. $x3_{i,m}$ is 1 if image i on device m passes accelerated CNN and is discarded, and is 0 otherwise. $x4_{i,m}$ is 1 if image i on device m passes expensive (after it has passed through accelerated) CNN, and is uploaded to the cloud afterwards, and 0 otherwise. $x5_{i,m}$ is 1 if image i on device m is discarded after passing expensive CNN, and 0 otherwise.

Note. To reduce the number of equations required to convey the constraints of the system, the decision variable $x1_{i,m}$ is defined differently when compared to $x2_{i,m}, \ldots, x5_{i,m}$. The cost of each decision is defined as:

$$cost1_{i,m} = l_{i,m}^{t} + c_{i,m}^{t} + l_{i,m}^{q} + c_{u}^{p},$$

$$cost2_{i,m} = c_{i,m}^{s} + \alpha_m + l_{i,m}^{t} + c_{i,m}^{t} + l_{i,m}^{q} + c_{u}^{p},$$

$$cost3_{i,m} = c_{i,m}^{s} + \alpha_m,$$

$$cost4_{i,m} = c_{i,m}^{s} + \alpha_m + \beta_m + l_{i,m}^{t} + c_{i,m}^{t},$$

$$cost5_{i,m} = c_{i,m}^{s} + \alpha_m + \beta_m.$$

Constraint (1) shows that the completion time of the system is greater or equal to the completion time of every image in every device (be it processed on the device or the cloud). Constraint (2) ensures that only one of the decision variables $x2_{i,m}, \ldots, x5_{i,m}$ could have value 1 at a time. Constraint (3) ensures that if an image is not offloaded

in the first stage, it will be processed locally, and if it is offloaded in the first stage, there will be no local processing. Constraints (4)-(7) represent the requirements for accelerated CNN and offload, accelerated CNN and discard, expensive CNN and upload, and expensive CNN and discard, respectively. Constraint (8) shows that the decision variables are binary.

As can be seen, the problem formulation is representing a nonlinear optimization problem, since the objective is a function of costs multiplied by decision variables $x1_{i,m}, \ldots, x5_{i,m}$, where the costs are also functions of $l^t_{i,m}$ and $l^q_{i,m}$. If we simplify the problem by assuming a powerful cloud with infinite servers (no cloud queueing delay), and the transmission times for each device are constant, still the transmission queueing delay remains, introducing thus non-linearity. Relaxing this delay component is not possible because it yields to the unrealistic assumption of infinite bandwidth. Therefore, the transmission queueing delay is non-avoidable and is the cause to the non-linearity of the problem. As a result, the problem cannot be formulated as an integer linear program, which could be solved by the CPLEX optimizer. In the following section, we propose a heuristic to solve our scheduling problem with no energy constraints.

NP-hardness of the problem. We prove the NP-hardness of our problem by reducing the well-known NP-hard job-shop scheduling problem [77] to our problem. In the original version of job-shop scheduling problem, we have n jobs $j_1, j_2, \ldots, j_n$ of varying processing times, which need to be scheduled on M machines with varying processing power while trying to minimize the makespan (the total length of the schedule or total time taken to complete all jobs). We will show next that the job-shop scheduling problem is a special case of our problem.

Considering the special case where the communication delay of images is zero and the truth values of processing times on every machine are known *a priori*, the data collection scheduling problem can be seen as a generalization of job-shop scheduling with the following assumptions. The set of all possible images to be processed represents the set of jobs, and each decision variable $x1_{i,m}, x2_{i,m}, \ldots, x5_{i,m}$ in our problem represents one machine in the job-shop scheduling problem. Therefore, the total of $5 * I * M$ machines exist in the job-shop scheduling problem given these assumptions, as for each image $i \in I$ there exist 5 different ways of processing it on a machine, and each way could be seen as a separate machine in the context of the job-shop scheduling problem. Also, assuming that the constraints on decision variables correlation in our problem are relaxed, then the problem defined in Eq.(1)-(8) can be seen as a simple job-shop scheduling problem, and thus, our problem is more general than the standard job-shop scheduling problem.

Thus, data collection scheduling problem is NP-hard.

3.5.2 Proposed Heuristic: no-energy constraints

Given that the optimization problem presented earlier is NP-hard, we resort to heuristic algorithms in order to be able to find a solution to this problem.

As mentioned in Section 3.3, PicSys consists of multiple stages. Initially, a decision is made to either offload the image immediately to the cloud, or go through the Filtering Stage, where accelerated processing takes place. At the end of this stage, the device decides either to offload the image to the cloud or to complete the processing locally. If the latter is the case, the image undergoes the Selection Stage with expensive processing. There are two possible outcomes from the last stage: (i) upload the image to the cloud, with no further processing needed there, or (ii) discard the image.

3.5.2.1 Estimating the cloud completion time

First, the device sends a *request packet* to the cloud and in response gets the cloud queue size, q_c. The device has knowledge of the number of images in its local transmission queue, q_t, and can accordingly calculate image i's transmission queueing delay $l^t_{i,m}$. Based on these values, the device estimates the completion time of the head-of-the-queue image if it is offloaded to the cloud. This estimate is given by:

$$e_c = l^t_{i,m} + c^t_{i,m} + (q_t + q_c + 1) * c^p_u. \tag{3.11}$$

Note that the term q_c is the source of uncertainty in e_c. Its value can change from the moment the device sends the request packet to the moment when the image arrives at the head of the transmission queue, e.g., other images from other mobile devices have arrived in the meantime and/or the cloud has already processed the images that have been there.

3.5.2.2 Estimating the device completion time

The device estimates its completion time based on both its hit-rate and the number of images remaining in its decision queue. A device is said to have completed its processing when the last image in the decision queue is processed or when the last positive image is offloaded/uploaded.

Let the number of images in the device's queue be q_m, and let the hit-rate of the queried object in the given device be h_m. Assuming k_1 and k_2 are chosen such that the

difference in recall between the two stages is not high, the estimated completion time of the device m when $c_m^t > c_m^s + \alpha_m + \beta_m$ can be calculated as follows:

$$e_m = (1 - h_m) * q_m * (c_m^s + \alpha_m) + c_m^s + \alpha_m + \beta_m + h_m * q_m * c_m^t. \tag{3.12}$$

If for the average transmission time of images it holds that $c_m^t \leq c_m^s + \alpha_m + \beta_m$, then

$$e_m = (1 - h_m) * q_m * (c_m^s + \alpha_m) + h_m * q_m * (c_m^s + \alpha_m + \beta_m) + c_m^t. \tag{3.13}$$

In both aforementioned estimates, we account for the worst case where the hits appear at the end of the local queue. Since in Eq.(3.12) the transmission delay is the largest delay component, the term $c_m^s + \alpha_m + \beta_m$ is not multiplied by the number of images with hits, but appears only once (corresponding to the first image with a hit). Likewise, in Eq.(3.13), c_m^t is not multiplied by the number of images with hits because the largest delay component is $c_m^s + \alpha_m + \beta_m$.

When processing everything on the mobile device, the worst case scenario is when the decision, for every image, occurs only after the second stage. There are two extreme cases corresponding to this scenario. In the first one, the transmission time c_m^t is much higher than $c_m^s + \alpha_m + \beta_m$, in which case the completion time is approximately $h_m * q_m * c_m^t$, similarly to Eq.(3.13). In the second extreme case, the values of $c_m^s + \alpha_m + \beta_m$ are much higher than c_m^t, and the completion time would have as the dominant component the term $q_m * (c_m^s + \alpha_m + \beta_m)$, similarly to Eq.(3.12).

For all the more common non-extreme cases corresponding to this scenario, we propose the following approximation for the completion time:

$$e_m = q_m * (c_m^s + \alpha_m + \beta_m) + w_m * h_m * q_m * c_m^t, \tag{3.14}$$

where w_m is a "correction factor", whose value is a function of c_m^t, c_m^s, α_m, and β_m. In almost all cases the value of w_m is 0 or very small. Only when $c_m^t >> c_m^s + \alpha_m + \beta_m$, the value of w_m is 1.

Note. In Eq.(3.13)-(3.14), we drop the i index for $c_{i,m}^s$, as it represents the *average* scaling delay of an image on that device. Similarly, c_m^t represents the *average* transmission time of an image from device m to the cloud.

3.5.2.3 Heuristic

We define the heuristic as follows:

- For each image in a device's local queue, the Initial stage and the Filtering stage perform the following:

    1. The device calculates the estimated completion time of the cloud e_c, based on the value of q_c it receives, using Eq.(3.11).
    2. The device calculates its local estimated completion time e_m using Eq.(3.12) or Eq.(3.13).
    3. The device calculates the speedup ratio $\eta = \frac{e_c}{e_m}$. If $\eta < 1$ the image is offloaded to the cloud, else it is processed locally.
    4. Every image that reaches the Selection stage and is labeled as a hit is uploaded to the cloud.

The logic for each stage is described in Algorithms 1-2. The complete heuristic is outlined in Algorithm 3. In the remainder of the chapter we refer to this as the **time-heuristic.**

Algorithm 1: INITIAL_STAGE_DECISION ()

```
1 calculate η ;
2 if η < 1 then
3 |   offload image;
4 else
5 |   accelerated-local processing;
6 |   FILTERING_ STAGE_DECISION ();
7 end
```

Note. Hit-rate is used to estimate the completion times, and the completion times are used to make the decision on the steps to follow in the system by each mobile user. When it comes to using the hit-rate feedback, how often the system sends a feedback to a user depends on how stable that prediction is. In this chapter, we use an average for the hit-rate.

3.5.3 Cloud tokens

When evaluating the performance of PicSys it becomes apparent that at the start of a query every device views the cloud as being unloaded, i.e., has a snapshot of a small q_c, and starts by offloading too many images. This leads to overloading the server, especially in cases where the number of devices is large, and the number of images on each device is

Algorithm 2: FILTERING_ STAGE_DECISION()

```
if class of interest in top-k_1 then
    calculate η ;
    if η < 1 then
        offload image;
    else
        expensive-local processing;
        SELECTION_ STAGE_DECISION ();
        // if the class of interest in top-k_2 then upload image, else
           discard it
    end
else
    discard image;
end
```

Algorithm 3: Decision process on Mobile Device

```
while there are images in the decision queue do
    if l^t_{i,m} ≥ c^s_{i,m} + α_m then
        accelerated-local processing;
        if class of interest in top-k_1 then
            if l^t_{i,m} ≥ β_m then
                expensive-local processing;
                if class of interest in top-k_2 then
                    upload to cloud;
                else
                    discard image;
                end
            else
                FILTERING_STAGE_DECISION ();
            end
        else
            discard image;
        end
    else
        INITIAL_STAGE_DECISION ();
    end
end
```

small[3]. To prevent this, at the beginning of the process we calculate a number of tokens and assign them to each device to control their upload rate. The device will only make the decision to offload if there is an available token, otherwise it will do local processing. If $|I_m|$ is the total number of images in device m, the number of tokens is calculated as

$$num\_tokens = \frac{max_{m \in M} |I_m| \alpha_m}{c_u^p}. \tag{3.15}$$

The usefulness of tokens in practice is elucidated later in Section 3.6.2.3.

3.5.4 Complexity

The running time of the algorithm on each device increases linearly with the number of images, i.e., it is $O(n)$, where n is the number of images on the device. The reason for this is that in every step of Algorithms 1-3 only constant number of operations are performed.

3.6 Experimental Evaluation

We evaluate our algorithm using both experiments and simulations. Due to the complexity, we are only able to find the optimal solution for cases with a small number of users.

In order to evaluate our heuristics, we introduce an idealized *look-ahead* algorithm, which has future knowledge of the truth values for every image as well as exact processing and scaling times. We assume the look-ahead algorithm acts as an *oracle*, which is able to correctly predict the overall outcome of all the images. Note that the performance of the *look-ahead* algorithm is not achievable in practice. Unless stated otherwise, we will be using the same CNNs as those described in Section 4. In this section, all processing times are based on our experimental results.

3.6.1 Baselines

We compare our heuristic with the optimal, the look-ahead algorithm, and the following three standard algorithms:

- **All-device:** All images are processed locally on the mobile device where they are

[3]Since distributing the tokens makes sense only for a small number of images on each device (for a large number of devices), the energy consumption on every device is negligible in these scenarios. As a result, this approach is oblivious to the residual energy left on devices for crowdsourcing.

originally stored. Then, every image that is positively classified is uploaded to the cloud.

- **All-cloud:** All images are immediately offloaded to the cloud for processing.
- **Greedy:** Every mobile device makes a greedy choice of whether to process the image at the head of its queue locally, or offload it based on the device's estimated completion time, and the completion time of the image if offloaded, assuming the image does not incur any queueing delay on the cloud.

3.6.2 Simulation Results

In this subsection, first, the performance of the time-heuristic is compared to the optimal solution for a small system. To do this in a controlled environment, we use a simulation. Then, we present the details of the testbed implementation, which is succeeded by results on a larger system via simulation, using values obtained from our experiments. We then proceed with comparisons to the case where the system is heterogeneous. After that we compare to no-filtering stage algorithms.

As a final note, unless stated otherwise, we assume a system of homogeneous devices in the scenarios to follow in this section. Hence, in those cases, we will mostly skip using the index m to simplify the notation.

3.6.2.1 Comparison to optimal

We start by comparing our heuristic to the optimal solution for a small system of three devices. Because this is a simulation of a small number of devices and images, we only use the time-heuristic and do not consider energy. The system parameters are as follows: $|M| = 3$, $|I| = \{10, 4, 6\}$, and $h = \{0.2, 0, 0.5\}$ for each device, respectively. The other parameters are $\bar{B}_i = 800$ kB, $r_m = 20$ Mbps, $\alpha = 0.3$ sec/img, $\beta = 0.6$ sec/img, $c_u^p = 0.01$ sec/img, and $c_i^s = 0.05$ sec/img. Assuming 0 is a miss and 1 is a hit, an example of truth values for each device are: Dev-1: {0,0,0,0,1,0,0,0,1,0}, Dev-2: {0,0,0,0}, Dev-3: {0,0,0,1,1,1}. Table 3.2 illustrates the completion time for all the algorithms.

Table 3.2. Completion time [sec]

opt	look-ahead	heuristic	greedy	all-device	all-cloud
1.74	1.74	1.83	1.98	4.5	2.86

Table 3.3. Decision trace optimal

d#	decisions										[sec]
1	i-off	i-off	i-off	i-off	i-off	f-disc	f-disc	f-disc	i-off	f-disc	1.68
2	i-off	i-off	i-off	i-off							1.12
3	i-off	i-off	i-off	i-off	i-off	s-up					1.68
	cloud completion time										1.74

Table 3.4. Decision trace heuristic

d#	decisions										[sec]
1	i-off	i-off	f-disc	i-off	f-off	f-disc	i-off	f-disc	i-off	f-disc	1.75
2	i-off	i-off	f-disc	f-disc							0.7
3	i-off	i-off	f-disc	i-off	f-off	s-up					1.83
	cloud completion time										1.75

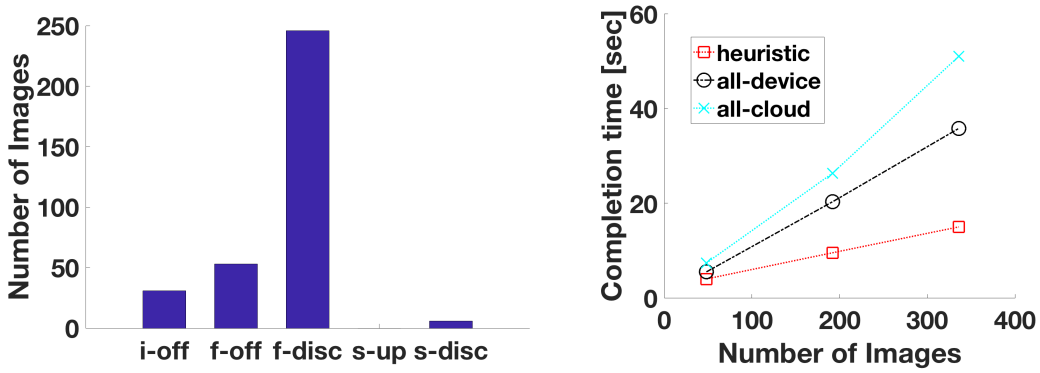
To get more insights, we look at the decisions made by our heuristic and see why it deviates from the optimal. Table 3.3 and Table 3.4 show the decisions each device makes in both the optimal and heuristic. We note that the optimal can be reached using various decision sequences; we only show one of them.

The decisions in the table correspond to *stage-decision*, where i refers to the Initial stage, f is the Filtering stage, and s is the Selection stage. The possible decisions are to offload -*off*-, upload -*up*-, or discard -*disc*-. This notation will be used throughout the remaining of the chapter.

For the heuristic, as the cloud is initially empty, and there are available tokens, all devices start by offloading. When the devices reach the third image in their local queues, the local transmission queue starts to grow, pushing devices to start processing locally. Taking a closer look at Dev-3, which is the slowest device in the system, the main cause of the delay is the choice made for image5. When the initial decision is made, the transmission cost is greater than the accelerated processing time, and the device chooses to start local processing. However, given that image5 is a hit, it will not be discarded after the Filtering stage. After that, the transmission queueing delay gets lower and image5 is offloaded to the cloud for further processing. In this scenario, image5 still incurs the transmission delay, causing an unnecessary delay in the process initiation of image6.

3.6.2.2 Testbed Implementation

We implemented PicSys on a testbed of three Samsung Galaxy S9 mobile devices and a server with NVIDIA GTX TITAN X. The devices are connected to the cloud through



(a) stage distribution: 336 images

(b) varying $|I|$

Figure 3.7. Implementation results.

Wifi. All devices and the cloud are using Caffe models. The expensive CNN is GoogLeNet, and for the accelerated CNN we use SqueezeNet. The Caffe models are pre-installed on the devices and their size is 25 MB and 3 MB, respectively. We queried for all images containing cars. Every device has 112 images, with a total of $|I| = 336$ images in the system. The hit-rate is $h = 0.25$, and the false positive rate $FPR = 0.166$. The transmission rate is 8 Mbps. We run the all-device, all-cloud and the time-heuristic and observe that our proposed heuristic outperforms the other approaches. Namely, our heuristic is 3.4× faster than all-cloud and 2.3× faster than all-device. In the case of all-cloud all 336 images are initially offloaded to the server and processed there, giving an average system completion time of 51.015 sec. For all-device, all images are processed locally on the device and 42 images are uploaded to the server as positive images, with an average completion time of 35.7 sec. The time-heuristic has the smallest average completion time of 14.99 sec.

The number of images in each stage, for the time-heuristic, is shown in Fig. 3.7(a).

In the previous scenario the total number of images in the system was fixed (336). Next, we consider how the makespan of the system changes as a function of the total number of images in the system. The other parameters remain unchanged. Fig. 3.7(b) illustrates that dependency for the three possible approaches. By varying the number of images in the system we can see that, as expected, with the increase in the number of images the completion time increases as well. From Fig. 3.7(b) it is clear that our heuristic outperforms both all-cloud and all-device. All-cloud has the largest completion time. As the number of images increases, there is a significant increase in the savings provided by the time-heuristic.

It is important to note that the code used in the system implementation is similar to

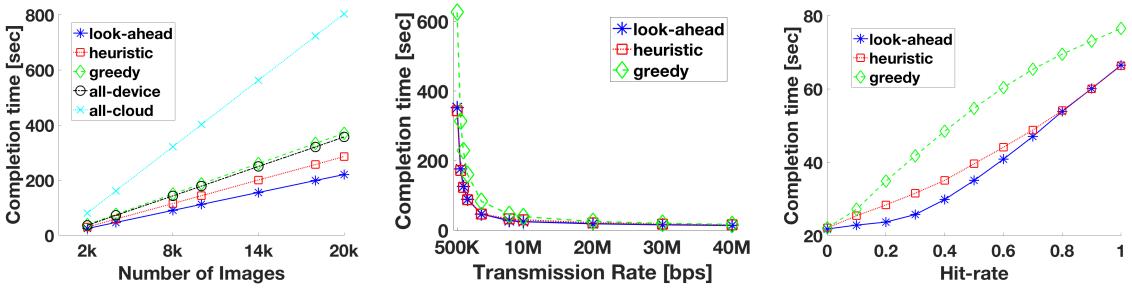
the code used for the experimentation results shown in Section 3.4.

3.6.2.3 Larger systems

In this section we use the measurements from the testbed to simulate a large-scale system in a controlled environment. We first consider the time-heuristic. We generate images with various sizes following a normal distribution $N(\mu, \sigma)$. Based on the results of Section 3.4.1, we use the following equation to calculate the scaling time of the images: $c_i^s = 0.033 * B_i + 0.031$. The transmission rates between the mobile devices and the cloud are set using a uniform distribution in the range $[\gamma r_m, r_m]$, where $\gamma < 1$. The processing rates using accelerated processing and expensive processing are set uniformly and randomly in the range $[\gamma\alpha, \alpha]$ and $[\gamma\beta, \beta]$, respectively. The processing rate on the cloud is also subject to a uniform distribution in the range $[\gamma c_u^p, c_u^p]$.

To match our implementation in Section 3.6.2.2, we choose Squeezenet as the accelerated CNN, and GoogLeNet as the expensive CNN.

We consider one cloud. Unless stated otherwise, the system parameters for the simulation are: $|M| = 20$, $h = 0.25$, $\mu = 1$ MB, $\sigma = 800$ kB, $r_m = 10$ Mbps, $\alpha = 0.2$ sec/img, $\beta = 0.4$ sec/img, $c_u^p = 0.01$ sec/img, $\gamma = 0.6$. For a choice of $k_1 = 5$, $k_2 = 3$, the corresponding true positive rates are $TPR_\alpha = 0.81$, $TPR_\beta = 0.82$, whereas the false positive rates are $FPR_\alpha = 0.0035$ and $FPR_\beta = 0.0017$, respectively. It is worth mentioning that the processing times, FPR and TPR, are based on experimental results. The results of the simulation are obtained by averaging over 100 independent runs. In the following, we will get more insights by varying different system parameters.



(a) Varying $|I|$ (b) Varying r_m (c) Varying h

Figure 3.8. Simulation results: time-heuristic.

Fig. 3.8(a) shows the comparison among all the algorithms. We can see that as the number of images in the system increases, the completion time increases, too. However, all-cloud has a higher growth rate caused by the larger queueing delays at the cloud. The

heuristic has the slowest growth rate and is closest to the look-ahead algorithm.

All-cloud and all-device never do better than the heuristic. When the optimal solution is to perform all-cloud or all-device, the heuristic does so. For better visualization, we exclude these two algorithms from the remainder of the figures.

Interestingly, even at very high transmission rates, the heuristic still results in some images being locally processed on the device. Fig. 3.8(b) shows that even at higher transmission rates, the processing rate of the cloud is not high enough to eliminate the cloud from being a bottleneck. On the other hand, at very low transmission rates, the best solution is for the majority of the images stored on mobile devices to be locally processed, and that is what our heuristic does. Although the heuristic and look-ahead curves look the same in the low rate region, they are not. E.g., at 8 Mbps the delay for the former is 32.7 sec, whereas for the latter it is 27.5 sec.

When varying the hit-rate in the system, all-cloud remains constant as all images are initially offloaded regardless of their truth values. All-device increases linearly since there is initially a constant cost of accelerated local processing that takes place regardless of the hit-rate. As the hit-rate increases, the number of images that go to expensive processing and are uploaded to the cloud increases. Since the heuristic integrates the hit-rate into its decision making process, it reaches look-ahead in both extreme cases of a hit-rate of 0 and 1, as illustrated in Fig. 3.8(c), whereas other algorithms fail to do so.

In general, we see that our heuristic outperforms the baselines and reaches the *look-ahead* in different cases. The number of devices used for the evaluation in this section was chosen to depict clearly the performance differences. Nevertheless, we have also run simulations with larger number of devices with similar conclusions drawn. E.g., in a system with 200 devices PicSys performs better than the other algorithms, and is only 3% worse than the look-ahead algorithm. In general, the system is aimed for a large number of users.

Finally, to demonstrate a case of the usefulness of tokens, we use the same setting as before in this section. For 100 devices, each of which contains 4 images, the cloud calculates that there should be around 75 tokens (according to Eq.(4.3)). The devices with the highest bandwidth receive these tokens (since they can utilize them faster than the other devices). As the tokens limit the initial offload, we obtain a completion time of approximately 4 sec. In the absence of tokens, devices initially see an empty cloud and offload immediately, increasing the completion time on the cloud. In that case, the system completion time is about 6 sec, which shows that in this case using tokens reduces the system completion time by 33%.

3.6.2.4 Heterogeneous systems

Next, to show that our algorithm is resilient to heterogeneity of devices and variations in parameters, we ran a simulation similar to that of Section 3.6.2.3 using 20 heterogeneous devices. These devices are grouped into four groups of five devices with the same characteristics. Different groups differ in processing rate and network bandwidth. Table 4 illustrates the results of our simulation. To highlight the differences, we assume a network with new devices and old devices, that are half as fast, as the clock speed for phones has almost doubled in the past years [78], [79]. We assume devices are in different locations, hence a variation in bandwidth. For illustration purposes, we use a factor of two i.e., $\alpha_{low} = 2\alpha$, $\beta_{low} = 2\beta$, $r_{low} = \frac{r_m}{2}$, and $\alpha_{high} = \alpha$, $\beta_{high} = \beta$, $r_{high} = r_m$.

Table 3.5 shows the average number of images processed at different stages for each device in the four different classes. Devices with higher bandwidth tend to offload more than other devices. Devices with high processing capabilities process more images locally than devices with lower processing rates and similar bandwidth. The devices with low bandwidth and low processing rates become the bottleneck, even as the heuristic reduces their completion times, compared to other algorithms. The completion time of the cloud in the system is 54.57, 64.51 and 160.17 sec for the heuristic, all-device and all-cloud, respectively.

Table 3.5. Heterogeneous system

bandwidth	processing	images per stage				device completion time [sec]		
r	α, β	i-off	f-off	f-disc	s	heuristic	all-device	all-cloud
high	high	19	5	66	10	27.748	35.424	80.07
high	low	31	12	56	1	35.8	63.16	80.07
low	high	4	1	76	19	37.83	42.964	160.07
low	low	15	4	68	13	51.802	63.542	160.07

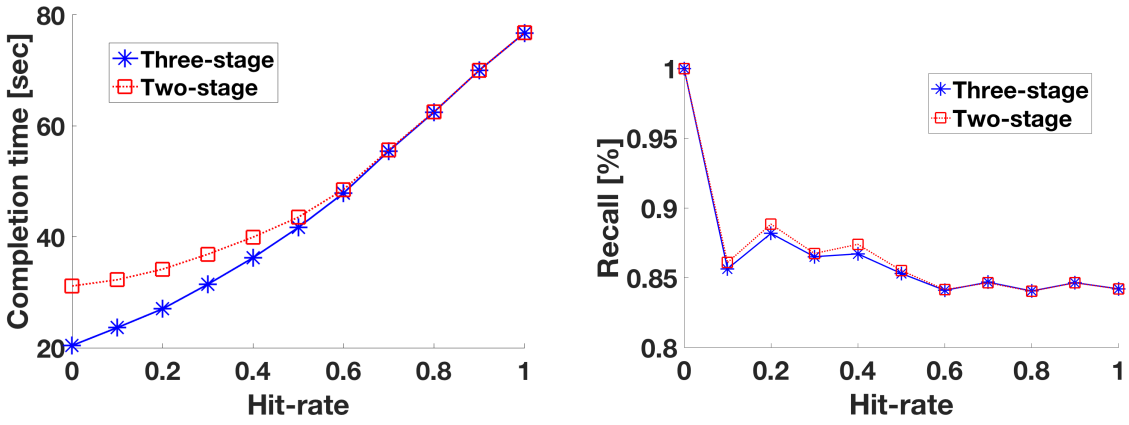
3.6.2.5 Comparison to no-Filtering stage

One of our main contributions is the addition of an intermediate, accelerated-processing, stage that acts as a filtering stage. In this set of simulations, we study the difference in completion times between our three-stage model and a "normal" two stage model, where there is no Filtering stage. In the second case, a mobile device initially makes the decision based on the network conditions, cloud backlog and hit-rate, and that decision is to either offload the image or process it locally. An image, if locally processed, has to go through all the layers of the CNN, since there is no accelerated CNN. To study the

benefits of adding a Filtering stage, we run the same set of simulations by varying the system parameters. Our model outperforms the standard two-stage model in all cases.

Fig. 3.9 shows that when using three-stages the completion time reduces by 35%; that decrease comes at a cost of less than 1% in recall. The gap between the two models increases as the hit-rate decreases. At low hit-rates, more images are processed locally. Many images are discarded in the Filtering stage and do not go through the expensive CNN, which results in time savings, as well as leads to the small decrease in recall.

Similarly, when the number of images in the system increases, given a fixed hit-rate, the number of negative images is going to increase. As a result, those images can be discarded after the Filtering Stage, leading to an increase in savings when using our three-stage model. From these results, *we can clearly see the advantage of adding an intermediate decision stage*, and having the option to offload later if the network or cloud conditions change.



(a) Time (b) Recall

Figure 3.9. Three vs. two stage.

The system in [25] has goals similar to ours but is geared towards video processing. It proposes an adaptive algorithm that attempts to reduce the completion time of processing videos across devices by assigning some videos to be processed in a cloud, and others locally on the devices. We adapted this system to process images instead of videos to provide a fair comparison. The key idea in the algorithm in [25] is to determine which devices have the longest expected completion time, and assign these to the cloud to reduce the overall system completion time. Unlike PicSys, the system in [25] does not consider the hit rate, or the fact that images that have the object of interest must be uploaded to the cloud even if they are processed locally on the mobile device From Fig. 3.10, it can be seen that PicSys does 44% better than [25]. This reduction in completion

time is due to accounting for the hit-rate and the introduction of a higher speed filtering stage. We also show that even in the absence of a filtering stage the two-stage algorithm of PicSys still performs better than [25] by about 15%.

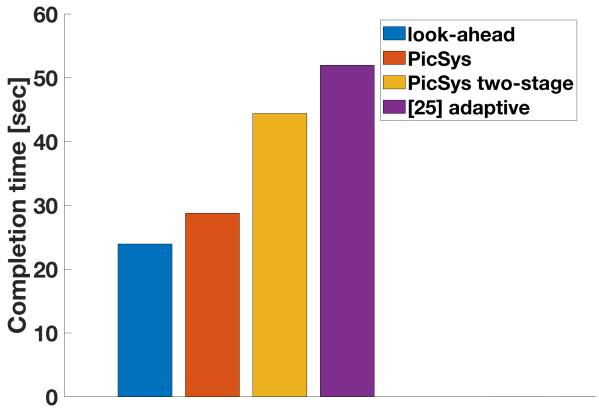


Figure 3.10. Comparison of PicSys to [25].

3.7 Related Work

Systems and techniques for computation offload have been long researched due to their promise in minimizing either mobile power consumption, or total computation time, or both. In that direction, Hermes [80] formulates an NP-hard problem to minimize the application latency while meeting prescribed resource utilization constraints and proposes a novel fully polynomial-time approximation scheme. In [66], authors optimize computation offload via distributing workload among multiple servers in a cloud. Glimpse [81] is a real-time object recognition system; it does local tracking and video labeling, and only sends frames that are likely to have new information to the server for object recognition.

Some video and image distributed crowd-sourcing systems have been proposed. Vigil [63] builds a video surveillance system that analyzes images from a number of cameras, leverages edge computing and only uploads a fraction of non-redundant videos, thereby optimizing bandwidth consumption. SmartEye [82] does image search and mines images to find image similarities and only uploads unique images via in-network deduplication. Neurosurgeon [83] dynamically partitions DNNs to move part of the processing to the edge, aiming to achieve low latency or energy consumption. The minimization is done for a single device only. A similar setting as ours is considered

in [25] and [10], but the choices of processing is restricted to either local processing or offloading.

In [84] and [85], the block mining process in exchange for monetary rewards is considered, where different incentive mechanisms that preserve privacy in crowdsensing applications are proposed. We assume voluntary participation in our work, but we can capture the case with incentives as well. This is beyond the scope of our work.

Early classification was studied in previous work such as Branchynet [86], where the authors propose using dynamic graphs with the addition of side branches to a linear CNN to allow early exit, based on a confidence metric to reduce inference cost. In [87], the system named Blockdrop is proposed to speed up ResNet by dynamically choosing which blocks to execute, while maintaining similar accuracy to the original ResNet CNN.

It is not straightforward to compare these systems in a unified network setup. Our system is not built for a specific CNN to be used, and as more research is developed, new techniques such as [23], [86], [87] and [88] to save on both energy and time are suitable for use. In our work we chose to use GoogLeNet [12] and partition it, as an example of early exit and Squeezenet [23] as an example of a compressed model. Similar to BranchyNet [86], we utilize the early learned features to get a prediction from early stages. As GoogLenet is a well-developed CNN, by using the caffe [36] framework it is easy for us to control our experiments and deploy the models on different platforms. We also show the effects of choosing the top-k values at each stage, and how that choice affects both accuracy and latency of the system.

Another work similar to ours is [89]. It uses code offloading, pipelining, and data parallelism to optimize for the makespan (execution time for a frame) and throughput (the rate at which frames are processed). This system operates in the following way: It divides tasks into the compute stage or network stage. In the former, the system calculates the estimated cost of offloading the stage or the estimated cost of spawning more works. In the latter, the estimated cost of offloading the source stage and the estimated cost of offloading the destination stage are calculated. The evaluation is performed on a netbook, laptop and a server, and the evaluation algorithms used are *all-offload*, *domain specific* (where the application specifications and developer's input are used to identify compute-intensive stages that can be offloaded to the server), and an *offline optimizer* (this is an oracle similar to our look-ahead algorithm). The objective in [89] is to minimize the completion time of a single device with no energy consideration. As opposed to [89], which does not consider energy consumption, PicSys is energy-aware, and minimizes the completion time of the entire system.

CloneCloud [90], built on [91], uses application VM clones at the server and partitions applications automatically. The objective is to minimize the completion time of the device. The main difference with PicSys is that this system focuses only on minimizing the completion time of only one device, whereas PicSys considers the entire system.

The system in [92], named ThinkAir, is used for similar applications. The main objective is to improve the performance through cloud computing, and dynamic scaling of computational power. The experiments were run based on an execution time policy with a boundary input value, which is the minimum value for input parameters for which offloading would be beneficial. The scenarios used are local processing on the phone, and sending to the cloud through Wifi and 3G. The evaluation is performed on a single device and on a single cloud. Again, as in the other related works, this system is run on a single mobile phone and optimizes the completion time of that single phone, as opposed to PicSys in which even large systems (with hundreds of phones) can be used and the overall performance optimized.

While PicSys uses offloading for crowdsourcing applications, as several of the other references, there are some key differences. Most importantly, PicSys considers the entire system, not a single device. It also collects all positively identified data, as opposed to simply labeling or processing data. Similarly, in cases where devices have high bandwidth, systems optimizing for a single mobile device result in *all-offload*; we have shown that such choice is not always optimal, given the backlog created on the server.

3.8 Conclusion

This chapter presents the design of PicSys, a system for efficiently searching for images over a set of mobile cameras. Since the problem is NP-hard, we propose low-complexity distributed online heuristics to optimize for the makespan. We show using experiments and simulations that the performance of the heuristics approaches that of a look-ahead oracle and performs better than other standard algorithms.

Chapter 4
Energy-Efficient Fast Image Search on Distributed Mobile Networks

In this chapter we extend the concepts of PicSys, presented in Chapter 3, where the system answers visual search queries on a mobile network. We solve the problem of minimizing the maximum completion time of the system over all devices, while taking into account the energy consumption of mobile devices.

4.1 Introduction

From an energy perspective, devices with high hit rates should offload their images because it saves the energy of performing the object classification processing on the device when images will eventually be uploaded anyway. For devices with low hit rates, the decision is more complex. If the accelerated CNN is less expensive from an energy standpoint than offloading the image over the wireless interface, local processing will be preferable. Whether this may be the case or not depends on the wireless technology and the quality of the wireless signal.

We take an in-depth look at the case where mobile devices are not plugged into power resources, and devices allocate a portion of their battery to perform the image search. The main contributions of our work are:

- We build on the PicSys system, presented in Chapter 3, which has the structure, via its multi-stage CNN pipeline, to support the design of algorithms that minimize query completion times with energy constraints.
- We perform experimentation to determine energy consumption for various CNNs. These measurements along with the processing times shown previously aid in the

selection of choices to meet our objective.

- We formulate and solve an optimization problem to decide where to run each of the stages of the computation, so that we utilize the available computational resources properly. In this chapter we show the case with energy constraints on mobile devices.
- We design a heuristic to solve the problem. The heuristic uses the previously designed time-heuristic as a building block for the energy-sensitive heuristic.
- We present simulation results for both LTE and WiFi.

4.2 PicSys Optimization

In this section we present the formal optimization problem for allocating resources in PicSys-energy, and introduce a heuristic algorithm. Ultimately, the goal is to process all images and have those with objects of interest uploaded to the cloud as fast as possible while respecting the energy constraints imposed by users of devices. The energy constraints are the budget for how much energy each user is willing to spend on the image processing application.

4.2.1 Mathematical Formulation: energy constraints

In Section 3.5.1 we considered the problem of minimizing the completion time where the mobile devices are willing to help without any conditioning on energy consumption. However, there may be users who would be willing to help but would like to limit their battery consumption for that purpose. In this section we consider the problem where every user has a finite amount of energy they are willing to allocate for crowd-sourcing. To this end, we adapt the previous optimization problem, but adding the energy consumption constraint.

Let a_m denote the percentage of battery the mobile device m would be willing to spend on image detection. The percentage of energy consumed for image i on device m is denoted as $E_{i,m}$. The new optimization problem becomes:

Table 4.1. Nomenclature and Notation

$cost1_{i,m}$	cost of direct offload
$cost2_{i,m}$	cost of accelerated processing followed by offload
$cost3_{i,m}$	cost of accelerated processing followed by discard
$cost4_{i,m}$	cost of accelerated processing, then expensive processing followed by upload
$cost5_{i,m}$	cost of accelerated processing, then expensive processing followed by a discard
a_m	% of battery device m dedicates to crowdsourcing
$E_{i,m}$	energy consumed for image i on device m (in %)
E_m^r	residual energy in the budget of device m (in %)
$e_{i,m}^s$	scaling energy of image i on device m (in %)
$e_{i,m}^t$	transmission energy of image i on device m (in %)
e_{α_m}	processing energy for accelerated CNN of device m (in %)
e_{β_m}	processing energy for expensive CNN of device m (in %)

$$\begin{aligned}&\text{minimize } T_{max}\\&\text{subject to: } (3.1), (3.2), (3.3), (3.4), (3.5), (3.6), (3.7), (3.8),\\&\sum_{i \in I_m} E_{i,m} \leq a_m, \qquad \forall m \in M.\end{aligned} \tag{4.1}$$

In Eq.(4.1), for $E_{i,m}$ we have

$$\begin{aligned}E_{i,m} = (1 - x1_{i,m})cost1_{i,m}^{(e)} + x2_{i,m}cost2_{i,m}^{(e)} +\\ x3_{i,m}cost3_{i,m}^{(e)} + x4_{i,m}cost4_{i,m}^{(e)} + x5_{i,m}cost5_{i,m}^{(e)}.\end{aligned} \tag{4.2}$$

Note that Eq.(4.2) is the energy counterpart of Eq.(3.10), and because of that $x1_{i,m}, \ldots, x5_{i,m}$ have the same meaning as in the first optimization problem, whereas $cost1_{i,m}^{(e)}, \ldots, cost5_{i,m}^{(e)}$ represent the cost in terms of energy, corresponding to the same stages as (time) costs $cost1_{i,m}, \ldots, cost5_{i,m}$ in Eq.(3.10), and are given as:

$$cost1_{i,m}^{(e)} = e_{i,m}^t,$$

$$cost2_{i,m}^{(e)} = e_{i,m}^s + e_{\alpha_m} + e_{i,m}^t,$$

$$cost3_{i,m}^{(e)} = e_{i,m}^s + e_{\alpha_m},$$

$$cost4_{i,m}^{(e)} = e_{i,m}^{s} + e_{\alpha_m} + e_{\beta_m} + e_{i,m}^{t},$$

$$cost5_{i,m}^{(e)} = e_{i,m}^{s} + e_{\alpha_m} + e_{\beta_m},$$

where $e_{i,m}^{s}$ is the energy cost (in %) of scaling image i on device m, e_{α_m} is the energy cost (in %) of the accelerated CNN for device m, e_{β_m} is the energy cost (in %) of the expensive CNN for device m, whereas $e_{i,m}^{t}$ (again in %) is the cost of uploading image i from device m.

Since the optimization problem with the energy constraints is essentially the extended version of the first optimization problem, and we have proved that the latter is NP-hard, adding a new constraint to it does not change the NP-hardness. Hence, the new optimization problem is also NP-hard. As a result, similarly to the first case, we will resort to using heuristics for this case as well.

4.2.2 Proposed Heuristic: energy constraints

The overall goal of the energy-heuristic is the same as the time-heuristic, but the energy-heuristic tries to ensure that the largest amount of images on a device will be processed before the battery budget is exhausted. At specific intervals the device probes for remaining energy budget. Based on the outcome, the device determines if it needs to choose low-energy processing options, or can be more aggressive and choose processing options that are of higher rate.

At each battery probe the heuristic works as follows:

- The mobile device checks to see if it has enough remaining energy in its budget to process all unprocessed images locally; this is the worst case energy processing scenario. If so, the device can run the time-heuristic without concern for energy because it has sufficient energy to complete the task regardless of the processing decisions.

- If there is not enough energy to process all images locally, in lines 2−8 of Algorithm 4, the device checks to determine if there is enough energy to offload all unprocessed images. If not, the device may not be able to process all images before exhausting its budget. In this case it checks to see the energy required to offload all remaining images, or execute the accelerated CNN only on all images, and chooses the cheapest option which will likely result in the most images being processed. This will depend on the type of wireless interface being used.

- If the device does have enough energy to offload all the images, the algorithm is more aggressive. The device first determines how many images can be fully processed locally using its remaining energy, which is the worst case energy scenario, and if it can process at least one, it sets the probe window to this size and runs the time-heuristic. If it does not have enough energy to process a single image locally, it determines the number of images it can process using only the accelerated CNN. If it can process at least one, it sets this value to the next probe window and runs the time-heuristic. If there is not enough energy to perform the accelerated CNN on a single image, then all images are offloaded.

Algorithm 4: ENERGY_ PROBE()

```
1  if E_m^r < num_imgs * (e_m^s + e_{α_m} + e_{β_m}) then
2      if E_m^r < num_imgs * e_m^t then
3          if num_imgs*e_m^t < num_imgs*(e_m^s + e_{α_m}) + num_hits*e_m^t then
4              ALL OFFLOAD;
5          else
6              ALL ACCELERATED ;
7          end
8          next_probe=window_size;
9      else
10         if E_m^r / (e_m^s + e_{α_m} + e_{β_m}) > 1 then
11             next_probe=max(E_m^r / (e_m^s + e_{α_m} + e_{β_m}),window_size);
12             TIME_HEURISTIC();
13         else
14             if E_m^r / (e_m^s + e_{α_m}) > 1 then
15                 next_probe=max(E_m^r / (e_m^s + e_{α_m} + e_{β_m}),window_size);
16                 restrict local processing to accelerated only;
17                 TIME_HEURISTIC()
18             else
19                 ALL OFFLOAD;
20             end
21         end
22     end
23 else
24     TIME_HEURISTIC();
25 end
```

Algorithm 4 shows the logic at each energy probe. Similar to Section 3.5.2, we drop the index i for the costs, as it represents the *average* values. In the remainder of the

chapter we call this the **energy-heuristic.**[1]

4.2.3 Design Considerations

We consider next the design related issue of energy probing.

4.2.3.1 Energy Probing

In our heuristic the algorithm probes the device for the remaining energy. Since battery percentages are represented in integer values, we choose our probing window accordingly. We estimate the amount of images a device can locally process using 1% energy and set that to be the window size:

$$window\_size = \frac{1}{e^s_m + e_{\alpha_m} + e_{\beta_m} + e^t_m}. \tag{4.3}$$

4.2.4 Complexity

The energy-aware algorithm has the same complexity as the time-heuristic shown in Section 3.5.4, i.e., $O(n)$, as it follows the time-heuristic with additional energy comparisons and restrictions, all of which induce constant number of operations in each step.

4.3 Experimental Evaluation

As mentioned previously, three factors need to be considered when choosing the deep learning models: recall, processing time and energy consumption. The requirement for the accelerated CNN is to have a faster processing time with lower energy expenditure than the expensive CNN.

Following the experiments shown in Section 3.4, we ran another set of comparisons where we measure the energy cost for processing images using the three CNNs: Squeezenet, GoogLeNet and the accelerated CNN at Classifier-2. The measurements were taken by profiling battery usage using Battery Historian [93]. We represent the energy as the percentage of battery consumed per image. We find that Classifier-2 CNN and Squeezenet have almost the same energy cost, which is 64% lower than that of the complete GoogLeNet.

[1]The energy-heuristic tends to follow the decisions of the time-heuristic whenever there is enough residual enrgy for crowdsourcing in the smartphone.

4.3.1 Simulation Results - Energy

In this set of simulations we analyze our energy-heuristic algorithm and highlight the important results. We consider user thresholds of 1%, 2% and 3% of total energy to be used for this application. Unless stated otherwise, the distributions and parameters for the simulations are similar to Section 3.6.2.3. The new parameters are: $e^s = 0.00016\%$/img, $e^t = 0.0006\%$/img for a 12 Mbps Wifi connection, $e^t = 0.0032\%$/img for LTE with the same throughput, $e_\alpha = 0.0037\%$/img and $e_\beta = 0.0103\%$/img. There are ten devices ($|M| = 10$), and $|I| = 10,000$.

Fig. 4.1 illustrates the completion times for the energy-heuristic in three different cases: Wifi network with devices using 1% of the total battery for crowd-sourcing each, Wifi network with devices using 3% battery for crowd-sourcing, and an LTE network with devices using 3% battery for the same purpose. From Fig. 4.1 it can be observed that in the Wifi scenario of 3% for crowd-sourcing the energy is sufficient and the devices follow the time-heuristic.

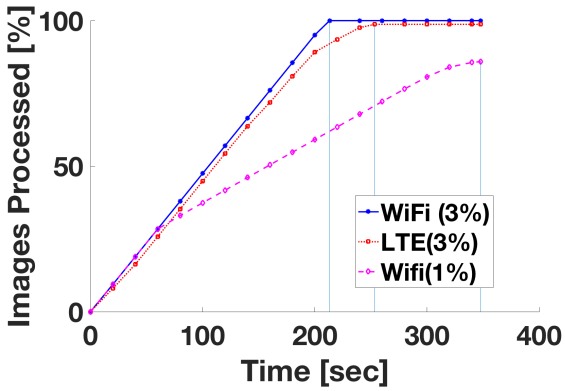


Figure 4.1. Energy-heuristic: varying a_m.

Having considered the effect of the maximum energy for crowd-sourcing, we proceed with looking at the distribution of stages at which the decisions are being made. In that direction, Fig. 4.2 depicts the number of images that go through each of the system's decision stages. The stages in the x-axis are as follows: 1:Initial-offload, 2:Filtering-discard, 3:Filtering-offload, 4:Selection-discard, 5:Selection-upload, 6:Image-not-processed.

As expected, the time-heuristic with no power budget has the shortest completion time. The higher the available energy, the closer the energy-heuristic is to the time-heuristic. When using Wifi, the stricter the energy constraint, the more initial-offloading, as shown in Fig. 4.2. With the strict energy constraint (1%), the system completes 86% of the images. When using the time-heuristic with the 1% energy constraint, only 51% of

the images are processed, because the time-heuristic does not consider the energy budget. This illustrates the effectiveness of the energy-heuristic. We have also observed that in the 2%-case (not shown in Fig. 4.2), the device is able to process all the images, but is 1.3× slower than the time-heuristic.

On the other hand, when using LTE, the cost of offloading becomes expensive and a 3% energy constraint is not sufficient to follow the time-heuristic, as in the case of Wifi. In Fig. 4.2 we see that at a restrictive energy constraint, LTE does more local processing than initial-offload, as opposed to Wifi.

To summarize, in energy-constrained cases, there is a tradeoff between completion time and the amount of images processed.

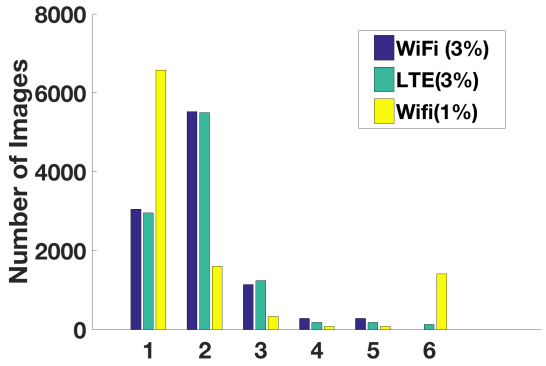


Figure 4.2. Decision stage distribution.

4.3.2 Related Work

Techniques such as pruning are also proposed to reduce network sizes and enabling them to run faster and lighter on mobile devices. In [88], there is a layer-by-layer pruning, which reduces the network size, using energy estimation. Energy estimation is performed based on data movement in the memory and the number of multiply and accumulate (MAC) operations, as well as data sparsity.

More related works to ours are MAUI [91], Odessa [89], CloneCloud [90], ThinkAir [92], which perform fine-grained partitioning and code offload to minimize energy consumption and total execution time. Their primary focus is on enabling code-offload with minimal changes to applications.

The principal goal in [91] is to minimize energy consumption of mobile applications subject to latency constraints. The authors have shown that when there is a good network connection, there is an improvement in execution time. However, the system incurs a

large overhead in 3G. The main feature of the algorithm is that the programmer marks the portions of the code that can be run remotely. The server has a MAUI coordinator, the mobile device and the server both have a proxy, profiler and solver. The profiler calculates the runtime and energy cost for each method, as well as the other network parameters. The main limitation of [91] is that it aims to optimize for a single mobile device only. As opposed to [91], PicSys is run on a large number of devices where both Wifi and 4G interfaces can be used to send the images to the server. The other difference is in the objective function; the objective in [91] is to minimize energy consumption, while in PicSys we minimize the system completion time, given energy constraints.

PicSys optimizes for latency subject to energy constraints. If we were to optimize for a single device in a setting where the offload incurs a large cost, the results would lean towards processing all or most data locally. However, since in PicSys the positively classified data have to be *uploaded* to the cloud, PicSys finds better solutions, which are not found by other systems mentioned here.

4.3.3 Conclusion

In this chapter we build on the problem proposed in Chapter 3, which proposes a system that efficiently searches for images in a network that consist of multiple mobile devices. We improve the design of PicSys to be energy-aware. We show that the problem is NP-hard and propose an online heuristic to optimize for makespan given an energy constraint. We show experiments and simulations for both WiFi and LTE, and compare the heuristic to the time-heuristic shown in Chapter 3.

Chapter 5
EDIR: Efficient Distributed Image Retrieval of Novel Objects in Mobile Networks

Crowdsourcing data collection from a network of mobile devices is useful in various applications. Mobile devices store a large amount of visual data that can aid in different application scenarios. Trained Convolutional Neural Networks (CNNs) can be deployed on mobile devices to be used in searching for objects of interest. Querying for novel objects, for which models have not been trained yet, presents some unique challenges. When novel objects are queried, new models must be trained and distributed to all edge devices. In this chapter, we propose an efficient method and a system, called EDIR, which enables answering these queries while taking into account the bandwidth limitations encountered in wireless networks, as well as the limited energy and computational power on mobile devices. Through extensive experimentation, we show that using distance-based classifiers, specifically those relying on the Cosine distance, our approach leads to more efficient utilization of network resources by reducing the number of false positives. We perform analysis that enables the requester to tune the parameters of interest before issuing the query, and validate our theoretical results. EDIR reduces the amount of transferred data by more than 45% compared to other approaches while simultaneously achieving a good F1 score [94].

5.1 Introduction

Mobile devices, such as surveillance cameras, mobile phones and drones, are used for collecting data in many crowdsourcing applications [11], [69], with information media

being audio, images, or video. Very often, images are the source of information, whereas a key application is finding specific objects within these images. To find these objects of interests, mobile devices must have previously installed pre-trained models. Pre-trained models might be initially deployed on devices, but their use is limited as they are only trained on specific objects.

In cases when a new, novel object, is being searched for, the mobile device will need to install a new model, in our case a Convolutional Neural Network (CNN) that has been trained on the novel object. Mobile devices have constrained battery life and very often encounter limited network connectivity, making it hard to *repeatedly* download new models which, in general, are quite large. On the other hand, uploading all images from mobile devices to servers where they can be processed is not desirable since that would create a large backlog (there can be a large number of devices participating in crowdsourcing) on the server side [68], congesting the network and degrading the overall system performance.

Let us consider the case in which mobile cameras, in bandwidth- and energy- constrained networks, are used for surveillance or gathering intelligence in a time-sensitive manner. Such networks may be *tactical networks* [95] or cellular phone users using LTE or 5G who want to limit data download in order not to exceed their flat rate plan or who are not willing to deplete their battery on this type of tasks. An alert for a specific object that has just been spotted and not trained on before is issued. Training new models to search for new objects is time consuming and requires a large number of training samples, which may not be available. Even if training is possible, using transfer learning [96], and methods such as few-shot learning [97], the training time is long, and the bandwidth required to download the newly trained models on all edge devices in constrained environments is going to be excessive. As opposed to those methods, we introduce a more practical alternative method that aims to transfer a small amount of data, in a *timely manner*, when issuing and responding to such queries.

In this chapter, we address the challenge of retrieving images that contain specific objects from edge devices, for which models deployed on edge devices have not yet been trained. We attempt to enable these devices to utilize the available pre-loaded CNNs to respond to such queries without retraining at the edge. Our approach is to identify which pre-trained *base classes* are similar to the untrained (queried) *novel class*, and to find the threshold between the two classes that allows the novel class and base class to be distinguished from each other. We use a small number of samples of the novel class to find similar base classes.

The first challenge is to determine the similar base classes which we will use as the first filter in the process. The second challenge is to determine a proper distinction between the similar base class and the novel queried class that will enable us to distinguish between the two classes. The third challenge is to reduce the amount of transmitted data in the wireless network, which will reduce the energy consumption of mobile devices.

Specifically, our main contributions in this work are:

- We propose a flexible and tunable distance-based algorithm for solving the novelty object image retrieval problem using a small number of samples of the novel object and efficiently utilizing the network bandwidth and device's energy.
- Through system analysis, we enable the server to set the values of the input parameters in order to meet the user's objective and constraints before issuing a query.
- We implement the system, present a detailed performance evaluation and show the improvements it provides compared to other approaches.

The aforementioned contributions are achieved with the help of the system we propose in this chapter and implement in practice, which is called EDIR. When a query is issued, EDIR relies on the limited available information about the novel class and sets the system parameters to accommodate the issuer's request and the network constraints. EDIR is able to send a query with as little as 13 kB and achieves an F1 score of 0.7 for the novel class. EDIR also introduces the flexibility of tuning the system parameters, which makes it a better choice compared to other novel object classifiers.

The remainder of the chapter is organized as follows. Section 5.2 presents a brief overview of image processing and related definitions. Section 5.7 reviews some related works. An overview of the system is provided in Section 5.3. Section 5.4 presents the analytical model and details of the process for novel image retrieval related to both the server and edge devices. Some design considerations are provided in Section 5.5. In Section 5.6 we implement EDIR and evaluate its performance along with a comparison to another state-of-the-art algorithm. Finally, Section 6.8 concludes the chapter.

5.2 Overview of Image Processing and Definitions

In this section we present a brief background on image processing and define important concepts that help in understanding the chapter.

Object classification is the process of taking an input image and outputting a class or probability that an image contains a specific object. The specific objects that are searched for in an image are predefined during training time. This set of pre-trained objects are called *base classes* (denoted by $\mathcal{B}$).

Various performance metrics are used to show the performance of an object classification algorithm. The three metrics that we will consider are: *recall*, *precision*, and *F1 score*, which are widely used in the image processing community.

Recall, also known as **true positive rate** (TPR) is the rate of correctly retrieved positive samples. It is calculated as $\frac{TP}{P}$, where TP is the number of true positive samples and P is the number of ground truth positive samples.

Precision is the measure of the relevant instances among the predicted positives, and is calculated as $\frac{TP}{TP+FP}$, where FP is the number of false positive samples.

The **F1 score** is defined as $F_1 = \frac{2*Precision*Recall}{Precision+Recall}$, i.e., it is the harmonic mean of precision and recall. This measure is used to "make a balance" between precision and recall, and is the preferred measure when we have uneven class distributions, which is exactly the case in our work.

As a classifier, we use a Convolutional Neural Network **(CNN)**, which consists of a set of layers and ends with a *Softmax* layer that outputs a probability distribution of each trained object being contained in the image. The higher the probability, the more likely that the object is in the image. Often a value of k is set as a threshold. If an object is within the top k probabilities, this object is considered to be in the image; we refer to this threshold as **top-k**.

Given a trained CNN, we call the predefined classes that appear during training the **base** or **seen** classes. We refer to the **queried** (**unseen**) class as the **novel** class that has not appeared in training. These terms are used interchangeably throughout the chapter. Each class has a mean activation vector **(MAV)**. It is calculated by the server, by averaging on the extracted features from the penultimate layer in the CNN. We also define a **closest base class**, which is the base class that is most similar (closest in distance) to the novel class. Defining and setting closest classes is presented in Section 5.4.1. We also define another threshold c, which relates to the c base classes that are closest to the novel class. We refer to these closest classes as **closest-c**.

Table 5.1 summarizes the notation used throughout this chapter.

Table 5.1. Definitions and Notation

P	Number of ground-truth positive images
N	Number of ground-truth negative images
TP	Number of true positive images
TN	Number of true negative images
FP	Number of false positive images
FN	Number of false negative images
$\mathcal{U}$	The set of all classes
$\mathcal{B}$	The set of base classes
$\mathcal{C}$	The set of closest-c classes to the novel class
$\mathcal{K}$	The set of top-k highest output base classes of the Softmax layer for an image
b	Base class
q	Queried class
$\mathcal{X}$	The set of all images in the system
$\mathcal{X}_q$	The set of images of the queried class
$f(x)$	The output of the penultimate layer of the CNN
θ_b	Threshold to distinguish an image belonging to $b \in \mathcal{B}$ or q
n_b	Number of training images belonging to the base class b
N_b	Number of training images belonging to base classes
n_q	Number of samples of the novel queried class
n	Total number of training images
MAV	Mean activation vector
$\mathcal{D}$	The set of distances
d	The distance to an MAV
μ_b	$mean(\mathcal{D})$
σ_b	standard deviation of $\mathcal{D}$
μ_q	The mean of the queried class
σ_q	The standard deviation of the queried class
α	The empiric tuning coefficient

5.3 System Overview

The system consists of a set of mobile devices participating in crowdsourcing and a central server which can be accessed by those devices.[1] The devices and server each have a stored copy of the same trained CNN, which is trained on the set of base classes that are expected to be searched for. These mobile devices have stored images taken over time and respond to queries seeking objects contained in the images, as shown in Fig. 5.1. Devices may be asked to search for an object that they have not been trained for in the images they have already stored.

[1]Different mobile users are not necessarily within the coverage area of the same base station.

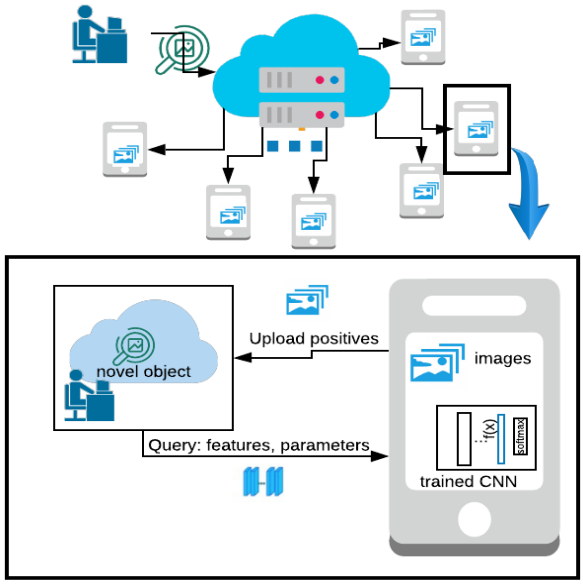


Figure 5.1. Overview of the system.

In EDIR, whose functioning is illustrated in Fig. 5.1, Fig. 5.2, and Fig. 5.3, the operation starts when the query issuer has access to one or more samples of a novel object that has recently been spotted. The query is sent to the server with the samples of the novel object and the requirements, i.e., if the bandwidth is constrained, or if the requester wants to focus on precision, recall or F1 score. The server finds the best settings based on these requirements. The settings include how many close classes are considered and whether or not k should be set to higher than 1. The server uses the trained CNN and the available samples, and calculates an MAV for the novel class. Then, it finds the closest c base classes to the novel class. The server sends the MAV of these base classes along with their class labels to the devices. It also sends the thresholds (denoted as θ) that are used to distinguish the base classes from the novel class, in addition to the value of k, as shown in Fig. 5.2. The devices receive the query and start by searching images for closest base classes. Within these images, the distances of the image to the thresholds are compared; images that pass both those tests are classified as containing the novel class and are sent to the server, as illustrated in Fig. 5.3.

EDIR is able to send as little as 13 kB of data from the server to edge devices when sending a query, as shown later in Section 5.6. More details on the server processing and parameter setup can be found in Section 5.4.

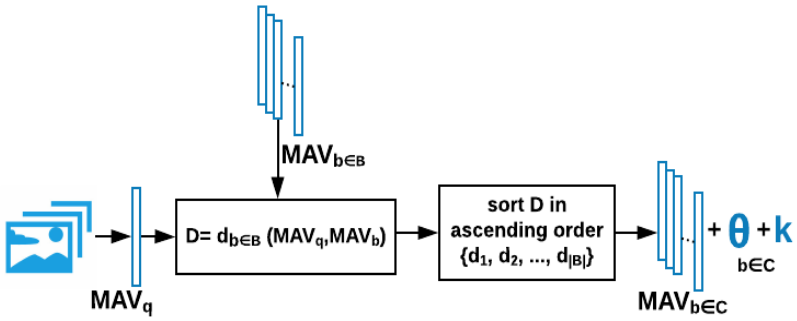


Figure 5.2. Server process overview.

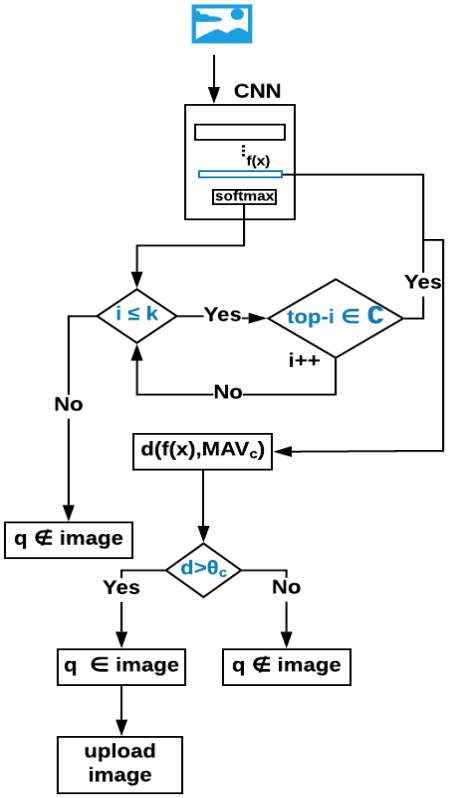


Figure 5.3. Edge device process overview.

5.4 EDIR Model

As shown in Fig. 5.1, the requester sends a request for images containing a novel class. The server calculates the parameters to help the mobile devices in searching their locally stored images for the novel class. Once an image is labeled as positive, i.e., containing the object of interest, the image is offloaded to the server. In this section, we present

our theoretical analysis, and illustrate the process on both the server and edge (mobile) device.

5.4.1 Server

The training images for the base classes are initially available on the server. They aid the server in calculating the MAV for each of the base classes and the corresponding values of threshold θ. When a query is issued, the server receives a small number of samples of the queried novel object. Subsequently, it calculates the closest base classes to the novel class, as well as the appropriate values of c and k, as shown in Fig. 5.2.

Let $x \in \mathcal{X}$ be an input image, and $b_i \in \mathcal{B} = \{b_1, b_2, ...b_m\}$ be an output base class label, where m is the number of base classes on which the CNN is pre-trained. Let q be the queried novel class. Then, $\mathcal{U}$ is the set of all possible classes $\mathcal{U} = \mathcal{B} \cup \{q\}$, and $\mathcal{X} = \mathcal{X}_{b \in \mathcal{B}} \bigcup \mathcal{X}_q$ is the set of all images. We denote the set of base classes that have the top-k maximum values after the Softmax layer as $\mathcal{K}$. We refer to the set of base classes that are the c closest from the novel class as $\mathcal{C}$.

The server sets three different parameters: k, c, and θ_b. As mentioned previously, the term top-k is used to refer to the k predictions that have the highest probabilities (after the Softmax layer). Closest-c are the top c base classes that are closer in distance (more similar) to the queried novel class. Finally, θ_b for every $b \in \mathcal{B}$ is the threshold used to distinguish an image x as belonging either to the base class b or to the novel class q. We will discuss the tuning of these parameters later in this section.

5.4.1.1 Calculating closest base classes

In order to calculate the closest base classes, a distance measure is used. We considered both the Euclidean and Cosine distances, and found the Cosine distance to perform better. Other distance measures, such as the Mahalanobis distance, have been proposed [26], against which we compare our results to in Section 5.6.3.

Given what was said above, EDIR uses Cosine distance to measure the distances between features extracted from the CNN. For vectors $\vec{u}$ and $\vec{v}$, the distance is expressed as

$$\text{dist}(\vec{u}, \vec{v}) = 1 - \text{sim}(\vec{u}, \vec{v}), \tag{5.1}$$

where $\text{sim}(\vec{u}, \vec{v})$ is the Cosine similarity between two vectors, and is defined as

$$\text{sim}(\vec{u}, \vec{v}) = \frac{\sum_{i=1}^{a}(u_i v_i)}{\sqrt{\sum_{i=1}^{a}(u_i)^2}\sqrt{\sum_{i=1}^{a}(v_i)^2}}, \tag{5.2}$$

with a being the dimension of the vector.

Every base class $b \in \mathcal{B}$ is represented using the mean μ_b and the standard deviation σ_b, based on the training samples $x \in \mathcal{X}_b$.

Let x be an image, $f(x)$ is the output of the penultimate layer of the CNN and n_b is the number of training images belonging to class b. The mean activation vector of class b, MAV_b, is calculated using images $x \in \mathcal{X}_b$ as

$$MAV_b = \frac{1}{n_b} \sum_{x \in X_b} f(x). \tag{5.3}$$

Let $\mathcal{D}$ be the distances between every feature vector $f(x)$ extracted from $x \in \mathcal{X}_b$, and the MAV_b, $\mathcal{D} = \text{dist}(f(x), MAV_b)$. The mean and standard deviation of $\mathcal{D}$ are denoted as $\mu_b = mean(\mathcal{D})$ and $\sigma_b = std(\mathcal{D})$, respectively.

When the query is issued, the server has access to a small number of samples of the novel class, $x \in \mathcal{X}_q$. It uses this small number of samples to calculate the MAV_q of the novel queried class, similarly to the MAV of the base classes, using the adapted version of Eq.(5.3).

The closest base classes are defined as the base classes with the smallest distance to the novel class, where the measure of distance is between the MAV's of each class. Let $b \in \mathcal{B}$ be a base class. The distances between the MAV of the novel class and the MAV of the base classes are calculated as $\text{dist}(MAV_q, MAV_b)$, using Eq.(5.1).

The advantage of our approach is that it can capture other distance measures, besides the Cosine distance, as needed. Nevertheless, performing experiments on a large set of possible distance measures, we observed that the best results are achieved when using the Cosine distance.

5.4.1.2 Choosing k and c

Based on the constraints the requester indicates, the server sets the values of c and k based on the analysis it performs using the available training samples of the base classes, along with the small number of samples of images available for the novel queried class. In the following, we present the details of this analysis for different scenarios regarding

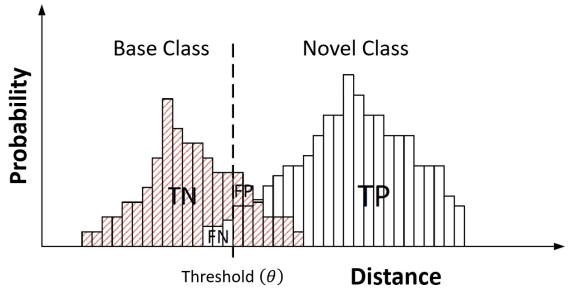


Figure 5.4. Classification histogram.

the values of k and c.

There are four types of outputs after classification: (i) true positives (TP): images of the novel class classified correctly; (ii) true negatives (TN): images discarded correctly as not belonging to the novel class; (iii) false positives (FP): images incorrectly classified as the novel class; (iv) false negatives (FN): images incorrectly discarded as not containing the novel class. These values are used to calculate various performance indicators, such as recall, precision and F1 score, as already mentioned in Section 5.2.

Let n_q denote the (usually small) number of samples of the novel queried class (positive samples), and let $N_b = \sum_{b \in \mathcal{B}} n_b$ denote the number of training samples for the base classes (negative samples). Then, $n = n_q + N_b$ is the total number of samples (the full population).

Further, let k be the number of maximum classes we use from the Softmax output. We denote the set of top-k outputs from the Softmax as $\mathcal{K}$, where $|\mathcal{K}| = k$. As mentioned, c is the number of closest base classes used for comparison in the second phase of the process. We denote the set of closest-c classes as $\mathcal{C}$, and $|\mathcal{C}| = c$. Note that $\mathcal{K} \subset \mathcal{B}$ and $\mathcal{C} \subset \mathcal{B}$.

For clarity, we start by showing the analysis for top-1 closest-1 in predicting the classification accuracy. This is followed by the cases of top-k closest-1, and top-1 closest-c. Finally, we show the general case of top-k closest-c.

top-1, closest-1. In this scenario, we denote the maximum output from Softmax as $k_1 = \max(\mathcal{K})$. The closest base class is denoted as c_1. The variable d represents the distances to the MAV of the closest base class, and $\mathbb{P}$ denotes the probability of an event.

In order for an image to be classified as true positive, its ground truth must be "positive" ($Label = \mathcal{P}$), the maximum class of the Softmax must be the closest base class ($\max(\mathcal{K}) = c_1$), and it must be to the right of the threshold, i.e., $d > \theta_{c_1}$. See Fig. 5.4 for the latter condition.

Result 1 *The expected number of true positives for a given closest base class b_i can be estimated as*

$$\mathbb{E}[TP|c_1 = b_i] = n\mathbb{P}(Label = \mathcal{P}) \cdot \mathbb{P}\left(k_1 = b_i|Label = \mathcal{P}\right) \cdot \mathbb{P}\left(d > \theta_{b_i}|k_1 = b_i, Label = \mathcal{P}\right). \tag{5.4}$$

Taking into account all the possible outcomes of the closest base classes, for the overall expected number of true positives we get

$$\mathbb{E}[TP] = \sum_{i=1}^{m} \mathbb{E}[TP|c_1 = b_i]\mathbb{P}(c_1 = b_i), \tag{5.5}$$

where $\mathbb{P}(c_1 = b_i)$ is the probability that the closest base class from the set of novel images is class b_i.

For an image to be classified as a true negative, its ground truth must be "negative" ($Label = \mathcal{N}$), and it either (i) must have been discarded in the first phase, i.e., the image's maximum class from Softmax is not the closest base class, or (ii) the closest base class is the highest value from the Softmax of all the $\mathcal{B}$ classes, and it was on the left of the threshold (see Fig. 5.4).

Result 2 *The expected number of true negatives for a given closest base class b_i can be estimated as*

$$\mathbb{E}[TN|c_1 = b_i] = n\mathbb{P}(Label = \mathcal{N}) \cdot (\mathbb{P}(k_1 \neq c_1|Label = \mathcal{N}) + \mathbb{P}(k_1 = c_1|Label = \mathcal{N}) \cdot \mathbb{P}(d \leq \theta_{b_i}|k_1 = c_1, Label = \mathcal{N})). \tag{5.6}$$

The expected number of true negatives over all possible outcomes of the closest base class is

$$\mathbb{E}[TN] = \sum_{i=1}^{m} \mathbb{E}[TN|c_1 = b_i]\mathbb{P}(c_1 = b_i). \tag{5.7}$$

top-k, closest-1. In this scenario, we extend the range of ranked Softmax values to k when looking for the closest class. This is the only change in the analysis compared to the case of top-1 closest-1. If $\mathcal{K}$ denotes the top-k Softmax values for a given image, then $c_1 \in \mathcal{K}$ indicates that the closest class is in top-k. Taking this into account, we have:

Result 3 *The expected number of true positives, given that the closest base class is b_i, is*

$$\mathbb{E}[TP|c_1 = b_i] = n\mathbb{P}(Label = \mathcal{P}) \cdot \mathbb{P}(c_1 \in \mathcal{K}|Label = \mathcal{P}) \cdot \mathbb{P}(d > \theta_{b_i}|c_1 \in \mathcal{K}, Label = \mathcal{P}). \quad (5.8)$$

Replacing Eq.(5.8) into Eq.(5.5), we obtain the "overall" expected number of true positives, $\mathbb{E}[TP]$.

Similarly, for the true negatives we have the following:

Result 4 *The expected number of true negatives when the closest base class is b_i is*

$$\mathbb{E}[TN|c_1 = b_i] = n\mathbb{P}(Label = \mathcal{N}) \cdot (\mathbb{P}(c_1 \notin \mathcal{K}|Label = \mathcal{N}) + \mathbb{P}(c_1 \in \mathcal{K}|Label = \mathcal{N}) \cdot \mathbb{P}(d \leq \theta_{b_i}|c_1 \in \mathcal{K}, Label = \mathcal{N})). \quad (5.9)$$

Replacing Eq.(5.9) into Eq.(5.7), we obtain the expected number of true negatives over all the possible closest base classes, $\mathbb{E}[TN]$.

top-1, closest-c. As opposed to the top-k closest-1 case, in top-1 closest-c we look if any of the c closest base classes $\mathcal{C}$ has the highest Softmax value $k_1 = \max(\mathcal{K})$ for a given image.

As before, in the first step the image passes through the CNN and if the highest value of the Softmax does not belong to one of the closest c classes, then the image is immediately discarded. Otherwise, in the second phase the value of its distance to the mean of the base class with the highest Softmax value against the corresponding threshold is compared. Based on the outcome of the comparison, the image is either classified as containing the novel object or not.

Let $\mathcal{C} = \mathcal{C}_0$ denote one of the possible combinations of the order of the closest c base classes. For that outcome, we have the following:

Result 5 *The expected number of true positives, given that the set of closest c classes is $\mathcal{C} = \mathcal{C}_0$, is*

$$\mathbb{E}[TP|\mathcal{C} = \mathcal{C}_0] = n\mathbb{P}(Label = \mathcal{P})\mathbb{P}(k_1 \in \mathcal{C}_0|Label = \mathcal{P}) \cdot \mathbb{P}(d > \theta_{k_1}|k_1 \in \mathcal{C}_0, Label = \mathcal{P}). \quad (5.10)$$

Note that we denote by θ_{k_1} the base class in the closest set that has the highest Softmax for the image. Other than that, the same explanation holds as for the previous cases in terms of how the images are classified as true positives.

The expected number of true positives over all the possible combinations of closest base classes $\mathcal{C}$ is

$$\mathbb{E}[TP] = \sum_{\mathcal{C}_0} \mathbb{E}[TP|\mathcal{C} = \mathcal{C}_0]\mathbb{P}(\mathcal{C} = \mathcal{C}_0), \tag{5.11}$$

where $\mathbb{E}[TP|\mathcal{C} = \mathcal{C}_0]$ is given by Eq.(5.10).

Similarly, for the true negatives we have:

Result 6 *The expected number of true negatives, given that the set of closest c classes is $\mathcal{C} = \mathcal{C}_0$, is*

$$\begin{aligned}\mathbb{E}[TN|\mathcal{C} = \mathcal{C}_0] = n\mathbb{P}(Label = \mathcal{N}) \cdot (\mathbb{P}(k_1 \notin \mathcal{C}_0|Label = \mathcal{N}) + \\ \mathbb{P}(k_1 \in \mathcal{C}_0|Label = \mathcal{N}) \cdot \mathbb{P}(d \leq \theta_{k_1}|k_1 \in \mathcal{C}_0, Label = \mathcal{N})).\end{aligned} \tag{5.12}$$

The expected number of true negatives over all the possible combinations is then obtained from

$$\mathbb{E}[TN] = \sum_{\mathcal{C}_0} \mathbb{E}[TN|\mathcal{C} = \mathcal{C}_0]\mathbb{P}(\mathcal{C} = \mathcal{C}_0), \tag{5.13}$$

where $\mathbb{E}[TN|\mathcal{C} = \mathcal{C}_0]$ is given by Eq.(5.12).

top-k, closest-c. As its name suggests, in the last step of the CNN we check whether the highest k-values of the Softmax output correspond to one or more of the closest base classes or not. If not, the image is discarded. If they do, then in the second stage for the (closest) classes that are in the highest k Softmax classes we perform the comparison against the threshold value of the corresponding base classes (see Fig. 5.4). It suffices for one of the distance comparisons $d > \theta_b$ to hold, for the image to be classified as belonging to the novel class. Only if none of the comparisons with the closest base classes that are in top-k Softmax do not result in a novel class, then that image is discarded (declared to be a negative).

An image will undergo the second phase if and only if the following holds

$$\exists c \in \mathcal{K} \cap \mathcal{C},$$

where $c \in \mathcal{C}$. The previous relation says that a base class that is in closest-c classes will pass the first phase if it is in top-k.

Let $\mathcal{C} = \mathcal{C}_0$ denote one of the possible combinations of the order of the closest c base classes. Following the above reasoning, for that outcome we have (for more details see Appendices A, B, and C):

Result 7 *The expected number of true positives for a given set of closest base classes*

$\mathcal{C} = \mathcal{C}_0$ *is obtained from*[2]

$$\mathbb{E}[TP|\mathcal{C} = \mathcal{C}_0] = n\mathbb{P}(Label = \mathcal{P}) \cdot \mathbb{P}(\exists c \in \mathcal{K} \cap \mathcal{C}|Label = \mathcal{P}) \cdot \mathbb{P}(\exists c : d > \theta_c | c \in \mathcal{K} \cap \mathcal{C}, Label = \mathcal{P}). \quad (5.14)$$

The expected number of true positives over all the possible combinations of closest base classes $\mathcal{C}$ is

$$\mathbb{E}[TP] = \sum_{\mathcal{C}_0} \mathbb{E}[TP|\mathcal{C} = \mathcal{C}_0]\mathbb{P}(\mathcal{C} = \mathcal{C}_0). \quad (5.15)$$

Essentially, in the last probability term of Eq.(5.14), $\exists c : d > \theta_c | c \in \mathcal{K} \cap \mathcal{C}$ captures the condition that if at least in one of the comparisons of the closest classes that are in top-k Softmax against the threshold the image is found to not belong to the base class, it is classified as a true positive.

The expected number of false negatives can be derived from Eq.(5.15) as

$$\mathbb{E}[FN] = n_q - \mathbb{E}[TP]. \quad (5.16)$$

In the case of true and false positives, the output of the comparison against the base classes in the second phase is essentially an OR operation (it suffices one of the comparisons to hold for the image to be classified as belonging to the novel class). As opposed to these two, when it comes to true and false negatives, there is an AND relation. Namely, for an image that has passed the Softmax phase in order to be considered as not belonging to the novel class, *for all* the closest base classes in top-k Softmax the comparison with the corresponding threshold values must result in a negative output. Hence, the AND relation.

Following the above reasoning, we have:

Result 8 *The expected number of true negatives when the set of the closest base classes is* $\mathcal{C} = \mathcal{C}_0$ *is obtained from*

$$\mathbb{E}[TN|\mathcal{C} = \mathcal{C}_0] = n\mathbb{P}(Label = \mathcal{N})(\mathbb{P}(\nexists c \in \mathcal{K} \cap \mathcal{C}|Label = \mathcal{N}) + \mathbb{P}(\exists c \in \mathcal{K} \cap \mathcal{C}|Label = \mathcal{N}) \cdot \mathbb{P}(\forall c : d \leq \theta_c | c \in \mathcal{K} \cap \mathcal{C}, Label = \mathcal{N})). \quad (5.17)$$

[2]In order for an image to be classified as true positive (negative), the first condition it must meet is to have its ground truth be "positive" ("negative"), i.e., $Label = \mathcal{P}$ ($Label = \mathcal{N}$).

The expected number of true negatives over all the possible combinations is given by

$$\mathbb{E}[TN] = \sum_{\mathcal{C}_0} \mathbb{E}[TN|\mathcal{C} = \mathcal{C}_0]\mathbb{P}(\mathcal{C} = \mathcal{C}_0). \tag{5.18}$$

The expected number of false positives can be derived from Eq.(5.18) as

$$\mathbb{E}[FP] = N_b - \mathbb{E}[TN]. \tag{5.19}$$

These expected values are used to estimate various image classification metrics, based on what the query issuer specifies.

As a final note, when dealing with a resource-constrained environment, the best option is to choose $k = 1$, $c = 1$, since the smallest number of images will be uploaded to the server. However, this option always gives *the smallest number of true positives*. In Section 5.6.2 we provide an in-depth numerical example of a server's setup for the values of c and k.

5.4.1.3 Calculating θ

Ideally, if a large number of samples from the novel class is available, and assuming that the distributions of both the closest base class distance and the novel class distance from the MAV of the closest base class are uniform, we can conclude that the optimal threshold for a base class is the point where the "normalized" histograms of the base and novel classes intersect. We note that our system aims to classify novel classes given a small number of samples of these novel objects, meaning that creating a distribution from such samples is not efficient. Based on that, we choose to use the second filter (checking if $d > \theta_b$, where $d = dist(MAV_b, f(x))$) to determine if the image belongs to the base class distribution or not. If the image does not belong to the base class, we classify it as belonging to the novel class.

In practice, when measuring the distance between a sample of the base class and the MAV of that base class itself, the distance tends to be small and the distribution is *right-skewed*. We also note that the samples of the novel class have a larger distance from the MAV of the base class compared to samples belonging to that base class. Therefore, the samples of the novel class fall to the right of the base class distribution, as can be observed from Figs. 5.4 and 5.5. The mean and standard deviation of the base class distribution are used to set the threshold, which is defined as

$$\theta_b = \mu_b + \alpha\sigma_b, \tag{5.20}$$

where α is an empirical *tuning coefficient* whose value is bounded by the increase in the false positive and true positive rates, μ_b and σ_b are the mean and the standard deviation of the distance of the base class samples to itself, respectively.

Since the base classes are previously trained for, the server has access to the training samples for these classes. Hence, $MAV_{b\in\mathcal{B}}$ and $\theta_{b\in\mathcal{B}}$ are calculated and stored on the server.

5.4.1.4 Server algorithm

The process occurring on the server side, when a query is issued, is shown in Algorithm 5.

Algorithm 5: SERVER_SIDE ()

stored values : CNN, $MAV_{b\in\mathcal{B}}$, $\theta_{b\in\mathcal{B}}$
input : samples of q

1 $MAV_q = \frac{1}{n_q}\sum_{x\in\mathcal{X}_q} f(x)$;
2 $\mathcal{D} = \text{dist}_{b\in\mathcal{B}}(MAV_q, MAV_b)$;
3 sort $\mathcal{D}$ in ascending order $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{B}|}\}$;
4 //set parameters: k and c (from Section 5.4.1.2)
5 $\mathcal{C}$ = the first c base classes based on $\mathcal{D}$
6 **return** $\mathcal{C}$, $MAV_{b\in\mathcal{C}}$, $\theta_{b\in\mathcal{C}}$, k

Complexity. The complexity of computing MAV_q in line 1 is $O(n_q)$, whereas the complexity for computing the distance in line 2 is $O(|\mathcal{B}|)$. Sorting in line 3 induces $O(|\mathcal{B}|log|\mathcal{B}|)$ complexity, and line 4 requires kc operations. Hence, it follows that the complexity of Algorithm 1 is $O(n_q + |\mathcal{B}| + |\mathcal{B}|log|\mathcal{B}| + kc) = O(n_q + |\mathcal{B}|log|\mathcal{B}| + kc)$.

5.4.2 Edge devices

As Fig. 5.3 shows, every edge device receives the following parameters from the server: the closest base classes along with their MAVs and thresholds, as well as the value of k. The edge device processes its (stored) images one by one. Each stored image goes through the pre-trained pre-loaded CNN. Using the Softmax layer, the object is classified as one of the base classes. Based on this result, if the sample is not one of the closest classes, the classification remains as is. Otherwise, the image goes through a second filter, and is classified as either belonging to the novel class or to the base class, based on its

distance from the MAV of the base class, i.e, θ. Algorithm 6 illustrates the process on the edge device.

Algorithm 6: EDGE_DEVICE ()

```
   input  : CNN, C, MAV_{b∈C}, θ_{b∈C}, k, image x
   output : upload x to server if containing novel object
 1 if any top-k classes == c ∈ C then
 2 |   for i=0; i<|K|; i++ do
 3 |   |   if top-i == c ∈ C then
 4 |   |   |   c //closest class that is in top-k
 5 |   |   |   d = dist(f(x), MAV_c);
 6 |   |   |   if d > θ_c then
 7 |   |   |   |   x belongs to class q
 8 |   |   |   |   //send to server
 9 |   end
10 else
11 |   image is not classified as the novel class q
12 end
```

Complexity. For every image in the device queue, there is a maximum of kc comparisons after the image goes through the CNN. The complexity of Algorithm 2 is hence $O(kcL)$, where L is the number of images stored on the device.

5.5 Design Considerations

In this section, we discuss the choice of closest base classes, and illustrate how the decisions on the threshold values are made. In the remainder of this chapter we use two datasets CIFAR-100 and CIFAR-10 [13]. CIFAR-100 has 100 classes, each of which contains 500 training images and 100 test images. We use these 100 classes as our base classes. CIFAR-10 has 10 classes, with $1,000$ test images and $5,000$ training images per class. We use these classes as the novel classes.

5.5.1 Closest base classes and thresholds

For the discussion in this section, unless stated otherwise, we use a single run utilizing 10 random samples of the novel class. The closest base classes are calculated according to the analysis presented in Section 5.4.1.1. Table 5.2 shows the closest base classes when $c = 3$ for the different novel classes (CIFAR-10 classes). The numbers between the

Table 5.2. Closest base classes to the novel class

Novel class	c_1	c_2	c_3
airplane	otter(55)	bridge (12)	whale (95)
automobile	pickup truck (58)	bus (13)	train (90)
bird	otter (55)	lizard (44)	seal (72)
cat	possum (64)	wolf (97)	mouse (50)
deer	kangaroo (38)	rabbit (65)	cattle (19)
dog	rabbit (65)	cattle (19)	fox (34)
frog	lizard (44)	turtle (93)	snake (78)
horse	cattle (19)	camel (15)	kangaroo (38)
ship	tank (85)	bridge (12)	sea (71)
truck	pickup truck (58)	bus (13)	tractor (89)

brackets are class id's corresponding to the class labels. For example, the class 'ship' has class 'tank' as its closest base class, i.e., the distance from the MAV_{ship}, created using the 10 samples from the 'ship' class, and the MAV_{tank}, created using the 500 available training samples, is the smallest compared to the other MAV_b, where $b \in \mathcal{B}$.

In Figs. 5.5 and 5.6 we illustrate the intuition for picking the closest base classes to the novel class, and for setting distance thresholds. For illustrative purposes, 500 samples per class are used in both Fig. 5.5 and Fig. 5.6. We only use 10 images for the novel classes when performing actual experiments presented later.

Fig. 5.5 shows two distributions. The first distribution is the distance between the images belonging to the class 'tank' and MAV_{tank}. The second distribution shows the distance of samples belonging to the novel class 'ship' from MAV_{tank}. We clearly see that the distribution for 'tank' is skewed to the left, and the majority of images of tank are at a very small distance from the MAV of the tank class. The two distributions are distinguishable, with a small area of overlap between them, where we have the false negatives and false positives. Fig. 5.5 also shows the threshold calculated using Eq.(5.20), for $\alpha = 3$. Images to the right of the threshold ($d > \theta_{tank}$) are classified as 'ship'.

Choosing a class that is further away from the novel class might seem a good choice to use as the first filter, as fewer or no images would overlap between the two distributions. In Fig. 5.6, we choose class 'apple' and calculate the distances from MAV_{apple}. It is obvious that samples from the 'ship' class are easily distinguishable from samples of the 'apple' class, because they are far distance-wise from MAV_{apple}, and they fall to the right of the threshold ($d > \theta_{apple}$). However, if we choose to compare the novel class to an MAV of a high-distance class, images from base classes that are close in distance to the novel class will all be classified as belonging to the novel class incorrectly. In this case,

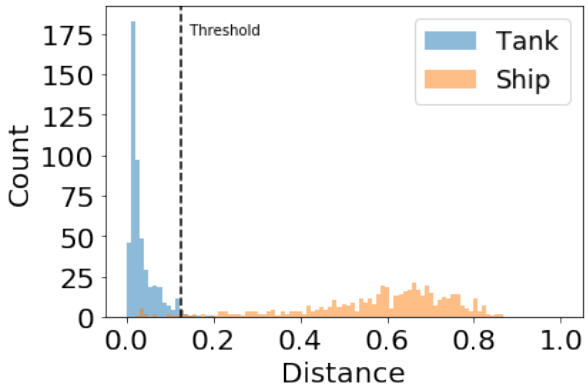


Figure 5.5. Distance to MAV_{tank}: left of threshold is classified as 'tank', right of threshold is classified as 'ship'.

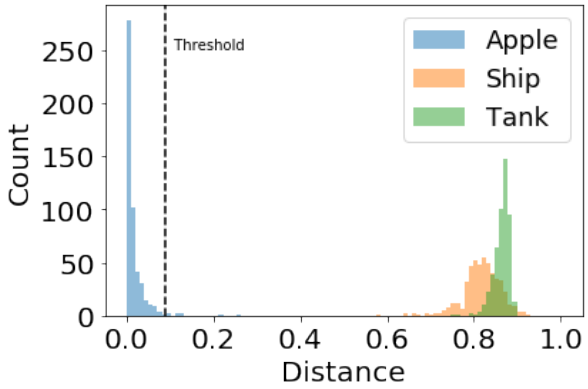


Figure 5.6. Distance to MAV_{apple}.

the images from the class 'tank' all fall to the right of the threshold as well, and will be false positives. Therefore, we choose the closest base classes in the first phase of the algorithm, and we choose the threshold to be based on the base class distribution as we have a large number of training samples for these classes, contrary to the novel class where we only have a small number of samples.

Table 5.2 shows that the closest base classes to 'ship' (novel class) are 'tank', 'bridge', and 'train' (base classes). In Fig. 5.7 we present a histogram of the Softmax classification of the 500 images of 'ship' from the CIFAR-10 training after the images go through ResNet-34, which has been trained only on the base classes. The images cannot be directly classified as 'ship' since the CNN is not trained on that class. They will be classified as one of the base classes. As Fig. 5.7 shows, the largest number of 'ship' images are classified as class 85, which corresponds to 'tank', followed by class 12 that corresponds to 'bridge', which is consistent with Table 5.2. It is also clear that the ship

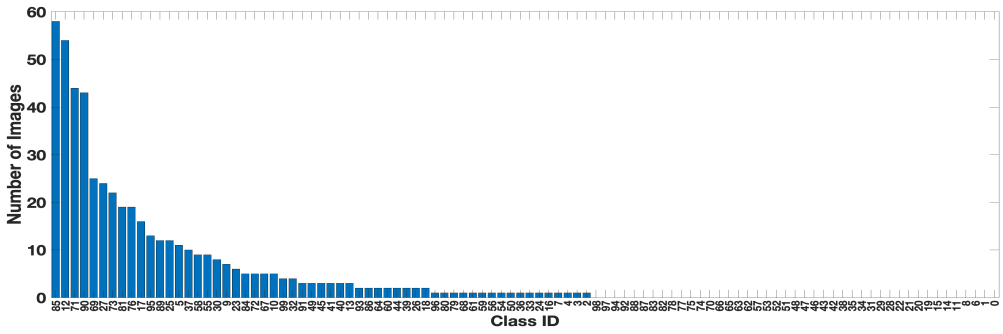


Figure 5.7. Classification of 'ship' samples ($k = 1$).

images are classified as four of the base classes a significant amount of time, motivating the need for using the closest c base classes, instead of only using the single closest base class. Fig. 5.7 also shows that by setting $c = 4$ and $k = 1$, about 45% of the images will pass the first filter and will be considered in the second stage. For an exact mapping between class label and class number, the interested reader is referred to CIFAR-100 dataset [13]. We realize that the order of closest classes may vary based on the number of samples as well as on the specific samples of the novel class available. Hence, the option of setting $c > 1$. However, in most cases, the actual closest class appears as one of the closest five classes. Therefore, the choice of limiting the value of c to 5.

5.6 Performance evaluation

In this section, we implement EDIR and demonstrate the use of our analysis in choosing the values of the parameters c and k. We note that the values of these parameters affect the number of images uploaded to the server and the F1 score. Following that, we compare EDIR to another state-of-the-art novel detection algorithm [26]. Finally, we show the effect of varying the number of available samples from the novel class.

5.6.1 System Implementation

We implemented both EDIR and the system of [26] using Samsung Galaxy S9 mobile devices and a Dell Precision T7500 with GeForce GTX TITAN X 12 GB GPU as the server.[3]

[3]We are considering a kind of worst-case scenario in terms of bandwidth, hence the competition is already captured by this congested scenario. As a result, it suffices to show the performance on a single phone.

In order to evaluate our algorithms, we use publicly available benchmark datasets. The two datasets we use are CIFAR-100 and CIFAR-10 [13]. For the pre-trained model we use ResNet-34 [60], which is trained on CIFAR-100, and is available in [98]. We draw novel classes from CIFAR-10. Positively-classified images are uploaded to the server using LTE or WiFi. Throughout this section, we assume that the user has 10 samples of the novel class, unless stated otherwise.

5.6.2 Choosing parameters c and k

Using the analysis from Section 5.4.1.2, we choose c and k based on the metric we want to optimize, whether that is constraining the number of uploads if the available bandwidth is limited, increasing the F1 score, or increasing the recall (TPR) while limiting false positive rate (FPR) to a specific value. In this section, we show that this estimated best set of values for c and k is close to the best set obtained experimentally.

Initially, the server has access to a number of images of the novel class. Those images are used to estimate the following parameters: TPR, FPR, Precision and F1 score, for various values of c and k based on the equations from Section 5.4.1.2 and Section 5.2. To estimate these parameters, the server sets the positive population to be the set of available images from the novel class, whereas the negative population consists of the training images for the base classes that have been used for training the CNN, because those images are available on the server.

We start by evaluating the server's decision of the values of the parameters c and k on the full training set. After that, we show the results when the server has a limited number of samples from the novel class (in this case 10 samples).

5.6.2.1 Full Training Dataset

To clearly illustrate the process of choosing c and k, we start by demonstrating a single instance (run) using the full training set for the novel class 'ship'. Using these $5,000$ samples, the server calculates MAV_{ship}. Then, based on that, it finds the closest base classes. We choose to limit the value of c to 5 as every added closest class increases the amount of data sent from the server to the mobile device. We also limit the value of k to 5, as increasing the value of k increases the number of TPs and FPs, and as a result, the number of images uploaded to the server.

The server estimates the best values for c and k, calculated based on the equations presented in Section 5.4.1.2. In this example, assuming the requester is looking to achieve

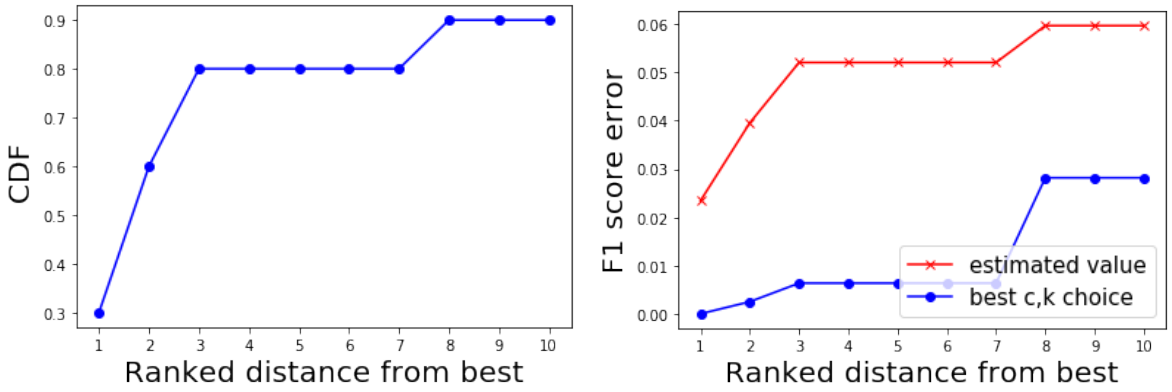
a high F1 score, the values of $c = 3$ and $k = 3$ are chosen by the server.

We then compare how well our selection of parameters performs against the best possible selection obtained experimentally. To find the actual best set of values for c and k, we perform experiments to determine the values of Recall, Precision and the F1 score for every combination of values of c and k using the $10,000$ images from the test set of the base classes and $1,000$ images from the test set of the novel class 'ship'.

Table 5.3 illustrates the server's estimated results when varying c and k. Table 5.4 shows the top-10 best choices in terms of highest F1 score. In this scenario, the best choice would match the parameters chosen by the server, where $c = 3$ and $k = 3$.

As mentioned previously, the choice of c and k affects both the number of uploads and the F1 score. We note that setting $c = 1$ and $k = 1$ always achieves the smallest number of image uploads, and highest precision.

In order to fully evaluate the suitability of our parameter selection analysis, we tested all 9 novel classes (CIFAR-10 classes excluding 'automobile', since class 'streetcar' exists in CIFAR-100).



(a) CDF of ranked distance away from best

(b) F1 error

Figure 5.8. Error in chosen parameters, full training set.

Fig. 5.8(a) illustrates the top 10 results for the Cumulative Distribution Function (CDF) of the distance of the estimated parameters from the best set of parameters found experimentally. Fig. 5.8(b) compares the performance of the estimated best choice for c and k with the values determined by the exhaustive search of $11,000$ images. For instance, from Tables 5.3 and 5.4 it can be observed that for class 'ship' the parameters estimated by the server match the best pair of c and k obtained experimentally. Hence, the ranked distance from best is 1, and the F1 score error from best choice is 0, and the F1 score error from the estimated value is 0.039. The results show that 80% of the time

Table 5.3. Estimation results when varying c and k for the full training set

c	k	TP	RECALL	FP	FPR	PRECISION	F1
1	1	336	0.0672	4	0.00008	0.98824	0.12584
1	2	739	0.1478	275	0.0055	0.72880	0.24576
1	3	1109	0.2218	726	0.01452	0.60436	0.32451
1	4	1436	0.2872	1337	0.02674	0.51785	0.36948
1	5	1718	0.3436	2094	0.04188	0.45068	0.38992
2	1	865	0.173	15	0.0003	0.98295	0.29422
2	2	1556	0.3112	469	0.00938	0.76840	0.44299
2	3	2058	0.4116	1184	0.02368	0.63479	0.49939
2	4	2470	0.494	2237	0.04474	0.52475	0.50891
2	5	2788	0.5576	3466	0.06932	0.44579	0.49547
3	1	1139	0.2278	21	0.00042	0.98190	0.36981
3	2	2025	0.405	726	0.01452	0.73610	0.52251
3	3	2618	0.5236	1736	0.03472	0.60129	0.55976
3	4	3043	0.6086	3186	0.06372	0.48852	0.54199
3	5	3383	0.6766	4746	0.09492	0.41616	0.51535
4	1	1343	0.2686	33	0.00066	0.97602	0.42127
4	2	2293	0.4586	1444	0.02888	0.61359	0.52489
4	3	2902	0.5804	2902	0.05804	0.50000	0.53721
4	4	3286	0.6572	4249	0.08498	0.43610	0.52429
4	5	3576	0.7152	5676	0.11352	0.38651	0.50182
5	1	1710	0.342	59	0.00118	0.96665	0.50524
5	2	2792	0.5584	2638	0.05276	0.51418	0.53538
5	3	3419	0.6838	4696	0.09392	0.42132	0.52139
5	4	3808	0.7616	6557	0.13114	0.36739	0.49567
5	5	4061	0.8122	8218	0.16436	0.33073	0.47005

Table 5.4. Experimental results: top-10 F1 scores

c	k	TP	RECALL	FP	FPR	precision	F1
3	3	530	0.53	507	0.0507	0.51109	0.52037
4	3	599	0.599	716	0.0716	0.45551	0.51749
3	4	615	0.615	770	0.077	0.44404	0.51572
4	4	671	0.671	966	0.0966	0.40990	0.50891
5	2	562	0.562	670	0.067	0.45617	0.50358
4	2	488	0.488	451	0.0451	0.51970	0.50335
5	3	688	0.688	1049	0.1049	0.39609	0.50274
3	2	428	0.428	281	0.0281	0.60367	0.50088
3	5	673	0.673	1056	0.1056	0.38924	0.49322
4	5	718	0.718	1218	0.1218	0.37087	0.48910

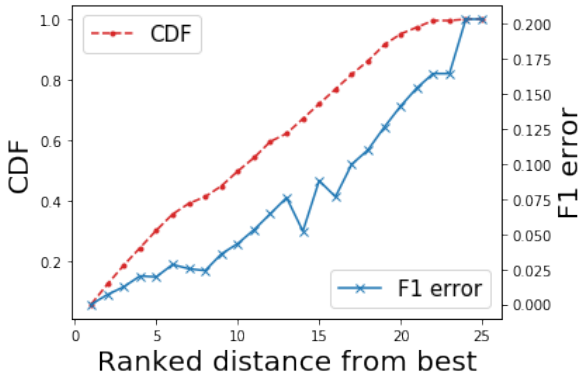


Figure 5.9. Error in chosen parameters, 10 samples.

the estimated best set of parameter values are within the best three selections with an F1 score error of less than 0.01.

5.6.2.2 Small Number of Samples

In this set of experiments, we assume the server has a small number of samples from the novel class (10 samples). These samples are used for choosing the values of c and k, similarly to Section 5.6.2.1. We evaluate the parameter selection by issuing a query for each of the nine novel classes. For each query we execute 25 random runs, with a total of 225 runs.

Given the large number of available negative samples on the server (500 images per base class, i.e., $50,000$ images in total), we decided to constrain the negative population to only 500 randomly chosen images. We made this decision in order to avoid having overly optimistic and misleading results that show very small false positive rates. We note that the limited number of positive samples affects the estimations as it is hard to accurately estimate the true positive rate from such a small sample, and that can introduce some errors in our estimations.

The goal of the system is to choose parameters that decrease the amount of FP uploads to the server. However, in order to increase the amount of TP, we try to achieve a higher F1 score. More details on this can be found in Section 5.6.3.

Fig. 5.9 shows that 55% of the time the algorithm chooses parameters that achieve an F1 score that is within 5% of the best choice, and is within the best 11 choices, and 82% of the time the margin of error is within 10%.

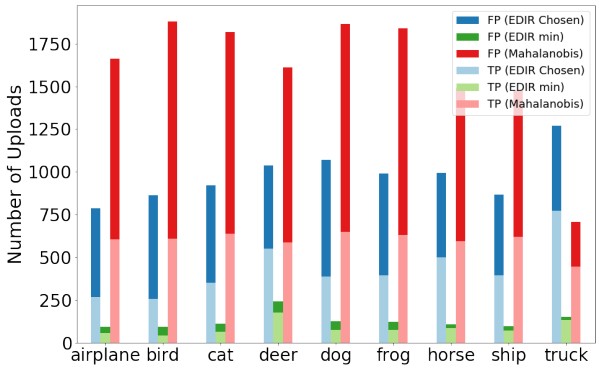


Figure 5.10. Number of uploads (TP and FP) for different novel classes.

5.6.3 Comparison to Mahalanobis-distance algorithm

In [26], the authors propose a novelty detection algorithm, where images are classified as not belonging to the closed set of classes in a trained CNN. They extend their algorithm and use Mahalanobis distance to do incremental learning for novel classes. The authors use the full training set for the novel classes. However, in our implementation, for a fair comparison, we generate the covariance matrix and class means using only the small set of available images from the novel class (in this case 10 images). Our algorithm adds flexibility in the sense that we can vary the choice of c and k to limit the uploads, especially the number of FPs. On the other hand, [26] has no such flexibility.

To compare to [26], we use two instances of EDIR:

- **EDIR-Min**: The more restricted energy and bandwidth option, by setting $c = 1$ and $k = 1$.
- **EDIR-Chosen**: It uses the server's chosen c and k values, as shown in Section 5.6.2.

The main goal is to respond to the query and upload images containing objects of the novel class in a bandwidth-constrained environment. Our goal is to reduce the number of FPs. Fig. 5.10 and Fig. 5.11 show the trade-off between the number of images that are uploaded and the F1 score when using EDIR and Mahalanobis distance-based approach and further shows how EDIR can adjust this trade-off. Fig. 5.10 shows the number of images uploaded to the server for each of the three algorithms when querying for different novel classes. EDIR-Min uploads the least amount of images and achieves the highest precision (ratio of correctly uploaded images by all uploads). On the other hand, Mahalanobis distance-based approach achieves the highest recall (rate of correctly positive images uploaded) with the drawback of uploading the most FPs.

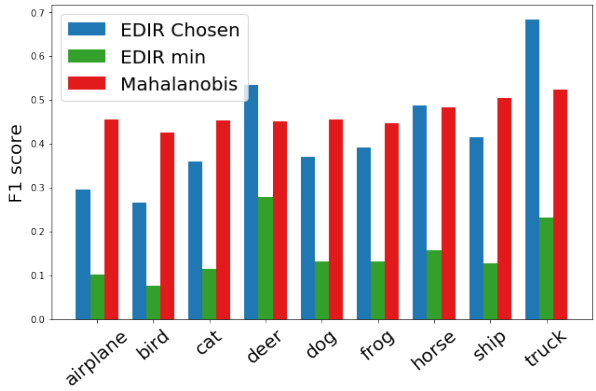


Figure 5.11. F1 score for different novel classes.

In order to achieve a recall and precision balance, the F1 score is optimized. In Fig. 5.11 we show the F1 score for each of the algorithms. We observe that Mahalanobis has the highest F1 score, but that when compared to EDIR-Chosen, over the 9 novel classes, the average difference in F1 score is only 10%. On the other hand, the number of FP uploads when using Mahalanobis-based approach is 4.5× higher than when using EDIR-Chosen. This shows that EDIR is much more efficient than Mahalanobis and only incurs a small reduction in the F1 score.

It is not desirable to upload a large number of false positives in networks with limited bandwidth. An important advantage of EDIR is its flexibility in controlling the number of TP and FP images by properly setting c and k. Increasing k increases both the number of TPs and FPs, resulting in an increase in the number of images uploaded to the server. Further, increasing c increases both the downloads from the server, where more MAVs have to be sent, and also increases the number of TP and FP images uploaded to the server. Nonetheless, increasing these values may increase the F1 score, thus providing the user with the flexibility in trading off resource constraints with performance. The Mahalanobis distance-based algorithm [26] does not have the ability to tune its parameters in order to adjust to the actual network conditions.

5.6.3.1 Bandwidth and Energy savings

When using EDIR, the MAV of every closest c class, the threshold of every closest c class, the labels of the closest c classes, and the value of k must be transferred to the user. Using ResNet-34, the output vector size of the extracted features is 1×512, meaning that every MAV is of that dimension, and is 13 kB (bytes). The values of the thresholds are $c * 25$ B, the labels of the closest base classes are $c * 25$ B, and the value of k is 25 B.

Table 5.5. Amount of data sent from the server to a device

Algorithm	Data
EDIR-Min	13 kB
EDIR-Chosen	39 kB
Mahalanobis-distance	7.9 MB

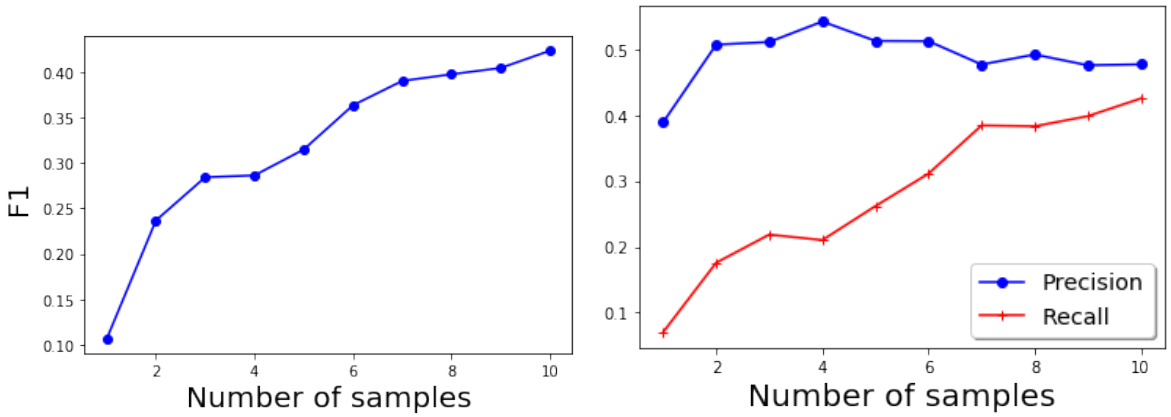
The basic premise of [26] is that the posterior probability of a CNN classifier with a softmax layer is equivalent to linear discriminant analysis (LDA) in a CNN-based feature space associated with the penultimate layer. The mean vectors along with the common covariance matrix are required to perform LDR by calculating the Mahalanobis distance. Therefore, when using the Mahalanobis distance-based algorithm [26], the server must send the the shared covariance that has dimensions 512×512 and is 6.7 MB, along with the class means with dimensions 101×512 and a total size of 1.3 MB. The amount of uploaded data to the server in both systems is $(TP + FP) * img\_size$.

Table 5.5 summarizes the amount of data that each algorithm sends to the device when a query is issued. We can see that EDIR-min and EDIR-Chosen send 99% less data to the device than the Mahalanobis algorithm. Fig. 5.10 illustrates the number of images that are uploaded to the server (the number of TP and FP images) for each algorithm. Uploading images to the server in wireless networks is the main source of energy drainage, especially when using outdoor networks, like LTE [68]. The number of uploaded images in response to the query is much higher when using the Mahalanobis algorithm than when using EDIR. Uploading FPs represents a waste of energy and network resources, thus EDIR is much more efficient than the Mahalanobis algorithm. While both EDIR and Mahalanobis require the same energy to run the CNN, EDIR uses 45% less energy and bandwidth for FP uploads, and 99% less energy and bandwidth on downloads when the query is issued.

5.6.4 Varying the number of novel samples

To demonstrate the effectiveness of our algorithm, and show how it performs with a small number of available samples from the queried novel class, we vary the number of samples from 1 to 10. Fig. 5.12(a) shows the F1 score using the values of c and k chosen by the server, by averaging over 10 runs for each of the 9 novel classes (total of 90 random runs per number of samples). Similarly, Fig. 5.12(b) shows both the precision and recall when varying the number of samples. Given the nature of our problem and since we are dealing with closed sets, F1 score is the more meaningful measure. For each run we randomly

choose the initially available image samples that the server has access to when a query is issued. The sample images from the novel class are used to identify the closest base classes. We realize that as the number of images increases the F1 score increases as well because the server is able to better identify the closest base classes, and can properly tune c and k. We extended the experiment and varied the number of samples up to 20 samples and noticed the diminishing return property in terms of the F1 score with the increase in the number of samples, after 10 samples.



(a) F1 Score (b) Precision and Recall

Figure 5.12. Varying the number of samples.

EDIR has the advantage of being able to run with no intensive training by only having access to a limited number of sample images of the novel class, even as low as a single sample, whereas other methods such as Mahalanobis [26] cannot do that. Despite having a small F1 score when the number of samples is low, EDIR still achieves a good precision, for example 0.51 when the number of samples is 2 only, therefore utilizing efficiently the bandwidth of the system since the majority of uploads are TP instead of FP.

5.7 Related Work

Image and video search in distributed mobile networks is an active area of research. In [99], the authors address the problem of re-identification, where the image of a person or an object captured by mobile cameras is matched to images stored in a database in a central server. In other scenarios, queries are issued and devices with stored videos or images respond to the query by searching in their stored data. In PicSys [68], the aim is to minimize the query response time for collecting useful images over all devices in the

system, given specific energy constraints. The usage of early exiting from CNNs, local processing and offloading to the server for faster processing are used in [68]. Focus [100] utilizes initial cheap processing at ingest times to reduce latency at query time for large datasets of videos. In Netvision [67], algorithms for optimal video offloading are proposed to reduce query response time. The objective of our work is also to respond to similar queries. However, our goal is different in that the query is issued for a novel class that the previously deployed CNNs are not trained for. Our work also considers low bandwidth networks and aims to reduce the amount of transmitted data between the server and mobile devices.

Deep learning models and **CNNs** have excelled in the object classification task irrespective whether they are used on devices with high computational power [3], [12], [60], or on mobile devices with lower computational and memory resources [23], [22]. Training CNNs is a computationally intensive task and requires a large number of labeled data in order to achieve a high accuracy.

Transfer learning methods are widely used to address the problem of limited labeled data. They utilize the fact that initial layers capture generic features, while later layers capture more specific features [101], enabling pre-trained models to serve as a basis for training new models on new classes [102]. These methods still require a high number of labeled data, and result in new trained CNN models that require transmission to edge devices where classification takes place.

To address the problem of limited training data, **few-shot** learning was introduced. Few-shot learning aims to train models that are able to generalize to new environments or tasks [103]. A task in few-shot learning includes a *support set* and a *query set*. The support set consists of n-samples for each of new k-classes (this problem is called n-shot, k-way). The query set consists of a number of unlabeled images that belong to one of the k classes. Validation sets are also used during training to evaluate epoch performance, and help in deciding upon the best performance outcome. The datasets used for few-shot learning, such as Omniglot [104] and miniImageNet [105], have a high number of classes to enable the generation of enough tasks to avoid over-fitting. Different methods of few-shot learning are used, among which the best known are *metric learning* [105], [106], and *meta-learning* [107].

Our work does not fit in the few-shot learning problem, because in our setup we face a more challenging scenario. This is due to the limited number of images from the query set, and also we are not able to generate enough training tasks. We also query for a single new class, as opposed to matching or classifying images to be one of a set of new

classes.

Zero-shot learning corresponds to the case when no labeled training samples are used to learn or classify a new task. This is done by training descriptors [108]. DeViSE [109] and [110] both use visual-to-semantic word space mapping and language models for classification. Initially, a language model on a text corpus is trained. After the Softmax layer in the classification model, the output is mapped into a word vector, enabling the model to find words that are close in the embedding space. Nevertheless, the issue with such methods is that they require model training, and aside from the size of the classification model we need to account for the language model as well. Even if the language models are pre-trained, these models are large in size. For example, *wiki word vectors* in English [111] is 9.6 GB. Downloading such models to edge devices would consume a large amount of bandwidth and time, and storing them would take considerable space on limited-memory devices, if at all possible.

Data augmentation is also used in solving the problem of lack of image samples for classification [8]. Generative Adversarial Networks (GANs) [112] are used to generate image samples to aid in training deep learning models [113]. Data augmentation helps in alleviating the problem of needing a large amount of data for training new classes. However, training models and transmitting them through the network still remain an issue. In general, most of these methods entail some sort of training, which requires time and the re-transmission of new models through the mobile network.

Open set recognition is the problem of recognizing from a much larger universe than the model was trained for. In [114], an image is identified as anomalous, and later as more anomalous images are found, new classes are discovered using k-means clustering. In [115], the authors estimate the probability of an image being from an unknown class by replacing the Softmax layer with what they call an *Openmax layer* that includes a label of 'unknown'.

The problem of detecting **out-of-distribution (OOD)** samples is an active area of research [116], [117], [26]. In our work, we do not aim to classify an image as not belonging to the trained classes, or identify an image as being *misclassified* or *anomalous*. We aim to classify new novel classes on which the CNN has not been previously trained.

One-class classifiers are also used for normal/anomaly detection [118], [119], where the queried class is set as the normal class. However, these methods also require a high number of labeled samples for training.

Novelty detection is the identification of a new class for which a model was not trained. In [26], the authors propose a Mahalanobis distance-based score metric in feature

space to improve the accuracy of detecting outliers. They extend the work and use the Mahalanobis distance-based score for class-incremental learning to classify new classes by updating the shared covariance as samples of the novel class appear. They also compute the class mean. We provide a detailed comparison between EDIR and this algorithm in Section 5.6.3, showing that EDIR outperforms it [26], especially in terms of reducing the number of false positive uploads and minimizing energy consumption.

As opposed to other works, EDIR addresses the problem of image retrieval of novel objects in mobile networks by taking into consideration the restrictions in such networks, as it is difficult to train for new classes on these constrained mobile devices. The limited availability of samples of the novel class is also an issue that is taken into account by EDIR. Since transmitting newly trained models on networks with limited bandwidth is not feasible, EDIR enables setting parameters to adapt to the network resources and conditions.

5.8 Conclusion

In this chapter we presented EDIR, a system that enables efficient retrieval of images of novel objects from mobile devices. We propose a distance-based algorithm (Cosine distance) to achieve an accurate, energy- and bandwidth-efficient method for retrieving images that contain novel classes, with the use of a very small number of samples seen at the centralized location. Through extensive experimentation, we showed that our method is more flexible then another state-of-the-art image classification algorithm. We also presented the analysis that enables the query issuer to set different system parameters in order to allow for a better utilization of (often limited) network resources.

Chapter 6
VidQ

As mobile devices become more prevalent in everyday life and the amount of recorded and stored videos increases, efficient techniques for searching video content become more important. When a user sends a query searching for a specific action in a large amount of data, the goal is to respond to the query accurately and fast. In this chapter, we address the problem of responding to queries which search for specific actions in mobile devices in a timely manner by utilizing both visual and audio processing approaches. We build a system, called VidQ, which consists of several stages, and that uses various Convolutional Neural Networks (CNNs) and Speech APIs to respond to such queries. As the state-of-the-art computer vision and speech algorithms are computationally intensive, we use servers with GPUs to assist mobile users in the process. After a query is issued, we identify the different stages of processing that will take place. Then, we identify the order of these stages. Finally, solving an optimization problem that captures the system behavior, we distribute the process among the available network resources to minimize the processing time. Results show that VidQ reduces the completion time by at least 50% compared to other approaches [120].

6.1 Introduction

Users record and store large amounts of video files on their mobile devices. A method for efficiently searching for actions within these videos is highly desirable. Action recognition methods have been researched in the computer vision field [4]. However, these methods are associated with high computational cost, and are not suitable to run on all platforms, such as mobile devices [121]. When searching for actions within videos, there are various markers or cues that the action took place; spoken words, specific sounds, different objects or the action itself are all markers to search for when looking for an action of

interest.

Our goal is to utilize various media types (audio, images, video) that are contained within information-rich videos stored on mobile devices to help efficiently search for actions. In our setup, the mobile device has access to a server with a more powerful GPU over a wireless network (cellular, WiFi). While having a fast GPU on a server might entice the mobile users to send the complete video to the GPU for processing, this may not be possible or will not lead to improved performance. Factors that can impact the performance of offloading to network based processors include limited network capacity, the large size of videos, or a backlog of videos from many mobile devices at the GPU, causing an increase in the completion time of the process [68]. A more viable solution, which we explore in this chapter, is to split the video into shorter clips and then use a combination of mobile device processing and a server-based GPU to detect objects, sound, words or actions.

In our system, a user queries their device to search for a specific action within a set of videos. Along with the query, various keywords, sounds and objects that are associated with the action are specified. Three challenges need to be addressed in order to respond to such queries: (i) determining which markers are suitable for different types of events and video types, and based on that identifying the correct processing methods and models to use, (ii) identifying the correct order of processing in order to search for these markers (i.e., which action/object/sound to search for first), and (iii) choosing the proper entity to do the processing (mobile device or server).

The goal of the system is to process all the videos as fast as possible and identify the clips that contain the queried action. To aid in expediting the processing, we introduce a processing pipeline that consists of the processing stages that take place, in a specific order, to satisfy the query. The various processing stages used are: audio classification, speech-to-text, object detection, and action recognition. Each stage runs its own models on either the mobile device, the server, or both. Determining which processing stage to use for a clip and where to run it in order to both expedite the process and obtain correct answers is very important for different applications.

Specifically, the main contributions of our work are as follows:

- We propose a multi-media pipeline for efficient action recognition in stored videos using both audio and visual processing.
- We formulate an optimization problem to order the pipeline stages and decide where to run each stage (mobile device, server), whose objective is to minimize the

total completion time.

- We show that the optimization problem is NP-hard, and we propose a heuristic that reduces query response time by at least 50% compared to other methods.
- We implement our system and present detailed experimentation results, including some interesting engineering insights.

The remainder of this chapter is organized as follows. We present the system overview in the next section. Then, in Section 6.3 we describe each stage and measure the performance of the models in terms of speed, energy and accuracy. The system setup and the optimization problem are presented in Section 6.4. This is followed by the heuristic in Section 6.5 and performance evaluation results in Section 6.6. A description of some related work is provided in Section 6.7. Finally, Section 6.8 concludes this work.

6.2 System Overview

The system consists of a *mobile device*, which receives queries, either from its user or an external source, to find actions in stored videos, and a *server* containing GPUs for expedited processing. The processing is decomposed into a pipeline where each step in the pipeline searches for different markers of the desired action, such as a sound or word in the audio track, or an object or action in the video track. We consider searching for different markers, because as seen in the following sections, in many cases searching for markers is faster and requires less energy than searching for actions directly. The stages of the pipeline that search for sounds, objects or actions use their own trained models that are stored on the mobile devices and the server. A detailed discussion of the different processing stages is provided in Section 6.3.2.

Mobile Device. As shown in Fig. 6.1, videos are initially captured and stored locally on mobile devices. The user of the mobile device issues a query searching for an action within a large collection of its stored videos which can either be short video clips or long videos. The mobile device, in cooperation with the server, processes the videos and responds to the query by returning the videos or video clips that contain the queried action.

The mobile device is capable of running various Convolutional Neural Networks (CNNs) and speech models. This allows the mobile device to search videos for objects and sounds, or to convert speech to text and search for words or phrases. Currently, mobile devices are not capable of executing the models required to perform action

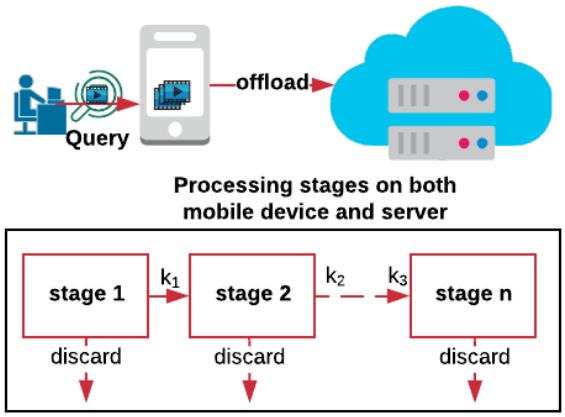


Figure 6.1. Overview of the system.

recognition [121]. However, even for processes that can take place on a mobile device, the processing times on mobile devices are long compared to the processing times on server GPUs. On the other hand, offloading videos to the server requires the upload of large files, which may add significantly to the completion time, and requires the availability of high network bandwidth, which is not always available, especially when the device is operating in an environment with many competing users for the network resources.

In order to expedite the process, mobile devices may act as a filter by performing local processing to reduce the number of videos or smaller audio files that must be uploaded to the server for further processing. In this way, the possibility of backlogging the GPU is reduced.

Server. As soon as the server receives a video or an audio file, it is capable to start processing it on its GPUs. More advanced processing techniques, such as 3D CNNs that perform action recognition, can be executed on servers. As will be seen in Section 6.3, the server can process audio 11× faster than the mobile device, and can process images 2.5× faster than mobile devices.

6.3 Experimentation

In this section, we present the details of the extensive experimentation we have conducted on each of the processing stages on both the mobile devices and GPUs, where applicable. The outcomes of these experiments support the design choices made for the system we present in the chapter, VidQ. These results serve as a guideline to decide which types of processing are suitable for different types of queries in order to meet goals for different

performance metrics detailed below, and thus shape the heuristic accordingly. We begin by introducing the main performance metrics used in this chapter. Then, we discuss each processing stage by introducing the models, APIs, and datasets used for each stage, along with the performance measures of the processing times on both the GPU and the mobile device, as well as the energy cost on the mobile device. Finally, we discuss examples of different types of queries that VidQ is capable of responding to.

6.3.1 Performance Metrics

In this section, we introduce the four main performance metrics used in this chapter: *accuracy*, *recall*, *precision*, and *F1 score*. First, we discuss the four auxiliary parameters for the calculation of these metrics: number of *true positives* (TP), *true negatives* (TN), *false positives* (FP), and *false negatives* (FN). Given a set of test data, where the truth values are known, TP and TN are the number of correctly predicted samples, i.e,. positives correctly labeled or classified as positives and negatives correctly labeled as negatives, respectively. FP and FN are the samples labeled incorrectly. Classifiers aim to minimize these last two values.

Accuracy is the ratio of correct predictions and the total number of samples. It is calculated as $\frac{TP+TN}{TP+FP+FN+TN}$. Recall, also known as the true positive rate (TPR), is the rate of correctly classified samples from the actual positives, and is calculated as $\frac{TP}{TP+FN}$. Precision is the ratio of correctly classified samples and the predicted positives, calculated as $\frac{TP}{TP+FP}$. Finally, the F1 score is the harmonic mean of precision and recall, and is calculated as $2 * \frac{Recall*Precision}{Recall+Precision}$.

A factor affecting these metrics is the choice of top-k values. In classifiers, such as CNNs, the output of the last layer is a probability distribution of each of the trained classes. The higher the probability, the more likely it is that the object/sound/action is included. The value of k is set as a threshold, meaning that if the class we are searching for is within the top k probabilities, the video is labeled as positive. In this chapter, we set $k = 1$, unless stated otherwise.

6.3.2 Processing Stages

In this section, we discuss each of the processing stages used in the system. We present a brief background on the processing mechanisms, introduce the models and datasets used, and discuss the performance measurements. We use off-the-shelf open source models and architectures [45], [6], [43], and [44], without the need to develop or train new models.

Table 6.1. Video and audio file sizes for various clip lengths

File	Video size (.mkv)	Audio size (.wav)
Full length	1.9 GB	481 MB
1 min	15.4 MB	3.7 MB
2 min	30.9 MB	7.4 MB
5 min	75.5 MB	19 MB
10 min	145.5 MB	37 MB
15 min	226.4 MB	55 MB
20 min	291 MB	74 MB

For the audio classification, object classification, and action recognition tasks, we use TensorFlow [122], a free and open-source machine learning library. We use Vosk [6] as the speech recognition toolkit for the speech-to-text task.

To study the processing times and energy, we used a 2:11:07 long video and process the video on both the mobile device and the GPU. We use TensorFlow on the GPU, and TensorFlow Lite (tflite) on the mobile device. Energy measurements are taken by profiling battery usage using Battery Historian [93], and are represented in terms of the percentage of battery consumed per process. The initial issue we face on the mobile device is that there is not enough memory to load the full video at once for processing. To overcome this issue, we crop the long video into smaller clips of equal sizes, and then process each clip individually. This is also beneficial from the practical standpoint when making the decision to offload those clips to the server for processing, as discussed later in the chapter. In Table 6.1, we show the original size of the video along with the sizes of each clip when we split it into clips whose duration is 1, 2, 5, 10, 15, and 20 minutes.

Audio Classification. Audio classification allows us to search for sounds that are closely associated with an action. Various audio processing tasks have been studied so far. Among them, the best known are audio classification [123], music tagging [5], and audio segmentation [124]. In order to process audio data, sound waves are sampled at specific time intervals and the amplitude is measured. For our specific application, we use audio classification, also known as *sound classification*, where each audio segment is labeled by the class or category to which it belongs, such as a siren or singing. This task involves the extraction of audio features, and then using those features as inputs to the classification model. In recent years, various CNNs have been built and trained to perform this task [123]. In order to train these CNNs, various datasets have been gathered, such as UrbanSound [20] and AudioSet-YouTube [125].

For *audio classification*, we use YAMNet [45], a deep net trained to classify 521 audio

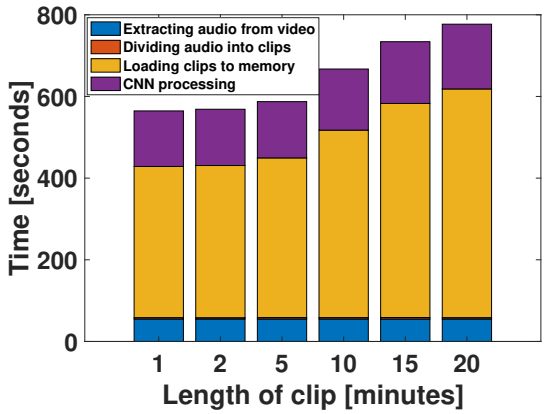


Figure 6.2. Mobile device processing times for full video divided in clips of various sizes: Audio classification.

classes. The classes are drawn from the AudioSet YouTube corpus [125]. According to [126], the model's accuracy is 79.16% for fixed size audio samples.

In Fig. 6.2 we show the processing times for running audio classification on the mobile device, using YAMNet, for the entire video when it is divided into clips of various lengths. The processing time is the sum of the time required to extract the audio component (wav) from the video, the time to divide the original video into mini-clips, the time to load the clips into memory for processing, and the time to process the clips using the CNN. In terms of energy, each 1-minute clip uses 0.011% of the battery on the mobile device.

When processing the video on the GPU, there is enough memory to process the full video in one shot. On the GPU it takes 61.6 seconds to perform the audio classification on the full video, including 33.25 seconds to extract the audio from the video, 0.6 seconds to load the audio into memory and 27.7 seconds to process the audio in the GPU. For a 1-minute clip, the GPU takes a total time of 1 second to load and classify the audio, as opposed to 4 seconds that it takes on the mobile device.

As can be seen from Fig. 6.2, dividing the videos into shorter segments, such as one- or two-minute clips provides noticeable improvements. While the processing times do not increase linearly with the length of the video, by reducing the size of the clips we can greatly reduce the amount of processing of subsequent stages or the amount of data to be uploaded, if most clips do not require further processing. For the remaining part of the chapter, unless stated otherwise, we show the results when videos are divided into 1-minute clips.

Speech-to-text. Speech-to-text allows us to search for certain spoken words that are

closely associated with an object. Also known as *speech recognition*, this is the process of inputting an audio waveform, performing speech-to-text transforms and getting a transcription of the spoken phrases [127]. However, there are some issues with speech-to-text, including the lack of accuracy due to the effect of background and environment noise, the complexity of reliable speech segmentation and different accents [128].

For *speech-to-text* we use *vosk-model-small-en-us-0.15*, a light weight model with a word error rate of 9.85% on LibriSpeech [6], a corpus of approximately $1,000$ hours of English speech. *Word to error rate* (WER) is a performance metric for automatic speech recognition (ASR) systems. It takes into consideration deletion, insertion and substitution of words by the ASR system.

Speech-to-text is the most time consuming process in the pipeline. It takes 15 seconds to process a 1-minute clip on the mobile device and has an energy cost of 0.038%. On the GPU, the processing time for a 1-minute clip is 11 seconds.

Object Classification. This is the process of classifying or labeling objects that appear within an image, based on a set of predefined objects used in the training phase. This allows us to search for an object that is closely associated with an action. Various models have been developed for this task that achieve high accuracy [12], [60].

For the *object classification* task, MobileNet [43], which is trained on the well-known ImageNet dataset [15], is used. The model can classify $1,000$ different objects, with an accuracy of 71%.

For the task of object classification, we extract frames at the rate of 1 frame per second (fps). If we choose to process the full video on the mobile device, the process is decomposed into two stages: extracting the frames from the video (which takes 18 minutes), and processing each frame using the MobileNet CNN (which takes 4 minutes). Therefore, the total time is 22 minutes. On the other hand, dividing the long video into smaller clips provides us with the ability to parallelize the process of frame extraction of a clip while processing the frames of another clip at the same time. The process of dividing the video into clips takes 39 seconds; the frame extraction along with CNN processing takes 14 minutes. Processing a 1-minute clip on the mobile device takes 6 seconds, and uses 0.028% of the device's battery. On the other hand, processing this clip on the GPU takes 2 seconds. We note that frame extraction in this case uses *ffmpeg-cpu* and is not expedited using the GPU.

Action Recognition. This is the process of identifying or classifying the action or activity occurring in a video clip [129]. Action recognition is more computationally complex than other vision algorithms (image classification or object detection). It also

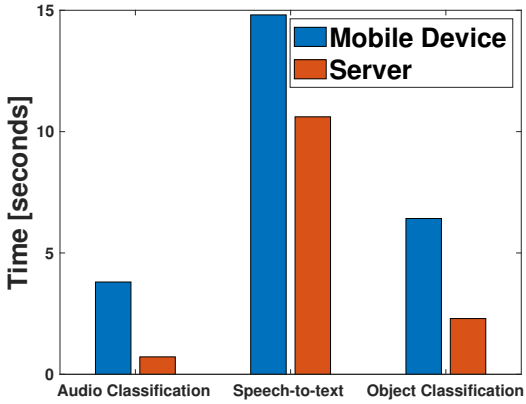


Figure 6.3. Mobile device and server processing times: 1-minute clip.

achieves lower accuracy. For example, image classification on ImageNet, which consists of $1,000$ classes (i.e., different objects), reaches an accuracy of 90.2% [130]. On the other hand, for a smaller action dataset with only 400 classes, such as Kinetics-400, the top-1 accuracy achieved is only 84.8% [131].

For *action recognition*, the Two-Stream Inflated 3D ConvNet (I3D) [44], trained on the Kinetics-400 dateset [56], is used. The CNN is able to recognize 400 different actions with an accuracy of 74.2%. This processing task can only be run on the GPU and not on mobile devices because 3D CNNs are not yet supported by *android-api* [121].

Processing the full video by dividing it into 1-minute clips and then processing each clip by using 5 second intervals for each action lasts 14 minutes in total, with an average processing time of 6 seconds per 1-minute clip.

Fig. 6.3 summarizes the processing times for the three different processing stages on both the server and the mobile device. Action recognition is excluded from the figure as it is only executed on the server. Energy measurements for each stage were shown to illustrate the suitability of implementing these algorithms on mobile devices. We see that stages with longer execution times use more energy. Energy will not be considered further and is not included in the optimization.

6.3.3 Query Types

In this section, we discuss various queries that might be issued by the user. We show results for processing on the server, as all the stages are able to run on the GPU. The goal of this section is to show relative processing times of the different stages and the effectiveness of using different markers on different types of videos and actions. Mainly,

we have two types of videos: long videos where the user is looking for short clips within the videos where the actions occur, or short videos in which the user is looking for a specific action. For each application, we describe the query and the different processing stages needed to respond to the query. In Section 6.4 we address the proper ordering of the stages and the location where each stage is processed, so that the best performance is achieved.

6.3.3.1 Long Videos Query

In this set of experiments, we have a number of long videos in which we search for a specific action. After the query is issued, the system returns the clips from within these videos where the action occurs. In the following, we consider two types of videos. The first are videos with matches of American football, whereas the second are ice hockey games.

American Football Games. In this query, the user is searching for all the field goals for a specific team that occur in a set of 3 NFL games. The total duration of all videos is 6 hours, 44 minutes and 34 seconds (about 404 minutes). We are searching for field goals scored by kicker 'Justin Tucker', a player of the 'Ravens'. The total number of field goals in all the videos is 11. However, they occur in 13 clips, as some field goals start at the end of a clip and finish at the following clip.

To respond to this query, we have two different stages: speech-to-text searching for the words 'Tucker', and action recognition searching for 'high kick'. We split the videos into clips of 1-minute each, with a total of 406 clips. Table 6.2 shows the confusion matrix for performing speech-to-text on all 406 clips versus performing action recognition on all the clips. Table 6.3 depicts the evaluation metrics for both stages individually.

We study the two options of alternating the order of the two stages and choose to set stage 1 as speech-to-text and stage 2 as action recognition. This ordering is chosen for two reasons. First, as mentioned previously, action recognition can only be done on the server. Therefore, starting with that stage would require the offload of the videos to the server, as opposed to offloading only the audio component. As shown in Table 6.1, the size of the video is 4 times the size of the audio component. Starting with speech-to-text drastically reduces the uploading time.

Second, assuming we do the processing for both stages on the server, the three videos require 78 minutes of processing if using action recognition followed by speech-to-text, and 71 minutes to process the videos using speech-to-text followed by action recognition. That decrease in processing time is due to the fact that speech-to-text discards more

Table 6.2. NFL: Confusion matrix for various stages

Stage	TP	FP	TN	FN
Speech-to-text	9	15	378	4
Action recognition	7	132	261	6
Speech then action recognition	5	7	386	8

Table 6.3. NFL: Evaluation metrics for various stages

Stage	Accuracy	Recall	Precision	F1 score
Speech-to-text	0.953	0.692	0.375	0.486
Action recognition	0.660	0.538	0.050	0.092
Speech then action recognition	0.963	0.384	0.416	0.4

negative clips than action recognition, as shown in Table 6.2. This is equivalent to saying that the second stage will have fewer videos to process. Note this is significantly shorter than the 406 minutes of the original video. Based on the aforementioned reasons, action recognition is ordered as the last stage. A formal algorithm for determining stage ordering and processing locations is provided in Section 6.4.

As shown in Table 6.2 and Table 6.3, speech-to-text outperforms action recognition in every metric. *However, adding the action recognition stage after the speech-to-text stage helps improve both accuracy and precision as it reduces the number of FPs.*

Hockey Games. For this query, we have videos of three full NHL games with size 9.76 GB and length of approximately 8 hours. The objective is to search for home team goals. In order to do so, we observe that two things occur when a goal is scored. A *horn sound* is signaled and the commentator says the word "scores". We are unable to use action recognition in this scenario because the only hockey-related action in the Kinetics-400 dataset is *playing ice hockey*, which would match the entirety of the video length. As mentioned in Section 6.3.2, background noise affects the accuracy of speech-to-text models, and given this specific application, we observe that the commentators speech cannot be transcribed while the horn is active. Therefore, speech-to-text cannot be used in this query.

As a consequence, to respond to this query, a single stage of audio classification is used, which is searching for a horn sound. We also notice that the horn is set at the end of the game and at the end of each of the other two periods[1], causing some false positive clips to be returned.

In the three videos of interest, there are a total of 18 home team goals. After running these videos through our system, we were able to correctly classify 14 of these clips,

[1]We remind the reader that each ice hockey match is split into three periods.

giving us a recall of 78%. We also have 7 false positive clips, which result in a precision of 67%. Accuracy is 98%, whereas the F1 score is 72%.

6.3.3.2 Short Videos Query

In this set of experiments, we have a set of 40 videos of various lengths averaging at around 2 minutes, with a total of 1 hour, 30 minutes and 16 seconds. The total size of the videos is 1.1 GB. The videos are drawn from the test set of the Kinetics-400 dataset. We have 10 videos of each of the following classes: playing basketball, skateboarding, playing the drums, and laughing. These specific classes are chosen because they have associated labels in both the action dataset (Kinetics-400) and the audio classification dataset (YAMNet). The videos are downloaded from Youtube and are chosen such that there is no music added to the background of the video. Therefore, the sounds can be detected by the audio CNN.

In the four cases, the queries are issued and associated with both a sound and an action. Therefore, the two stages that are used to respond to this query are audio classification and action recognition. In order to expedite the processing, we run the processing stage that takes the shortest time first to eliminate as many videos in the first stage as possible. In this case, audio classification is set to be stage 1 and action recognition is stage 2.

In this dataset, each activity contains very distinct set of voices, resulting in audio classification that returns a large number of TPs. Adding action recognition as a second stage does not add any TPs or FPs as it only processes the videos that have passed the first stage. We also notice that action recognition as a second stage reduces FPs without reducing TPs noticeably, albeit at the expense of processing time. Therefore, starting with audio classification is the better option, especially given the fact that action recognition can only be done on the server and requires an additional cost of all the videos being uploaded to the server.

Given the nature of videos, there are a number of different sounds that can be heard in videos, such as: the sounds of vehicles in the background if the video is shot on the street, the sound of people speaking if there is a gathering, and the sound of grunting or heavy breathing if it is a sports or exercise video. For this reason, we run our experiments by varying the top-k values for both stages from $k = 1$ to $k = 5$, and study the obtained results. We set the top-k value for the audio classifier to be 3. Note that, as shown in [68], this variable controls the number of false positives along with the number of true positives.

Table 6.4. Confusion matrix for each of the two stages

Audio Classification					
Query	number of videos	TP	FP	TN	FN
Basketball	40	6	1	29	4
Skateboarding	40	7	0	30	3
Drums	40	8	1	29	2
Laughing	40	10	0	30	0
Action Recognition after Audio Classification					
Query	positive videos	TP	FP	TN	FN
Basketball	7	6	0	30	4
Skateboarding	7	7	0	30	3
Drums	9	7	0	30	3
Laughing	10	10	0	30	0

Playing Basketball. This query is associated with two sounds from the YAMNet dataset: *basketball bounce* and *dribble*. The actions associated with this query from the Kinetics-400 dataset are: *dribbling basketball, dunking basketball, playing basketball*, and *shooting basketball*. The first stage of processing (audio) of the 40 videos correctly labels 6 clips as containing basketball playing along with 1 false positive. The false positive video is filtered out and classified as a negative after going through the second stage (action recognition). This illustrates the advantage of having two stages - the first and faster stage eliminated 35 videos from entering the second stage, and the second stage eliminated the false positive.

Skateboarding. This query is associated with the sound *skateboarding*, and the action *skateboarding*. In the first processing stage, which corresponds to detecting the sound of skateboarding, 7 out of 10 of the skateboarding videos were labeled positive out of the 40 videos tested in total. Those 7 videos move to the second stage and all are positively classified as containing the action *skateboarding*, using the i3d CNN action recognition model. By using the audio classifier as the first stage, we are able to eliminate the processing of 33 videos by the i3d CNN, which is slower than audio classification, as shown in Section 6.3, and requires the offloading of these videos to the server. In this case, the second stage did not improve performance.

Playing the drums. This query is associated with the following sounds from the YAMNet dataset: *drum kit, drum machine, drum, snare drum, rimshot, drum roll*, and *bass drum*. The action associated with this query is *playing drums*. In this scenario, 9 videos pass the first filter, with 1 false positive that is eliminated in the second stage.

Laughing. This query is associated with the following sounds from the YAMNet

Table 6.5. Evaluation metrics for the complete query

Query	accuracy	recall	precision	F1 score
Basketball	0.9	0.6	1	0.75
Skateboarding	0.925	0.7	1	0.8235
Drums	0.925	0.7	1	0.824
Laughing	1	1	1	1

dataset: *laughter, baby laughter, giggle, snicker, belly laugh,* and *chuckle/chortle.* The action associated with this query is *laughing.* As the sound of laughter seems to be very distinguishable from other sounds, this query yields a 100% accuracy after the first stage. After audio classification, all 10 laughter videos are labelled as positives, and similarly they all pass the second stage and are labeled as *laughing* after going through the action recognition CNN. No false positives occur in this case. Thus, nothing is gained by executing the second stage.

These examples clearly show the impact of having a first stage that is able to quickly and efficiently discard a large number of videos. We also see the impact of the second stage in discarding the false positive videos, despite the fact that in some cases the second stage did not improve performance. Table 6.4 illustrates the confusion matrix for each of the four queries in both stages, and Table 6.5 depicts the values of the four metrics of interest in this chapter. As can be seen from Table 6.5, the highest F1 score is achieved from the laughing videos, whereas the lowest is achieved with the basketball videos. We ran examples for additional two classes and observed similar results. Therefore, we chose to omit their results and the corresponding discussion.

6.4 VidQ

In this section, we present the system setup and the optimization formulation. First, we note that the different processing stages have different processing times, and different values of accuracy and recall, as shown in Section 6.3. As previously discussed, action recognition can only take place on the GPU. These general properties and constraints help us create the rules to make the decisions for using the heuristic to solve the problem.

In order to solve the problem, two main issues need to be addressed. First, the ordering of the processing stages must be set to achieve the recall or accuracy requirements in the most efficient way. Second, the location where the processing will take place (either locally on the mobile device, or on the server) must be determined to reduce the completion time of the task. The following guidelines are used to make these decisions:

1. Ordering of the stages:

    - If the requester is seeking a high accuracy or recall, the stage in which a higher accuracy or recall is achieved is the stage that is ordered first, because clips or videos excluded from the first stage cannot be retrieved in future stages.
    - If stages have similar performance in terms of recall or accuracy, or if having a high accuracy or recall is not the priority for the requester, the stage with shorter processing times, which may act as a filter, is placed first. That ordering results in discarding more videos or clips in the initial stages quicker, hence reducing the query overall response time.
    - If a stage can only be processed on the server, it will be ordered last, in order to give the initial stages the ability to filter out videos before offloading, unless the cost of offloading all the videos is very low.

2. Processing location:

    - If processing can only take place on a specific machine, then that stage takes place on that machine. For example, action recognition is only done on the server.
    - If the video is very large, breaking the video into mini clips before offloading aids in both distributing the processing of the clips, and offloading smaller videos.
    - If the bandwidth is very low, processing on the device is more appropriate because transmission times for videos can be very high.
    - If the bandwidth is very high, offloading videos to the server may be a better option because processing on the server is faster than processing on the mobile device.
    - The mobile device should make a calculated decision to either process the video/clip locally or offload to the server based on the network bandwidth, the number of videos or clips to be processed, the size of the video or clip, and the size of the queue on the server.
    - If the bandwidth is very low, and the stage can only be executed on the server, human verification may be faster than offloading the video.

6.4.1 Optimization Formulation

Having introduced the processing stages and quantifying the performance of these stages both in terms of processing time and the ability to detect actions, we proceed with modeling the system behavior and formulating the optimization problem, whose approximate solution will serve as the basis for the operation of our system (VidQ). Our goal is to minimize the query response time, given the stages that will be used and the order that they will be invoked to meet our accuracy goals. Before proceeding any further, Table 5.1 summarizes the notation used throughout the remainder of this chapter.

We assume that initially the queried videos are stored on the mobile device. Videos are captured using the same hardware and are all on the same scale. The device processes videos in series, but can transmit videos or clips of videos while it is processing. The stages that are going to be used and the ordering of stages in the pipeline are given. In the following, we use $\mathbb{I}$ to denote the set of all videos clips on the mobile device. Let $\mathbb{G}$ be the set of all processing stages that take place. The set of all the audio processing stages (speech-to-text and audio classification) is denoted by $\mathbb{A}$, whereas $\mathbb{V}$ is the set of all visual processing stages (object classification and action recognition). The following holds: $\mathbb{G} = \mathbb{A} \bigcup \mathbb{V}$. We set $c_{i,g}$ to be the audio or visual component of video i at stage g. If video i goes through three processing stages, and stages 1 and 3 use similar components, e.g., both stages have the audio component as input, then $c_{i,1} = c_{i,3}$.

The mobile device knows the transmission rate of the network for offloading r. The top-k_g values are known *a priori*, as k_g is set to meet the accuracy requirement for each processing stage $g \in \mathbb{G}$. The processing time for a component of video i in stage g when carried out on the mobile device is denoted by $\alpha_{i,g}$, and the processing time on the server is denoted as $\beta_{i,g}$. In general, the processing for each stage can take place on either the mobile device (captured by the decision variable $x_{i,g}$), or on the server (captured by the decision variable $y_{i,g}$). Note that in our heuristic, presented in the next section, we account for the fact that action recognition cannot take place on the mobile device. The parameters $x_{i,g}$ and $y_{i,g}$ are indicator variables whose values are either 1 or 0, i.e., $x_{i,g} = 1$ ($y_{i,g} = 1$) only if the processing takes place on the mobile device (server). Otherwise, it is 0.

Let q be the queried class. We define a binary variable for each stage, $z_i^{k_g}$. The parameter $z_i^{k_g}$ is 1 if q is in the top-k_g classes after the processing stage g is completed, and 0 otherwise.

We formulate the following optimization problem that corresponds to our system:

$$\text{minimize } T_{max}$$

$$\text{subject to:}$$

$$T_{max} \geq t_i^{comp}, \qquad \forall i \in \mathbb{I}, \tag{6.1}$$

$$x_{i,g} + y_{i,g} \leq 1, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.2}$$

$$\sum_{g \in \mathbb{G}} x_{i,g} + \sum_{g \in \mathbb{G}} y_{i,g} \geq 1, \qquad \forall i \in \mathbb{I}, \tag{6.3}$$

$$x_{i,g+1} - z_i^{k_g} x_{i,g} \leq 0, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.4}$$

$$y_{i,g+1} - z_i^{k_g} x_{i,g} \leq 0, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.5}$$

$$y_{i,g+1} - z_i^{k_g} y_{i,g} \leq 0, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.6}$$

$$x_{i,g} \in \{0,1\}, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.7}$$

$$y_{i,g} \in \{0,1\}, \qquad \forall i \in \mathbb{I}, \qquad \forall g \in \mathbb{G}, \tag{6.8}$$

where t_i^{comp} is the completion time of video i, and is a function of the decision variables. Let g' be the last processing stage that video i passes through before being dropped, i.e., is labeled as a miss. The parameters t_i^{comp} can be calculated as

$$t_i^{comp} = t_{i,g'}^{comp} = t_{i,g'}^{start} + \Delta_{i,g'}, \tag{6.9}$$

where $t_{i,g'}^{start}$ and $\Delta_{i,g'}$ are the start time for video i and the total processing time for processing video i, respectively. For the initial processing stage ($g = 1$), if the video component is offloaded, it instantly moves to the offload queue on the mobile device. Otherwise, it waits in the local queue to be processed locally on the mobile device. We have

$$t_{i,1}^{start} = \begin{cases} \sum_{j=i-1}^{0} x_{j,1}\alpha_{j,1}, & \text{if } x_{i,1} = 1 \\ 0, & \text{otherwise} \end{cases}$$

For consecutive stages we have

$$t_{i,g}^{start} = t_{i,g-1}^{comp}, \tag{6.10}$$

and

$$\Delta_{i,g} = \Delta_{i,g}^{m} + \Delta_{i,g}^{s}, \tag{6.11}$$

where $\Delta_{i,g}^{m}$ is the processing time of video i in stage g on the mobile device, and $\Delta_{i,g}^{s}$ is

the processing time of video i in stage g on the server. They are defined as:

$$\Delta^m_{i,g} = x_{i,g}(\tau^{loc}_{i,g} + \alpha_{i,g}), \tag{6.12}$$

and

$$\Delta^s_{i,g} = \begin{cases} y_{i,g}(\beta_{i,g} + \tau^{off}_{i,g} + \tau^{trans}_{i,g} + \tau^{server}_{i,g}), & \text{if } c_{i,g} \neq c_{i,j \in \mathbb{G} | j<g} \\ y_{i,g}\beta_{i,g}, & \text{otherwise} \end{cases} \tag{6.13}$$

where $\Delta^s_{i,g}$ ensures that an audio/visual component of a video is only offloaded to the server once.

The parameters $\tau^{loc}_{i,g}, \tau^{off}_{i,g}$, $\tau^{trans}_{i,g}$, and $\tau^{server}_{i,g}$ are the local queueing delay on the mobile device where video components wait for access to the local devices processing unit, the offload queueing delay where video components wait to be offloaded to the server for processing, the transmission delay, and the queueing delay on the server to access the GPU, respectively.

The local queueing delay is calculated based on the number of queued components to be locally processed and is calculated as

$$\tau^{loc}_{i,g} = \max\{0, t^{start}_{k,l} + \tau^{loc}_{k,l} + \alpha_{k,l} - t^{start}_{i,g}\}, \tag{6.14}$$

where (k, l) is the index of the last video component that is to be locally processed before $c_{i,g}$, i.e, $x_{k,l} = 1$, and is defined as

$$(k, l) = \max_{k,l}\{k \in \mathbb{I}, l \in \mathbb{G} | t^{start}_{k,l} > t^{start}_{i,g}, x_{k,l} = 1\}. \tag{6.15}$$

Let $B_{c_{i,g}}$ be the size of a visual or audio component that needs to be processed, and r be the transmission rate in the network. The transmission time is $\tau^{trans}_{i,g} = \frac{B_{c_{i,g}}}{r}$. The offload queueing delay can be calculated based on the estimated number of the previously offloaded components still remaining in the queue. Assuming the size of the components are known and the transmission rate is constant, the offload queueing delay at time $t^{start}_{i,g}$ is

$$\tau^{off}_{i,g} = \max\{0, t^{start}_{k,l} + \tau^{off}_{k,l} + \tau^{trans}_{k,l} - t^{start}_{i,g}\}, \tag{6.16}$$

where (k, l) is the index of the last video component that is to be offloaded before $c_{i,g}$,

i.e, $y_{k,l} = 1$, and is defined as

$$(k, l) = \max_{k,l}\{k \in \mathbb{I}, l \in \mathbb{G} | t^{start}_{k,l} > t^{start}_{i,g}, y_{k,l} = 1\}. \tag{6.17}$$

The queueing delay on the server after an image is transmitted is

$$\begin{aligned}\tau^{server}_{i,g} = \max\{0, t^{start}_{k} + \tau^{off}_{k,l} + \tau^{trans}_{k,l} + \tau^{server}_{k,l} \\ +\beta_{k,l} - (t^{start}_{i,g} + \tau^{off}_{i,g} + \tau^{trans}_{i,g})\},\end{aligned} \tag{6.18}$$

with the index (k, l) defined previously in Eq.(6.17).

We note that τ^{loc}, τ^{off}, and τ^{server} for the first video components entering the respective queues are 0, because all the queues are empty at the beginning.

As mentioned previously, the parameters $x_{i,g}$ and $y_{i,g}$ are the binary decision variables of the problem; $x_{i,g}$ is equal to 1 if the component of video i in stage g does processing on the mobile device and 0 otherwise, whereas $y_{i,g}$ is 1 if the component of video i does processing stage g on the server, and 0 otherwise. For example, if query q is issued searching for video clips where people are skateboarding: $\mathbb{G}$ = {audio classification (sound of skateboard), object detection (skateboards), action recognition (skateboarding)}, $\mathbb{A}$ ={audio classification (sound of skateboard)}, and $\mathbb{V}$ ={object detection (skateboards), action recognition (skateboarding)}.

Constraint (1) shows that the completion time of the system is greater or equal to the completion time of every video stored on the device. Constraint (2) ensures that only one from the ordered pair of decision variables $(x_{i,g}, y_{i,g})$ can have the value 1 at a time, i.e., a single mode of processing takes place on either the mobile device or the server, but not on both. Constraint (3) shows that at least one processing stage needs to take place for every clip. Constraints (4)-(6) show that a video can move to the second processing stage if and only if it passes the previous one and meets the top-k requirement. Constraints (7)-(8) capture the fact that the decision variables are binary.

Complexity: This optimization problem belongs to the class of integer programs, which are known to be NP-hard [132]. The problem structure does not allow for an algorithm with a performance guarantee. Hence, we resort to a heuristic.

6.5 VidQ Heuristic

In this section, we present the heuristic [2] that is used to approximate the solution of the optimization problem. Assuming that the stages are chosen appropriately, and ordered based on the aforementioned rules, the main focus in this section is on the distribution of the processing between the mobile device and server.

The videos are initially stored on the mobile device and are cropped into video clips of equal lengths. The size of each video clip, or audio portion of the video, is known. For a specific stage, let B_i be the size of a video clip or audio portion that needs to be processed, and r be the transmission rate in the network. The transmission time is $\tau_i^{trans} = \frac{B_i}{r}$. If the stage does a type of audio processing, only the audio component is transmitted. If the stage performs visual processing, the visual component is transmitted. In the case where action recognition occurs, video clips that are a hit from previous stages are directly offloaded and put in the head of the offload queue because action recognition can only be performed on the server.

The first step in the heuristic is to find the number of the components to be offloaded for the initial stage. Given that the system has a single user, and the fact that videos are cropped and have approximately the same sizes, the order of offloads is not critical, and we choose to offload the first clips/components in the queue.

We start by explaining the algorithm for the initial stage. This is followed by the description of the consecutive stages. For notational simplicity, we remove the subscript g from now on.

After pre-processing, the device estimates the completion time of the initial stage, when all the videos are processed on the mobile device, using the following expression:

$$T_d = \sum_{i=1}^{|\mathbb{I}|} \alpha, \tag{6.19}$$

Pre-processing is the necessary pre-processing like audio extraction, audio cropping, and video cropping. and the variable $|\mathbb{I}|$ denotes the number of videos in the system.

The completion time of the initial stage, if all the videos are offloaded and processed on the server, depends on the transmission time. If the transmission time is lower than the processing time on the server ($\tau_i^{trans} < \beta$), which happens in cases where the network

[2] We will be using interchangeably the notions 'heuristic' and 'VidQ' in the remainder of the chapter.

bandwidth is high, the completion time is estimated as

$$T_s = \tau_1^{trans} + \sum_{i=1}^{|\mathbb{I}|} \beta. \tag{6.20}$$

If the network bandwidth is low and the transmission time is larger than the processing time on the server ($\tau_i^{trans} \geq \beta$), the completion time is estimated as

$$T_s = \sum_{i=1}^{|\mathbb{I}|} \tau_i^{trans} + \beta. \tag{6.21}$$

It is worth pointing out that α and β are both average values as the user is not able to know these specific values before hand.

In order to minimize the completion time of the system, the goal is to distribute the video clips among those resources such that both machines achieve relatively equal completion times. Since initially all the video clips are stored on the mobile device, if no transmission or distribution happens in the system, the completion time will be calculated using Eq.(6.19). The cost of offloading a specific clip to the server can be estimated based on the transmission rate and the size. When a clip reaches the server, it is either processed directly on the GPU, or it incurs a queueing delay if the GPU is busy processing previously offloaded clips.

Alg. 7 shows the heuristic for distributing the video processing in the first stage. It returns the number of videos that should be offloaded to the server for processing. After the heuristic determines the number of offloads, the device starts processing clips that have been assigned to be locally processed, and the offload queue starts transmitting the clips that have been assigned for offload.

Complexity: For every video, there is a constant number of operations to decide if the video is to be offloaded or processed locally. The complexity of Alg. 7 is therefore $O(n)$, where n is the number of videos to be processed in that stage.

For consecutive stages, we have three cases. The *first case* is when the processing for the stage can only take place on the server, i.e., the second stage is action recognition. In that scenario, as soon as a component is labeled as a hit in the previous stage, the video component will be placed at the head of the offload queue and offloaded to the server for processing. Accordingly, a new estimation for the completion time on the server is calculated using Eq.(6.20). Based on this new estimation and the estimation of the completion time of the mobile device to process clips in its local queue, clips from the tail of the offload queue will be moved to the local queue to balance the completion

Algorithm 7: VidQ Heuristic- First Stage

output : the number of videos to be offloaded to the server

$n = |\mathbb{I}|$
$T_d = \sum_{i=1}^{n} \alpha$
$T_s = 0$
`// for the first video:`
$T_d = T_d - \alpha$
$T_s = \tau_1^{trans} + \beta$
if $T_d < T_s$ **then**
  return num=0;
end
`// for the remaining videos:`
for $i = 2$ *to* n **do**
  $T_d = T_d - \alpha$
  if $\tau_i^{trans} < \beta$ **then**
    $T_s = T_s + \beta$
  else
    $T_s = T_s + \tau_i^{trans}$
  end
  if $T_d < T_s$ **then**
    return num=i-1;
  end
end

times on both machines.

The *second case* corresponds to the scenario when a different component (audio or visual) is being processed in this stage compared to the previous stages. Alg. 7 captures the same procedure with the exception of setting n to be the number of videos that have passed the previous stage, i.e., the videos that are positively labeled. Based on that, the number of offloads is determined, and the system completes processing when this stage is completed or if there are any stages that follow, they would fall under one of the three cases and the decisions would be made accordingly.

The *third case* is when the same component (audio or visual) is being processed in this stage as any of the previous stages. Along with the device having access to all the clips, the server will also have a number of positively labeled clips that have been offloaded in previous stages. In this case, both the mobile device and server estimate their completion times if the locally stored positively "passed" videos from previous stages are processed in their respective locations. If the mobile device has a longer processing time, it will offload clips to the server, similarly to Alg. 7. However, if the server's completion

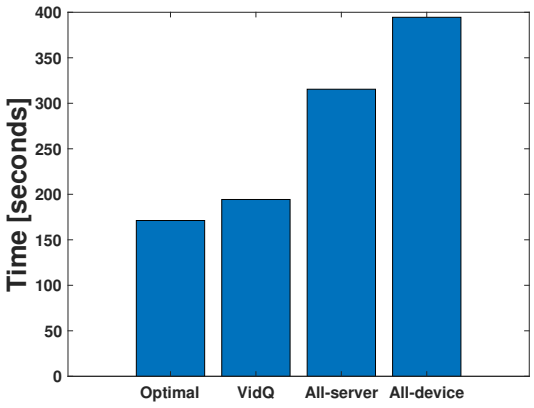


Figure 6.4. Query response time with different approaches.

time is longer, we start processing more videos on the mobile device, by calculating new estimates of T_s and T_d if clips are set to be processed on the device instead of the server. No components should be transmitted back to the mobile device as the original data is stored there. We only need to transmit the video id's.

6.6 Performance Evaluation

We start by comparing the performance of VidQ to the optimal solution. We build a simulator to run the comparison in a controlled environment. Following that, we use a full example from Section 6.3.3.1 to experimentally compare VidQ to three different baselines and a state-of-the-art algorithm.

6.6.1 Comparison to Optimal

Due to the complexity of the problem and its non-linearity we are unable to solve the problem using solvers such as Gurobi. We are also not able to model the system using optimization modeling languages such as Pyomo that use solvers like MindtPy to solve Mixed-Integer Nonlinear Programs (MINLP), mainly due to the presence of recursive equations Eqs.(6.14), (6.16), and (6.18), where different versions of the unknown variable are present on both sides of the equation.

To see how VidQ performs, we compare its results to the optimal solution of a system of 25 1-minute video clips, where the latter are obtained using brute-force search. In this example the system parameters are derived from NFL (American football) query presented in Section 6.3.3.1, where for stage 1 we have: the average values of $\alpha_1 = 14.5$

seconds, and $\beta_1 = 9.6$ seconds, and for stage 2 we have $\beta_2 = 7.6$ seconds, the transmission rate $r = 20$ Mbps, $|\mathbb{I}| = 25$, and the hit-ratio is 0.16 (i.e., 4 clips contain the action of interest)[3]. We exclude pre-processing as it takes the same time in both cases. The optimal solution completes processing in 171.2 seconds, while VidQ completes processing in 194.3 seconds. We also performed all of the processing on the mobile device, and all of the processing on the server, which resulted in completion times of 315.45 seconds and 394.5 seconds, respectively. Fig. 6.4 shows the completion times for all the cases, and we can clearly see that VidQ performs better than all-device and all-server as it utilizes the computational power on both machines.

VidQ heuristic uses average values for the processing times on the server (β) and mobile device (α) to estimate the completion time of the system and make instantaneous decisions based on that. It is not possible for the device to know in advance the exact processing times or which videos are hits (positives) and requires further processing in the second stage. Despite that, VidQ is only 13% slower than the optimal approach, as opposed to all-device and all-server approaches that perform more than 83% worse than the optimal.

For the first stage, the heuristic initially makes the decision to offload 10 clips to be processed locally on the device, and as the clips start being processed and some hits are moved to the offload queue to be processed on the server for the second stage, the decision changes to process 11 video clips. The optimal solution has 12 clips processed locally, and 13 clips processed on the server. As for the second stage, all the clips that pass the first stage are offloaded to the server for processing because action recognition is only done on the server.

6.6.2 Experimental Evaluations

In this section, we show how VidQ performs on the NFL (American football) query presented in Section 6.3.3.1. The query is searching through 6 hours, 44 minutes and 33 seconds of video footage for the field goals that occur in a group of NFL matches. We compare VidQ to the performance that is achieved when using an action recognition CNN [4], All-server, and All-device baselines:

- *i3d [4]:* The videos are processed using the open source action recognition CNN i3d with no additional stages or pipelining, or pre-processing.

[3]Hit-ratio is defined as the ratio of clips containing the action of interest.

- *All-server:* The videos are pre-processed on the mobile device and offloaded to the GPU where all the processing stages take place.
- *All-device:* The processing is performed entirely on the mobile device, and action recognition cannot take place.
- *VidQ:* It can process any number of stages that give us the best accuracy. All the stages can run on either the mobile device or server, except action recognition that needs to be executed on the server.

There are two stages when responding to this query: speech-to-text and action recognition. The ordering of the stages is performed in line with the stage ordering rules mentioned at the beginning of this section. For the remainder of this section, we assume that $r = 20$ Mbps, and the videos are cropped to 1-minute clips.

i3d [4]. In this benchmark, given that the goal of the chapter is to do action recognition on stored videos, we use one of the open source action recognition CNN models with no additional pipe-lining or stages. As mentioned previously, since mobile devices are not capable of executing the models required to perform action recognition, the videos are initially offloaded to the server.

Along with the processing time, there is a transmission cost of offloading the 7 GB of data to the server when the transmission rate is 20 Mbps. However, the server can start processing as soon as the first video is fully offloaded. Therefore, we start transmitting the video with the smallest size first. In this case, the smallest video is 1.3 GB, and takes 8 minutes and 36 seconds to be transmitted. Following the offload of the videos to the server, the videos are cropped into 1-minute clips for processing, which takes 1 minute and 52 seconds. Finally, processing all the clips using i3d takes 51 minutes and 30 seconds.

The total completion time of *i3d* is 1 hour, 1 minutes and 58 seconds.

All-server. In this case, we assume the mobile device does pre-processing by extracting the audio from the videos and dividing them into 1-minute audio clips, before sending them to the server. This pre-processing leads to enabling the server to start processing as soon as the first audio clip arrives. Following that, the device also crops the full videos into 1-minute video clips in preparation to upload the positive clips to the server for stage 2.

Stage 1: The pre-processing for the first video takes 1 minute, and the transmission time for the first audio clip, which is 3.66 MB, takes 1.5 seconds. Therefore, the server can start processing after 1 minute and 1.5 seconds. The server then takes 1 hour, 4

minutes and 58 seconds to process all the audio clips. The total completion time of the first stage is 1 hour, 5 minutes and 59 seconds.

Stage 2: As soon as a clip is labeled a hit, the mobile device uploads that video clip to the server to be processed. In this scenario, we have 24 clips that are positively classified in stage 1. The time to do action recognition for the 24 clips is 3 minutes and 9 seconds.

The total completion time of *All-on-server* case, after adding the completion time of all stages, is 1 hour, 9 minutes and 8 seconds.

All-device. This case corresponds to the scenario when the mobile device has no access to a server to aid in processing, i.e., the entire process will run on the mobile device. In that case, the process undergoes the following phases:

Stage 1: In the first stage, the processing time will be the time for extracting audio from the original videos, the time to crop the videos to 1-minute audio clips, and the processing time of running speech-to-text on all the 406 1-minute clips. Extracting the audio from the three videos takes 2 minutes and 35 seconds, dividing the audio into 1-minute clips takes 13 seconds, and finally running speech-to-text on all the clips takes 1 hour, 38 minutes and 15 seconds. The completion time of stage 1 on the mobile device is 1 hour, 41 minutes and 3 seconds.

Stage 2: There are 24 positive videos that pass stage 1. Since offloading is not an option in this case, and action recognition can not run on the device, the user will have to manually confirm if the videos contain the action of interest. In this specific example, the user would need to look at 8 minutes of videos that are false positives.

The completion time of *All-on-device*, including the human verification in stage 2, is 1 hour, 49 minutes and 3 seconds.

VidQ. Assuming the same transmission rate of 20 Mbps, for stage 1 we have $\alpha_1 = 14.5$ seconds and $\beta_1 = 9.6$ seconds, whereas for stage 2 we have $\beta_2 = 7.6$ seconds.

Initially, the mobile device extracts the audio from the videos, and crops them into 1-minute audio clips. It takes 2 minutes and 48 seconds to complete this. Following that, Alg. 7 is run to determine the number of clips that will be offloaded and processed on the GPU. After that decision is made, the mobile device also crops the long videos into 1-minute video clips for the visual component of the clips that are a hit and pass stage 1 to be offloaded to the server for the processing of stage 2. This time is counted towards the completion time of the device, as the audio clips can be offloaded to the server and can start processing while the mobile device is cropping the visual component of the video.

By running Alg. 7, the heuristic calculates that the number of clips to be offloaded

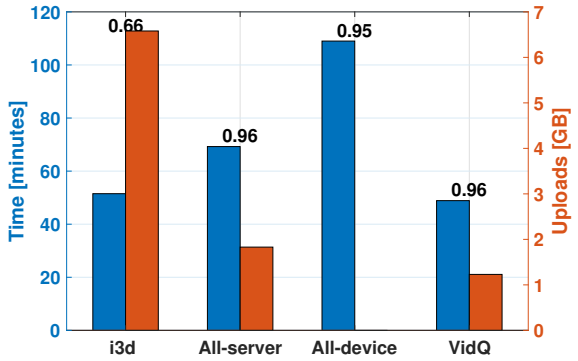


Figure 6.5. NFL example: query response time, amount of uploads and accuracy.

to the server for stage 1 is 244. As both the server and mobile device start processing their assigned clips, various hits will appear, indicating that these clips will move to stage 2 for further processing. Since stage 2 is action recognition, and can only be done on the server, these video clips will be directly offloaded to the server and will be put at the head of the offload queue. This will trigger the re-evaluation of the estimated completion times of both the server and the mobile device, and the heuristic will update the decisions for the future clips that have not been processed yet. At the end, the heuristic will process 163 clips on the mobile device for stage 1, 243 clips on the server for stage 1, and 24 clips on the server for stage 2.

The completion time of the mobile device is 38 minutes and 24 seconds, while the completion time of the server is 46 minutes and 5 seconds. Finally, the total query response time for VidQ is 48 minutes and 53 seconds.

Fig. 6.5 summarizes the total completion time for VidQ, and the three benchmark models, along with the amount of bandwidth each algorithm uses for uploads. The accuracy of each algorithm is shown as a number on top of its bars. As can be observed from Fig. 6.5, all three systems that use markers to help locate the action in the videos greatly reduce the bandwidth required for uploads vs. the i3d approach, and all three achieve much higher accuracy. Our approach leads to a performance improvement of more than 50% compared to *all-device* in terms of completion time. While the completion time of i3d is very close to the result from VidQ, it is clear that the amount of bandwidth needed is very high, and given that speech-to-text is not used, the number of FPs remains high, leading to a low accuracy.

In the scenario where the bandwidth is high (such as 1 Gbps for instance), there would be no big added benefit in increasing uploads because the largest delay component would be that at the GPU queue, given that the processing time for speech-to-text and

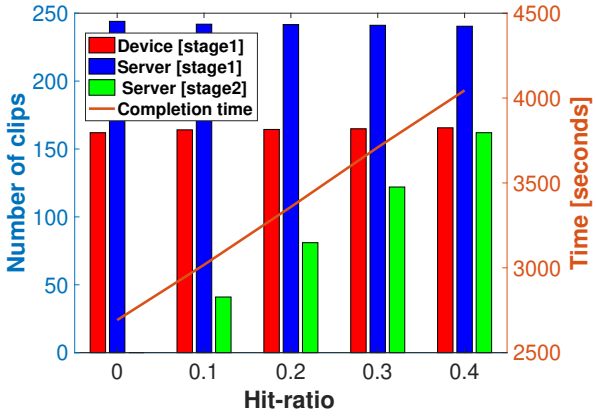


Figure 6.6. Varying hit-ratio.

action recognition is relatively high compared to other stages, such as audio classification. In the case of audio classification where the server processing time is low, having a higher bandwidth would be more advantageous.

6.6.3 Sensitivity Analysis

In this section, we show the results when varying the different system parameters. Unless stated otherwise, the system parameters are as follows: the number of clips to be searched $|\mathbb{I}| = 406$, the number of clips containing the action of interest is 25 clips (meaning the hit-ratio is 0.06157), the average values of $\alpha_1 = 14.5$ seconds, $\beta_1 = 9.6$ seconds, for stage 2 we have $\beta_2 = 7.6$ seconds, and the transmission rate $r = 20$ Mbps.

Varying hit-ratio. As our system is geared towards applications where the user searches a large number of videos for specific actions, the hit-ratio in such scenarios is low. From Fig. 6.6 it is clear that as the number of hits increases, the query response time increases as well, as the number of videos that pass stage 1 and need processing in stage 2 increases. The main delay in the completion in this scenario is that the second stage, which is action recognition, can only run on the server. The completion time of the server becomes higher as the hit-ratios increase.

Varying server processing time (β). The effect of varying the processing time of the server (β_1 and β_2) is shown in Fig. 6.7. In real applications, the reduction in processing time can be achieved by either increasing the number of GPUs on the server, or replacing the CNN with a faster CNN when new algorithms are developed. As the processing time of the server decreases, more clips are offloaded to be processed on the server and the completion time of the system reduces. Note that the number of clips

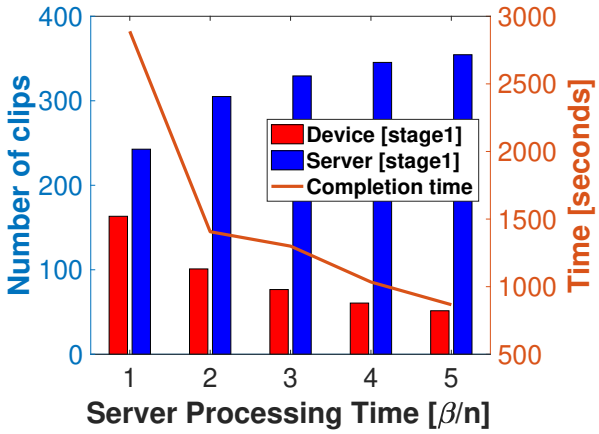


Figure 6.7. Varying server processing time (β).

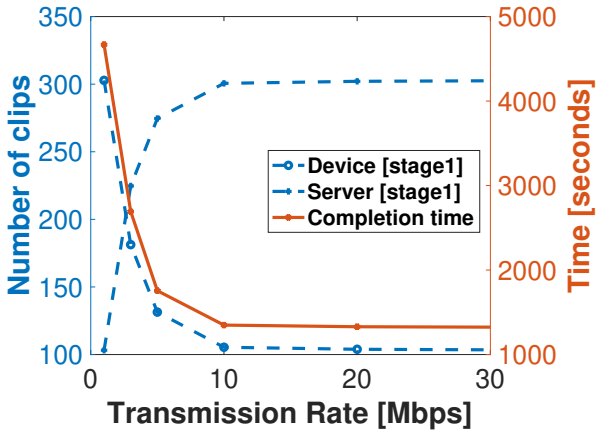


Figure 6.8. Varying transmission rate (r).

processed in stage 2 is excluded from the figure because it is equal in all cases, as the hit-ratio is constant.

Varying transmission rate. As the transmission rate increases, the number of clips offloaded to the server increases and the completion time decreases, as processing speeds on the server are faster than on the device. At some point, the processing speed of the GPU is not high enough and the server becomes the bottleneck. Therefore, the number of offloads remains constant and the curve flattens. Fig. 6.8 shows the results when $\beta_1 = 4.8$ seconds, and $\beta_2 = 3.8$ seconds (twice as fast compared to the scenario from the beginning of Section 6.3).

6.7 Related Work

With the increased use of mobile devices and the advancement of edge computing, deep learning for mobile devices has become an active area of research. Even with the advances in deep learning optimization on mobile devices [133], [134], these gadgets are behind performance-wise compared to more powerful GPUs. Offloading has been one of the solutions for distributing the load and enabling advanced processing. Mobile cloud computing, where mobile devices with limited computational power utilize offloading to more powerful servers to improve performance, is an active area of research. Systems like [68], [67], [10] utilize cloud offloading for video and image querying applications.

In [68], the system is designed to collect positively classified images in a network of mobile devices. The objective is to minimize the completion time. The goal is achieved by dividing the processing pipeline into two stages, a fast less accurate CNN that aims to exclude negative images, and a second slower more accurate CNN to complete the processing of images that have a higher probability of being positives. In [67], the system deals with processing videos with the goal of minimizing processing times. The system consists of multiple mobile devices having access to multiple video clouds. The goal is achieved by relocating the videos to the video clouds and choosing the appropriate cloud to offload the video for processing. In [10], the mobile device chooses to either do video offload or frame offload to a server where video processing takes place. Similar to these systems, we use video offload and aim to minimize query response time. However, our focus is on searching for actions of interest from videos instead of simple objects from extracted frames, making it a more complex and challenging problem. We also utilize both visual and audio features of the videos.

Recent advances in video processing enable more accurate action recognition. Many action recognition models apply 3D or 2D convolutions on spatio-temporal data [129], [135], making them computationally intensive and hard to implement on edge devices. In [129], the authors modify 3D CNNs to use what they call Bidirectional Encoder Representations from Transformers (BERT) layer to improve accuracy. In [135], the authors utilize both RGB and optical flow for action recognition. They train 3D CNNs with a different loss function that enables them to obtain flow features without explicit optical flow computation at inference time.

Research in the field of audio processing, such as audio classification [123] and automatic speech recognition [136], has also advanced considerably. Specifically, the use of audio for video retrieval is studied in works like [137], and [138]. In [137], the

authors present a framework for event classification using the audio stream as input. The framework handles extracting semantic information in real-world videos. In [138], the authors propose an end-to-end audio event detection CNN. The CNN enables the use of the extracted features for different audio and video analytic tasks.

As mentioned previously, as research in the field of audio and video processing progresses and new models with shorter processing times and ability to be deployed on mobile devices are introduced, these models can be used in the various stages of the pipeline in VidQ.

Multi-modal models using both visual and audio features have been leveraged for increasing the efficiency of action recognition. In [139], the authors model visual and audio components of a video jointly as a fused multi-sensory representation. In [140], audio is used to select frames for action recognition in long untrimmed videos. The authors use image-audio pairs to find the key moments, where action recognition is performed. Similar to these papers, in our work, we use both audio and visual processing for video retrieval. However, instead of using complex computationally intensive models, we divide the task into multiple simpler stages that can be executed on both the mobile device and the server. We study the division and ordering of these stages, the models used in each stage and resource allocation to minimize the query response time.

A more detailed review of audio classification, speech to text, image processing and action detection and the relevant models and algorithms is presented in Section 6.3.2.

6.8 Conclusion

This chapter presents VidQ, a system for retrieving videos with specific actions from a set of locally stored videos on a mobile device. We solve the problem by using both audio and visual processing. We divide the task into various stages to enable efficient resource allocation and processing distribution. We formulate the optimization problem and design a heuristic algorithm whose objective is to minimize the query response time. By implementing multimodal searching, and dividing the processing into stages, we were able to reduce the query response time and enable the distribution of the processing on both the mobile device and server GPU. Finally, we show that VidQ outperforms other state-of-the-art approaches.

As part of our future work, we plan to build a system where the query is issued to multiple devices, where the objective would be to minimize the completion time over all devices in the system constraining the maximum amount of energy a device is willing to

spend in the process.

Chapter 7
Conclusion and Future Work

In this dissertation we presented solutions and algorithms for various types of image and video information queries in mobile networks. The networks we work with consist of mobile devices that have both energy and computation limitations, and servers that have more powerful GPUs. Our work focuses on networks with limited bandwidth in the uplink and downlink. In our work we utilized various computer vision and audio processing algorithms and techniques. We deployed the models on both the mobile devices and servers and focused on distributing the processing in the network nodes to optimize for query response time, energy and network bandwidth.

In Chapter 3 we presented the data collection problem where the requester broadcasts a query, searching for an object of interest, to multiple mobile devices in the system. We optimized for query response time, where the goal was to collect all the images that are positively classified as containing the object of interest to the centralized server. Initially, all the images are stored locally on the mobile devices. For each image, the device makes the decision to offload the image for processing on the server or locally processing it on the device based on the network conditions, hit-rate and cloud-backlog. If the device makes the decision that an image is processed locally on the device it goes through two processing stages (Filtering and Selection). The Filtering stage uses a smaller less accurate CNN and aims to discard as much negatives as quickly as possible. After the Filtering stage, the mobile device has an option to change the decision to offload the image to the server for processing or to continue processing on the device. This decision is made using updated information, about the server and network, that the device receives. The Selection stage uses a more accurate CNN, similar to the one stored on the server. Since the problem is NP-hard we proposed a heuristic and showed that our heuristic outperforms other algorithms and systems.

In Chapter 4 we extended the work in Chapter 4 to include an energy constraint.

Each device has a specific percentage of it's battery that is allocated to respond to the query. We proposed a heuristic and evaluated it on both LTE and WiFi and compared it to the time-heuristic in Chapter 3. In our current work we assume prior knowledge of the hit rate/ratio. Future work can include finding more sophisticated methods for accurately estimating and updating hit-rates.

In Chapter 5 we addressed a different query where the request is to search for novel objects that the deployed CNNs are not trained to search for. In this scenario, we face different challenges such as the availability of a limited number of samples of the novel queried class making it difficult to train new models. Another challenge is that deploying new models in a bandwidth constrained network is not feasible. To solve these issues we proposed a distance-based classifier utilizing the pre-loaded models in the system nodes. Our algorithm enables tuning system parameters to adapt to different network conditions. We compared our algorithm to other approaches and showed that we save in the amount of data transfer in both the uplink and downlink. Human in the loop, or classifier feedback can be explored as a method to increase accuracy of novel detection. Other methods such as generating new images and using Generative Adversarial Networks (GANs) can be studied for training and searching for novel classes. In this work we searched a bounded number of images that are stored on mobile devices, while utilizing pre-trained CNNs. Future work would look at scaling the system to include a large amount of both classes and images to be searched. Techniques such as inverted indexing can be used to solve such problems. Inverted indexing [141], initially introduced for searching documents, provides a mapping between words and their locations in text documents. Applications for these data structures have extended to include image searches [142], [143] and [144].

Finally, Chapter 6 solves a more complex problem where the query is for an action occurring in a video, as opposed to an object in an image. The mobile device where the videos are stored issues the query and has access to a server with GPUs. The goal is to optimize for query response time. In order to respond to such queries we designed a multi-stage pipeline that uses both the visual and audio components of the video. The stages are chosen and ordered in a manner that enables efficient processing distribution, and expedites the query response time, while achieving a decent recall or chosen metric. We formulated the problem and developed a heuristic to solve this NP-hard problem. We showed through experiments and simulations that our system outperforms different baselines and algorithms.

As new technologies emerge and mobile devices are able to run more complex CNNs, such as 3D models, our algorithms can be revisited. In the future, the systems can

consider user mobility, where the transmission rate varies during the query response time, and algorithms would adapt to the dynamic network changes. Training customized vision and audio models for object and action recognition with the system designs and stages as well as the resource allocation process in mind can be a future direction for research.

Bibliography

[1] Gu, T. "Newzoo's Global Mobile Market Report: Insights into the World's 3.2 Billion Smartphone Users, the Devices They Use the Mobile Games They Play," *newzoo*.
URL https://newzoo.com/insights/articles/newzoos-global-mobile-market-report-insights-into-the-worlds-3-2-billion-smartphone-users-the-devices-they-use-the-mobile-games-they-play/

[2] Cosgrove, E. "One billion surveillance cameras will be watching around the world in 2021, a new study says," *CNBC*.
URL https://www.cnbc.com/2019/12/06/one-billion-surveillance-cameras-will-be-watching-globally-in-2021.html

[3] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012) "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, **25**.

[4] Carreira, J. and A. Zisserman (2017) "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308.

[5] Choi, K., G. Fazekas, M. Sandler, and K. Cho (2018) "A comparison of audio signal preprocessing methods for deep neural networks on music tagging," in *2018 26th European Signal Processing Conference (EUSIPCO)*, IEEE, pp. 1870–1874.

[6] Shmyrev, N. (2021), "vosk-api," https://github.com/alphacep/vosk-api.

[7] Goodfellow, I., Y. Bengio, A. Courville, and Y. Bengio (2016) *Deep learning*, MIT press Cambridge.

[8] Shorten, C. and T. M. Khoshgoftaar (2019) "A survey on image data augmentation for deep learning," *Journal of Big Data*, **6**(1), p. 60.

[9] O'Mahony, N., S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh (2019) "Deep learning vs. traditional computer vision," in *Science and Information Conference*, Springer, pp. 128–144.

[10] Lu, Z., K. S. Chan, and T. La Porta (2018) "A computing platform for video crowdprocessing using deep learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, pp. 1430–1438.

[11] Zhang, X., Z. Yang, Y. Liu, and S. Tang (2019) "On reliable task assignment for spatial crowdsourcing," *IEEE Transactions on Emerging Topics in Computing*, **7**(1).

[12] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015) "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

[13] Krizhevsky, A., V. Nair, and G. Hinton (2009) "Cifar-10 and cifar-100 datasets," *URl: https://www. cs. toronto. edu/kriz/cifar. html*, **6**.

[14] Hussain, M., J. J. Bird, and D. R. Faria (2018) "A study on cnn transfer learning for image classification," in *UK Workshop on Computational Intelligence*, Springer, pp. 191–202.

[15] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009) "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, pp. 248–255.

[16] Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014) "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, pp. 740–755.

[17] Everingham, M., L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman (2010) "The pascal visual object classes (voc) challenge," *International journal of computer vision*, **88**(2), pp. 303–338.

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

[19] Soomro, K., A. R. Zamir, and M. Shah (2012) "UCF101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*.

[20] Salamon, J., C. Jacoby, and J. P. Bello (2014) "A Dataset and Taxonomy for Urban Sound Research," in *Proceedings of the 22nd ACM international conference on Multimedia*, Orlando, FL, USA.

[21] Pytorch (2019), "TORCHVISION.MODELS," `https://pytorch.org/docs/stable/torchvision/models.html`.

[22] Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017) "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*.

[23] Iandola, F. N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer (2016) "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," *arXiv preprint arXiv:1602.07360*.

[24] Bianco, S., R. Cadene, L. Celona, and P. Napoletano (2018) "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, **6**, pp. 64270–64277.

[25] Lu, Z., K. S. Chan, R. Urgaonkar, and T. La Porta (2016) "On-demand video processing in wireless networks," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, IEEE, pp. 1–10.

[26] Lee, K., K. Lee, H. Lee, and J. Shin (2018) "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Advances in Neural Information Processing Systems*.

[27] Deng, L. and Y. Dong (2014) "Foundations and trends® in signal processing," *Signal Processing*, **7**, p. 200.

[28] Lowe, D. G. (1999) "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, Ieee, pp. 1150–1157.

[29] Goldenshluger, A., A. Zeevi, et al. (2004) "The Hough transform estimator," *The Annals of Statistics*, **32**(5), pp. 1908–1932.

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

[31] Altman, N. S. (1992) "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, **46**(3), pp. 175–185.

[32] LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998) "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, **86**(11), pp. 2278–2324.

[33] Karpathy, A. et al. (2016) "Cs231n convolutional neural networks for visual recognition," *Neural networks*, **1**(1).

[34] LeCun, Y. and M. Ranzato (2013) "Deep learning tutorial," in *Tutorials in International Conference on Machine Learning (ICML'13)*, Citeseer, pp. 1–29.

[35] Tran, D., L. Bourdev, R. Fergus, L. Torresani, and M. Paluri (2015) "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497.

[36] Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014) "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*.

[37] Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. (2016) "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*.

[38] Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019) "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, pp. 8026–8037.

[39] Wang, Z., K. Liu, J. Li, Y. Zhu, and Y. Zhang (2019) "Various frameworks and libraries of machine learning and deep learning: A survey," *Archives of computational methods in engineering*, pp. 1–24.

[40] Long, J., E. Shelhamer, and T. Darrell (2015) "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

[41] Girshick, R. (2015) "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

[42] Wang, L., Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool (2016) "Temporal segment networks: Towards good practices for deep action recognition," in *European conference on computer vision*, Springer, pp. 20–36.

[43] Chollet, F. (2021), "tensorflow/../mobilenet.py," `https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/applications/mobilenet.py`.

[44] Las Casas, D. (2019), "kinetics-i3d," `https://github.com/deepmind/kinetics-i3d`.

[45] Plakal, M. and D. Ellis (2020), "YAMNet," `https://github.com/tensorflow/hub/blob/master/examples/colab/yamnet.ipynb`.

[46] "Tensorflow Lite, howpublished = `https://www.tensorflow.org/lite/`„" .

[47] Subramanian, V. (2018) *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*, Packt Publishing Ltd.

[48] Xie, S., A. Kirillov, R. Girshick, and K. He (2019) "Exploring randomly wired neural networks for image recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1284–1293.

[49] Yun, S., D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo (2019) "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6023–6032.

[50] He, K., G. Gkioxari, P. Dollár, and R. Girshick (2017) "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

[51] Zhang, H., K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal (2018) "Context encoding for semantic segmentation," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7151–7160.

[52] Wu, C.-Y., R. Girshick, K. He, C. Feichtenhofer, and P. Krahenbuhl (2020) "A Multigrid Method for Efficiently Training Video Models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 153–162.

[53] Tran, D., H. Wang, L. Torresani, and M. Feiszli (2019) "Video classification with channel-separated convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5552–5561.

[54] Everingham, M. and J. Winn (2011) "The pascal visual object classes challenge 2012 (voc2012) development kit," *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, **8**.

[55] Gauen, K., R. Dailey, J. Laiman, Y. Zi, N. Asokan, Y.-H. Lu, G. K. Thiruvathukal, M.-L. Shyu, and S.-C. Chen (2017) "Comparison of visual datasets for machine learning," in *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, IEEE, pp. 346–355.

[56] Kay, W., J. Carreira, K. Simonyan, B. Zhang, C. Hillier, et al. (2017) "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*.

[57] shelhamer (2017), "BVLC AlexNet Model," `https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet`.

[58] shelhamer (2017), "BVLC GoogleNet Model," `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`.

[59] forresti (2016), "SqueezeNet," `https:/github.com/forresti/SqueezeNet/tree/master/SqueezeNet_v1.1`.

[60] He, K., X. Zhang, S. Ren, and J. Sun (2016) "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

[61] KaimingHe (2016), "deep-residual-networks," `https://github.com/KaimingHe/deep-residual-networks#models`.

[62] Guo, B., Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou (2015) "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM computing surveys (CSUR)*, **48**(1), pp. 1–31.

[63] Zhang, T., A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee (2015) "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438.

[64] Zhang, X., X. Zhou, M. Lin, and J. Sun (2018) "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856.

[65] Othman, M., S. A. Madani, S. U. Khan, et al. (2013) "A survey of mobile cloud computing application models," *IEEE communications surveys & tutorials*, **16**(1), pp. 393–413.

[66] Dinh, T. Q., J. Tang, Q. D. La, and T. Q. Quek (2017) "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, **65**(8), pp. 3571–3584.

[67] Lu, Z., K. Chan, R. Urgaonkar, S. Pu, and T. La Porta (2019) "NetVision: On-Demand Video Processing in Wireless Networks," *IEEE/ACM Transactions on Networking*.

[68] Felemban, N., F. Mehmeti, H. Khamfroush, Z. Lu, S. Rallapalli, K. S. Chan, and T. La Porta (2019) "PicSys: Energy-Efficient Fast Image Search on Distributed Mobile Networks," *IEEE Transactions on Mobile Computing*.

[69] Ganti, R., F. Ye, and H. Lei (2011) "Mobile crowdsensing: Current state and future challenges," *IEEE Communications Magazine*, **49**(11).

[70] Yang, D., G. Xue, X. Fang, and J. Tiang (2016) "Incentive mechanisms for crowdsensing: Crowdsourcing with smartphones," *IEEE/ACM Transactions on Networking*, **23**(3).

[71] Simonyan, K. and A. Zisserman (2014) "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*.

[72] sh1r0, "caffe-android-lib," `https://github.com/sh1r0/caffe-android-lib`.

[73] Zoo, C. M. "Available online: https://github. com/BVLC/caffe/wiki," .

[74] Bellard, F., M. Niedermayer, et al. "FFmpeg," *http://ffmpeg. org*.

[75] Ignatov, A., R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool (2018) "Ai benchmark: Running deep neural networks on android smartphones," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 0–0.

[76] Lu, Z., S. Rallapalli, K. Chan, and T. La Porta (2017) "Modeling the resource requirements of convolutional neural networks on mobile devices," in *Proceedings of the 25th ACM international conference on Multimedia*, pp. 1663–1671.

[77] Arisha, A., P. Young, and M. El Baradie (2001) "Job shop scheduling problem: An overview," .

[78] Samsung, "Galaxy S4," .
URL `https://www.samsung.com/uk/smartphones/galaxy-s4-i9505/GT-I9505ZKABTU/`

[79] Samsung, "Galaxy S9," .
URL `https://www.samsung.com/uk/smartphones/galaxy-s9/specs/`

[80] Kao, Y.-H., B. Krishnamachari, M.-R. Ra, and F. Bai (2017) "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, **16**(11), pp. 3056–3069.

[81] Chen, T. Y.-H., L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan (2015) "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pp. 155–168.

[82] Hua, Y., W. He, X. Liu, and D. Feng (2015) "SmartEye: Real-time and efficient cloud image sharing for disaster environments," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, pp. 1616–1624.

[83] Kang, Y., J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang (2017) "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, **45**(1), pp. 615–629.

[84] Li, M., J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng (2018) "CrowdBC: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, **30**(6), pp. 1251–1266.

[85] Wang, J., M. Li, Y. He, H. Li, K. Xiao, and C. Wang (2018) "A blockchain based privacy-preserving incentive mechanism in crowdsensing applications," *IEEE Access*, **6**, pp. 17545–17556.

[86] Teerapittayanon, S., B. McDanel, and H.-T. Kung (2016) "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, pp. 2464–2469.

[87] Wu, Z., T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris (2018) "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826.

[88] Yang, T.-J., Y.-H. Chen, and V. Sze (2017) "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695.

[89] Ra, M.-R., A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan (2011) "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 43–56.

[90] Chun, B.-G., S. Ihm, P. Maniatis, M. Naik, and A. Patti (2011) "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314.

[91] Cuervo, E., A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl (2010) "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62.

[92] Kosta, S., A. Aucinas, P. Hui, R. Mortier, and X. Zhang (2012) "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom*, IEEE, pp. 945–953.

[93] Google, B. H. "Available online:https://github.com/google/battery-historian," .

[94] Felemban, N., F. Mehmeti, T. F. La Porta, and H. Kwon (2021) "EDIR: Efficient Distributed Image Retrieval of Novel Objects in Mobile Networks," in *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, IEEE, pp. 392–400.

[95] Suri, N., E. Benvegnù, M. Tortonesi, C. Stefanelli, J. Kovach, and J. Hanna (2009) "Communications middleware for tactical environments: Observations, experiences, and lessons learned," *IEEE Communications Magazine*, **47**(10), pp. 56–63.

[96] Zhuang, F., Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He (2021) "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, **109**(1).

[97] Qiao, S., C. Liu, W. Shen, and A. L. Yuille (2018) "Few-shot image recognition by predicting parameters from activations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7229–7238.

[98] Lee, K. (2019), "Deep Mahalanobis Detector," `https://github.com/pokaxpoka/deep_Mahalanobis_detector`.

[99] Jankowski, M., D. Gündüz, and K. Mikolajczyk (2020) "Wireless Image Retrieval at the Edge," *IEEE Journal on Selected Areas in Communications*, **39**(1), pp. 89–100.

[100] Hsieh, K., G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu (2018) "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 269–286.

[101] Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014) "How transferable are features in deep neural networks?" *Advances in neural information processing systems*.

[102] Li, Z. and D. Hoiem (2017) "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **40**(12).

[103] Chen, W.-Y., Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang (2019) "A closer look at few-shot classification," *arXiv preprint arXiv:1904.04232*.

[104] Ager, S., "Omniglot-writing systems and languages of the world," `www.omniglot.com`.

[105] Vinyals, O., C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra (2016) "Matching networks for one shot learning," in *Proc. of the 30th International Conference on Neural Information Processing Systems*, pp. 3637–3645.

[106] Snell, J., K. Swersky, and R. Zemel (2017) "Prototypical networks for few-shot learning," in *Proc. of the 31st International Conference on Neural Information Processing Systems*, pp. 4080–4090.

[107] Finn, C., P. Abbeel, and S. Levine (2017) "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, PMLR, pp. 1126–1135.

[108] Lampert, C. H., H. Nickisch, and S. Harmeling (2009) "Learning to detect unseen object classes by between-class attribute transfer," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, pp. 951–958.

[109] Frome, A., G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov (2013) "Devise: A deep visual-semantic embedding model," in *Advances in neural information processing systems*, pp. 2121–2129.

[110] Socher, R., M. Ganjoo, C. D. Manning, and A. Ng (2013) "Zero-shot learning through cross-modal transfer," in *Advances in neural information processing systems*, pp. 935–943.

[111] Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017) "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, **5**, pp. 135–146.

[112] Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014) "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680.

[113] Bowles, C., L. Chen, R. Guerrero, P. Bentley, R. Gunn, A. Hammers, D. A. Dickie, M. V. Hernández, J. Wardlaw, and D. Rueckert (2018) "Gan augmentation: Augmenting training data using generative adversarial networks," *arXiv preprint arXiv:1810.10863*.

[114] Desai, N., R. K. Ganti, H. Kwon, I. Taylor, and M. Srivatsa (2018) "Unsupervised Estimation of Domain Applicability of Models," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, IEEE, pp. 34–39.

[115] Bendale, A. and T. E. Boult (2016) "Towards open set deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1563–1572.

[116] Hsu, Y.-C., Y. Shen, H. Jin, and Z. Kira (2020) "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10951–10960.

[117] Shalev, G., Y. Adi, and J. Keshet (2018) "Out-of-distribution detection using multiple semantic label representations," in *Advances in Neural Information Processing Systems*, pp. 7375–7385.

[118] Zaheer, M. Z., J.-h. Lee, M. Astrid, and S.-I. Lee (2020) "Old is Gold: Redefining the Adversarially Learned One-Class Classifier Training Paradigm," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14183–14193.

[119] Ruff, L., R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft (2018) "Deep one-class classification," in *International conference on machine learning*, pp. 4393–4402.

[120] Felemban, N., F. Mehmeti, and T. La Porta (2021) "VidQ: Video Query Using Optimized Audio-Visual Processing," in *Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pp. 51–60.

[121] (2019), "The use of tflite Model of C3D Network in Android," `https://github.com/tensorflow/tensorflow/issues/32866`.

[122] Abadi, M., A. Agarwal, et al. (2015), "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," Software available from tensorflow.org. URL `https://www.tensorflow.org/`

[123] Hershey, S., S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. (2017) "CNN architectures for large-scale audio classification," in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pp. 131–135.

[124] Rybach, D., C. Gollan, R. Schluter, and H. Ney (2009) "Audio segmentation for speech recognition using segment features," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4197–4200.

[125] Gemmeke, J. F., D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter (2017) "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 776–780.

[126] Shevchuk, D. (2020), "Audio Moment Retrieval based on Natural Language Query," .

[127] Carlini, N. and D. Wagner (2018) "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 1–7.

[128] O'Shaughnessy, D. (2008) "Automatic speech recognition: History, methods and challenges," *Pattern Recognition*, **41**(10), pp. 2965–2979.

[129] Kalfaoglu, M., S. Kalkan, and A. A. Alatan (2020) "Late temporal modeling in 3d cnn architectures with bert for action recognition," in *European Conference on Computer Vision*, Springer, pp. 731–747.

[130] Pham, H., Z. Dai, Q. Xie, and Q. V. Le (2021) "Meta pseudo labels," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11557–11568.

[131] Arnab, A., M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid (2021) "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6836–6846.

[132] Conforti, M., G. Cornuejols, and G. Zambelli (2014) *Integer programming*, Springer.

[133] Wu, J., C. Leng, Y. Wang, Q. Hu, and J. Cheng (2016) "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4820–4828.

[134] Nan, K., S. Liu, J. Du, and H. Liu (2019) "Deep model compression for mobile platforms: A survey," *Tsinghua Science and Technology*, **24**(6), pp. 677–693.

[135] Crasto, N., P. Weinzaepfel, K. Alahari, and C. Schmid (2019) "Mars: Motion-augmented rgb stream for action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7882–7891.

[136] Gupta, D., P. Bansal, and K. Choudhary (2018) "The state of the art of feature extraction techniques in speech recognition," *Speech and language processing for human-machine communications*, pp. 195–207.

[137] Wang, Y., S. Rawat, and F. Metze (2014) "Exploring audio semantic concepts for event-based video retrieval," in *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 1360–1364.

[138] Takahashi, N., M. Gygli, and L. Van Gool (2017) "Aenet: Learning deep audio features for video analysis," *IEEE Transactions on Multimedia*, **20**(3), pp. 513–524.

[139] Owens, A. and A. Efros (2018) "Audio-visual scene analysis with self-supervised multisensory features," in *Proceedings of the European Conference on Computer Vision (ECCV)*.

[140] Gao, R., T.-H. Oh, K. Grauman, and L. Torresani (2020) "Listen to look: Action recognition by previewing audio," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10457–10467.

[141] Buttcher, S., C. L. Clarke, and G. V. Cormack (2016) *Information retrieval: Implementing and evaluating search engines*, Mit Press.

[142] Zheng, L., S. Wang, Z. Liu, and Q. Tian (2014) "Packing and padding: Coupled multi-index for accurate image retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1939–1946.

[143] Chen, D. M., S. S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod (2010) "Inverted Index Compression for Scalable Image Matching." in *DCC*, p. 525.

[144] Zhou, W., H. Li, J. Sun, and Q. Tian (2017) "Collaborative index embedding for image retrieval," *IEEE transactions on pattern analysis and machine intelligence*, **40**(5), pp. 1154–1166.

Vita

Noor Felemban

Education:

- PhD, Computer Science and Engineering
  The Pennsylvania State University
  The Institute for Networking and Security Research (INSR)
  Advisor: Professor Thomas La Porta
  Thesis Title: Information Retrieval From Images and Videos in Mobile Networks

- MS, Computer Science and Engineering
  The Pennsylvania State University
  The Institute for Networking and Security Research (INSR)
  Thesis Title: On-demand Video Processing in Wireless Networks an Application: Action Recognition

- BS, Computer Engineering
  Prince Mohammad bin Fahd University, Saudi Arabia

Work Experience:

- (2013-2014) Teaching Assistant
  College of Computer Science and Information Technology, University of Dammam, Dammam, Saudi Arabia

- (2012-2013) Teaching Assistant,
  College of Computer Engineering and Science, Prince Mohammad bin Fahd University, Al-Khobar, Saudi Arabia.

Publications (selected)

- Felemban, N., et al. "PicSys: Energy-Efficient Fast Image Search on Distributed Mobile Networks." IEEE Transactions on Mobile Computing (2019).

- Felemban, N., Mehmeti, F., La Porta, T. F., and Kwon H. (2021) "EDIR: Efficient Distributed Image Retrieval of Novel Objects in Mobile Networks," in 2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS), IEEE, pp. 392–400.

- Felemban, N., Mehmeti, F., La Porta, T. F (2021) "VidQ: Video Query Using Optimized Audio-Visual Processing," in Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks, pp. 51–60.

- Mehmeti, F., Felemban, N., Lu, Z., Wheatman, K., Cirincione, G., & La Porta, T. F. (2021). Quality of information in gathering information via video analytics for military networks. IEEE Communications Magazine, 59(2), 50-55.