HUMAN ACTIVITY RECOGNITION FROM EGOCENTRIC VIDEOS AND ROBUSTNESS ANALYSIS OF DEEP NEURAL NETWORKS

Yantao Lu
Syracuse University

Abstract

In recent years, there has been significant amount of research work on human activity classification relying either on Inertial Measurement Unit (IMU) data or data from static cameras providing a third-person view. There has been relatively less work using wearable cameras, providing egocentric view, which is a first-person view providing the view of the environment as seen by the wearer. Using only IMU data limits the variety and complexity of the activities that can be detected. Deep machine learning has achieved great success in image and video processing in recent years. Neural network based models provide improved accuracy in multiple fields in computer vision. However, there has been relatively less work focusing on designing specific models to improve the performance of egocentric image/video tasks. As deep neural networks keep improving the accuracy in computer vision tasks, the robustness and resilience of the networks should be improved as well to make it possible to be applied in safety-crucial areas such as autonomous driving.

Motivated by these considerations, in the first part of the thesis, the problem of human activity detection and classification from egocentric cameras is addressed. First, a new method is presented to count the number of footsteps and compute the total traveled distance by using the data from the IMU sensors and camera of a smart phone. By incorporating data from multiple sensor modalities, and calculating the length of each step, instead of using preset stride lengths and assuming equal-length steps, the proposed method provides much higher accuracy compared to commercially available step counting apps. After the application of footstep counting, more complicated human activities, such as steps of preparing a recipe and sitting on a sofa, are taken into consideration. Multiple classification methods, non-deep learning and deep-learning-based, are presented, which employ both ego-centric camera and IMU data. Then, a Genetic Algorithm-based approach is employed

to set the parameters of an activity classification network autonomously and performance is compared with empirically-set parameters.

Then, a new framework is introduced to reduce the computational cost of human temporal activity recognition from egocentric videos while maintaining the accuracy at a comparable level. The actor-critic model of reinforcement learning is applied to optical flow data to locate a bounding box around region of interest, which is then used for clipping a sub-image from a video frame. A shallow and deeper 3D convolutional neural network is designed to process the original image and the clipped image region, respectively. Next, a systematic method is introduced that autonomously and simultaneously optimizes multiple parameters of any deep neural network by using a bi-generative adversarial network (Bi-GAN) guiding a genetic algorithm(GA). The proposed Bi-GAN allows the autonomous exploitation and choice of the number of neurons for the fully-connected layers, and number of filters for the convolutional layers, from a large range of values. The Bi-GAN involves two generators, and two different models compete and improve each other progressively with a GAN-based strategy to optimize the networks during a GA evolution. In this analysis, three different neural network layers and datasets are taken into consideration:

- First, 3D convolutional layers for ModelNet40 dataset. We applied the proposed approach on a 3D convolutional network by using the ModelNet40 dataset. ModelNet is a dataset of 3D point clouds. The goal is to perform shape classification over 40 shape classes.

- LSTM layers for UCI HAR dataset. UCI HAR dataset is composed of Inertial Measurement Unit (IMU) data captured during activities of standing, sitting, laying, walking, walking upstairs and walking downstairs. These activities were performed by 30 subjects, and the 3-axial linear acceleration and 3-axial angular velocity were collected at a constant rate of 50Hz.

- 2D convolutional layers for Chars74k Dataset. Chars74k dataset contains 64 classes

(0-9, A-Z, a-z), 7705 characters obtained from natural images, 3410 hand-drawn characters using a tablet PC and 62992 synthesised characters from computer fonts giving a total of over 74K images.

In the final part of the thesis, network robustness and resilience for neural network models is investigated from adversarial examples (AEs) and automatic driving conditions. The transferability of adversarial examples across a wide range of real-world computer vision tasks, including image classification, explicit content detection, optical character recognition (OCR), and object detection are investigated. It represents the cybercriminal's situation where an ensemble of different detection mechanisms need to be evaded all at once. Novel dispersion Reduction(DR) attack is designed, which is a practical attack that overcomes existing attacks' limitation of requiring task-specific loss functions by targeting on the "dispersion" of internal feature map. In the autonomous driving scenario, the adversarial machine learning attacks against the complete visual perception pipeline in autonomous driving is studied. A novel attack technique, tracker hijacking, that can effectively fool Multi-Object Tracking (MOT) using AEs on object detection is presented. Using this technique, successful AEs on as few as one single frame can move an existing object in to or out of the headway of an autonomous vehicle to cause potential safety hazards.

HUMAN ACTIVITY RECOGNITION FROM EGOCENTRIC VIDEOS AND ROBUSTNESS ANALYSIS OF DEEP NEURAL NETWORKS

By

Yantao Lu

M.S., Syracues University, Syracuse, US, 2015

B.S., Xian Jiaotong University, Xi'an, China, 2013

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

May 2020

Acknowledgements

First, I would like to thank my advisor, Prof. Senem Velipasalar, for her help and guidance during my entire Ph.D. study. In this period, I have gained much experience in research, and completed many studies in the area of computer vision and machine learning.

I would like to thank my parents and my wife Yilan for supporting me to pursue my Ph.D. degree.

I would also like to thank my Ph.D. defense committee members Prof. Lixin Shen, Prof. Qinru Qiu, Prof. Pramod Varshney, Prof. Makan Fardad and Prof. Garrett Katz for the reading and guidance of my thesis.

Finally, I would like to thank all my labmates and friends for their support during these years in Syracuse University and Baidu Sunnyvale research lab. I had really wonderful time studying and working with them.

For all the support, time, motivation, and patience: this thesis is dedicated to my family and friends, especially my wife Yilan.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

There has been significant amount of research work on human activity classification relying either on Inertial Measurement Unit (IMU) data or data from static cameras providing a third-person view. However there are relatively less work using wearable cameras, providing first-person or egocentric view, and even fewer approaches combining egocentric video with IMU data. Using only IMU data limits the variety and complexity of the activities that can be detected. On the other hand, most of the camera-based activity detection works use static cameras watching the subjects, and thus providing a third-person view. Wearable sensors are becoming more and more ubiquitous in our lives. Potential applications of activity recognition from egocentric videos include life logging, video diaries and video summarization, health care, elderly care, personal assistance to users or caregivers, navigation and assistance for the visually impaired, robotics, human-human and human-robot interaction and law enforcement. Compared to static cameras, there have been relatively less work as well as much fewer datasets focusing on wearable cameras, egovision and combination of wearable camera data with other sensor modalities.

Step counting is being increasingly used as an activity-level measure, which is evidenced by different types of widely available commercial wristbands, pedometers, and apps developed for smart phones and smart watches. In addition to measuring daily activity levels and keeping logs for health monitoring, an accurate and reliable count of footsteps can be used for motion estimation, calculating traveled distance and indoor navigation. Yet, most of the available devices and approaches for step counting rely only on accelerometer data, and thus are prone to over-counting. Accelerometer based step counting can also become unreliable during slow walking, or other activities such as jumping, and if a person stops and starts walking again. In addition, most existing devices calculate the traveled distance based on the counted number of steps and a preset stride length. On the other hand, most work on distance calculation relies on GPS data, which might not be suitable for GPS-denied areas and indoor environments. It is benefited to use an autonomous and robust method for counting footsteps, and tracking and calculating stride length by using both accelerometer and camera data.

The design of the network architecture and the choice of parameters are important factors affecting the performance of deep neural networks. However, there has not been much work on developing an established and systematic way of building the structure of a neural network, and this task heavily depends on trial and error, empirical results, and the designer's experience. Considering that there are many design and parameter choices, such as the number of layers, number of neurons in each layer, number of filters at each layer, the type activation function, the choice of using drop out or not and so on, it is not possible to cover every possibility, and it is very hard to find the optimal structure. In fact, often times some common settings are used without even trying different ones. Moreover, the hyper-parameters in training phase also play important role on how well the model will perform. Likewise, these parameters are also tuned manually in an empirical way most of the time. Genetic Algorithms (GA) have been used before to determine network parameters. Yet, GAs perform a finite search over a discrete set of pre-defined candidates.

There have been many methods for human activity classification, which rely on third-person video data from static cameras watching activities of person(s). Compared to human activity video datasets obtained from static cameras, there has been much less video data from egocentric cameras. Similarly, compared to works that use static cameras installed in the environment, there has been relatively less work using egocentric videos, meaning providing the first-person view from wearable cameras. Many approaches have been developed, which employ deep neural networks to perform human activity classification. In addition to the networks having deeper structures, higher resolution image data needs to be processed in many cases. This increases the computational complexity. Thus, researchers have also focused on speeding up the processing. However, most of the work are tailored to particular network structures, and may not generalize well to new architectures.

As neural network models are reaching higher and higher accuracies in computer vision fields, more research should focus on improving the robustness performance. However, there are few robustness improvement recently in image processing field. Recent research in adversarial learning has brought the weaknesses of deep neural networks (DNNs) to the spotlights of security and machine learning studies. Given a deep learning model, it is easy to generate adversarial examples (AEs), which are close to the original but are misclassified by the model. More importantly, their effectiveness sometimes transfer, which may severely hinder DNN based applications especially in security critical scenarios. While such vulnerabilities are alarming, little attention has been paid on the realistic threat model of commercial or proprietary vision-based detection systems against real-world cybercriminals, which turn out to be quite different from those intensively studied by aforementioned research. To overcome the weakness of deep learning in individual domain, real-world CV systems tend to employ an ensemble of different detection mechanisms to prevent evasions. To evade detections with uncertain mechanisms, attackers turn to generate adversarial examples that transfer across CV tasks. However, most of the methods are designed for image classification tasks, and rely on task-specific loss function (e.g., cross-entropy loss), which limits their effectiveness when transferred to other CV tasks. On the other hand, significant progress in Machine Learning (ML) techniques like Deep Neural Networks (DNNs) recently has enabled the development of safety-critical ML systems like autonomous vehicles. Recent results show that autonomous vehicles have become very efficient in practice and already driven millions of miles without any human interventions. Twenty US states including California, Texas, and New York have recently passed legislation to enable testing and deployment of autonomous vehicles. However, despite the tremendous progress, DNNs have been shown to be vulnerable to adversarial examples, inputs that are carefully crafted to fool the model. Even worse, these perturbed images, if carefully crafted, can stay adversary when taken as input from the physical world using a camera.

1.1 Research Impact

We explore a few different computer vision tasks within the context of improving approaches via fusion based approaches. These computer vision tasks have a variety of implications across many applications. Moreover, we explore robustness improvement on neural networks.

By leveraging data from multiple sensor modalities, more specifically egocentric video and IMU sensor data from wearable devices, we present a robust and autonomous method to perform fine-grain activity classification. In contrast to many CNN-based approaches, we use a capsule network to obtain features from egocentric video data. We incorporate a generative adversarial network-based approach to increase the range of parameters that can be chosen autonomously. The Bi-GAN allows the autonomous exploitation and choice of the number of neurons, for the fully-connected layers, and number of filters for the convolutional layers, from a large range of values. The Bi-GAN approach can be used to autonomously refine the number of convolutional layers and dense layers, number and size of kernels, and the number of neurons; choose the type of the activation function; and decide whether to use dropout and batch normalization or not, to improve the accuracy of different deep neural network architectures. Without loss of generality, the method has been tested with the ModelNet database, and compared with the 3D Shapenets and two

GA-only methods.

For robustness analysis, we present Dispersion Reduction (DR) attack to improve the cross-task transferability of adversarial examples. Specifically, our method reduces the dispersion of intermediate feature maps by iterations. Compared to existing black-box attacks, the results on MS COCO, PASCAL VOC and ImageNet show that DR method performs better on attacking black-box cross-CV-task models. One intuition behind the DR attack is that by minimizing the dispersion of feature maps, images become ”featureless”. This is because few features can be detected if neuron activations are suppressed by perturbing the input. Moreover, with the observation that low-level features bear more similarities across CV models, we hypothesize that the DR attack would produce transferable adversarial examples when one of the middle convolution layers is targeted. Evaluation on different CV tasks shows that this enhanced attack greatly degrades model performance by a large margin compared to the state-of-the-art attacks, and thus would facilitate evasion attacks against a different task model or even an ensemble of CV-based detection mechanisms. We hope that DR attack can serve as benchmark for evaluating robustness of future defense mechanisms.

In autonomous driving perception system, we are the first to study adversarial machine learning attacks against the complete visual perception pipeline in autonomous driving, i.e., both object detection and MOT. We discover a novel attack technique, tracker hijacking, that exploits the tracking error reduction process in MOT and can enable successful AEs on as few as one frame to move an existing object in to or out of the headway of an autonomous vehicle to cause potential safety hazards. The evaluation results show that on average when 3 frames are attacked, our attack can have a nearly 100% success rate while attacks that blindly target object detection only have up to 25%. Our discovery and results strongly suggest that MOT should be systematically considered and incorporated into future adversarial machine learning research targeting the visual perception in autonomous driving. Our work initiates the first research effort along this direction, and we hope that it can inspire more future research into this largely overlooked research perspective.

Moreover, the training procedure and model scale optimality, we investigate boosting tickets, sub-networks coupled with certain initialization that can be trained with significantly faster convergence rate. As a practical application, in the adversarial training scheme, we show pruning a weakly robust model allows to find boosting tickets that can save up to 49% of the total training time to obtain a strongly robust model that matches the state-of-the-art robustness. We reveals an direction to investigate whether there is a way to find boosting tickets without training the full model beforehand, as it is technically not necessary.

The research presented in this thesis resulted in several publications including respected Institute of Electrical and Electronics Engineers (IEEE) journals and international conference proceedings.

1.2 Publications

Peer-reviewed Journals

- Y Lu, S Velipasalar, ”Autonomous Human Activity Classification From Wearable Multi-Modal Sensors”, IEEE Sensors Journal 19 (23), 11403-11412
- Y Lu, S Velipasalar, ”Autonomous footstep counting and traveled distance calculation by mobile devices incorporating camera and accelerometer data”, IEEE Sensors Journal 17 (21), 7157-7166

Peer-reviewed Conferences

- Y Lu, S Velipasalar, ”Robust footstep counting and traveled distance calculation by mobile phones incorporating camera geometry”, 2016 IEEE International Conference on Image Processing (ICIP), 464-468

- Y Lu, S Velipasalar, ”Human activity classification from wearable devices with cameras”, 2017 51st Asilomar Conference on Signals, Systems, and Computers, 183-187
- Y Lu, S Velipasalar, ”Human activity classification incorporating egocentric video and inertial measurement unit data”, 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 429-433
- Y Lu, Y Li, S Velipasalar, ”Efficient Human Activity Classification from Egocentric Videos Incorporating Actor-Critic Reinforcement Learning”, 2019 IEEE International Conference on Image Processing (ICIP), 564-568
- Y Lu, S Velipasalar, ”Autonomous Choice of Deep Neural Network Parameters by a Modified Generative Adversarial Network”, 2019 IEEE International Conference on Image Processing (ICIP), 3846-3850
- Y Jia, Y Lu, J Shen, QA Chen, H Chen, Z Zhong, T Wei, ”Fooling Detection Alone is Not Enough: First Adversarial Attack against Multiple Object Tracking”, 2020 International Conference on Learning Representations (ICLR)
- Y Lu, Y Jia, J Wang, B Li, W Chai, L Carin, S Velipasalar, ”Enhancing Cross-task Black-Box Transferability of Adversarial Examples with Dispersion Reduction”, submitted to 2020 Conference on Computer Vision and Pattern Recognition (CVPR)

Book Chapter

- Y Lu, S Velipasalar, ”Wearable Sensor Applications: Processing of Egocentric Videos and Inertial Measurement Unit Data”, Embedded, Cyber-Physical, and IoT Systems, 149-173

1.3 Literature Review

1.3.1 Footstep Counting and Traveled Distance Calculation by Mobile Devices

With their widespread availability, and thanks to their rich set of onboard sensors and relatively powerful processors, smart phones are increasingly being employed in detecting different human activities. Zhang et al. [8] describe a hierarchical method of activity classification based on a smart phone, equipped with an embedded 3D-accelerometer. Ozcan and Velipasalar [9] present a fall detection method incorporating accelerometer and camera data on portable devices. A survey by Song et al. [10] presents various approaches, employing wearable sensors and mobile phones, for activity monitoring, fall detection, and heart rate and sleep sensing.

There is a significant body of work on activity monitoring and step counting based on accelerometer data. Guo et al. [11] present an evaluation of wearable accelerometer-based activity monitoring devices. Park et al. [12] presented an accelerometer-based activity tracker on smartphones. Capela et al. [13] use accelerometer data to analyze walking patterns and compute the distance traveled for clinical purposes when the phone is attached on a belt around the waist. Jang et al. [14] presented a robust two-axis accelerometer-based step detection method, in which the sensor is placed on the ankle. Alvarez et al. [15] proposed a three-axis accelerometer-based step length estimator algorithm. Pan and Lin [16] proposed an accelerometer-based step counting algorithm for smart phones, which does not require the user to have the smart phone attached to the body while walking. Brajdic and Harle [17] tried different locations for the smartphones to test the accelerometer-based step-counter algorithms, and found that certain locations, such as back pocket of pants, degrade the performance significantly.

In general, accelerometer-based approaches can count other routine movements, such as movements of the phone, as steps, and are prone to over-counting. Accelerometer-based step counting can also become unreliable during slow walking, or other activities such as jumping, if a person stops and starts walking again, and when people are exposed to acceleration, e.g. inside a moving vehicle or an elevator. Marschollek et al. [18] compared several accelerometer-based step counting algorithms on both healthy subjects and geriatric patients with mobility impairments. It is stated that [18] none of the algorithms worked very well, and that more research is needed to prove the validity of these algorithms for the elderly.

Instead of accelerometer data, Aubeck et al. [19] use data captured from a camera. They use template matching to detect steps, and state that template matching creates problems for fast movements, since moving objects are often fuzzy [19]. Ozcan and Velipasalar [20] also present a camera-based approach for step counting. The FAST feature points are detected, and the movement of the cluster center is used to count steps. However, these works still rely on a single type of sensor, in this case the camera, and do not address the issue of calculating the traveled distance.

Many of the aforementioned works only focus on the step counting and do not address the calculation of the total traveled distance. Most existing approaches and commercial devices calculate the traveled distance based on the number of counted steps and a pre-set stride length. This stride length is usually asked from the user, or is calculated based on the user's height. On the other hand, most work on step length calculation relies on GPS data, which might not be suitable for GPS-denied areas and complex indoor environments [21] [22] [23], for which the GPS data is either unavailable or highly unreliable.

1.3.2 Autonomous Human Activity Classification from Wearable Multi-Modal Sensors

Many approaches have been proposed to perform human activity classification from different sensors. Most of the existing methods rely either on Inertial Measurement Unit (IMU) data [24][25][26][27][28][29][30][31] or data from static cameras in the environment providing a third-person view [32][33][34][35][36].

Mannini and Sabatini [24] [25] use IMU data to classify activities of sitting, standing, lying down, walking, running, climbing stairs and cycling. Bayat et al. [30] also employ IMU data to classify activities, such as dancing, going up and down the stairs, slow and fast walking and running, and compare different classifiers. Ordóñez and Roggen [31] use accelerometer and gyroscope data, and employ convolutional and Long Short Term Memory (LSTM) recurrent units for activity recognition. Even though the systems using only IMU data are computationally efficient, they are limited in terms of the variety and complexity of the activities that they can detect. They also cannot provide enough context. For example, IMU data can help to detect a sitting activity, but cannot help determine the type of furniture the subject sits on or the environment the subject is in. Having data from a camera sensor fills in these blanks by providing abundant information about surroundings and the objects with which the subject is interacting.

On the other hand, most of the camera-based activity detection works use static cameras watching the subjects, and thus providing a third-person view. Karpathy et al. [32] merge different convolutional neural networks (CNN) for large-scale video classification, and present results on the UCF-101 Action Recognition Dataset [37]. Donahue et al. [33] use long-term recurrent network together with CNNs, and also evaluate on UCF-101 dataset. Instead of using 2D CNN and LSTM, different approaches have been presented using 3D CNNs for learning spatiotemporal features [34]. Montes et al. [35] use a 3D CNN together with LSTM to achieve temporal activity detection in untrimmed videos. Instead of using LSTM, Buch et al. [36] use 3D CNN together with GRUs (Gated Recurrent Unit) on videos from a third-person view. Heilbron et al. [38] present ActivityNet, which is a large-scale video benchmark for human activity understanding, and propose a method based on 3D CNNs. In this video set, the majority of videos are not egocentric.

Wearable sensors are becoming more and more ubiquitous in our lives. Potential applications of activity recognition from egocentric videos include life logging, video diaries and video summarization, health care, elderly care, personal assistance to users or caregivers, navigation and assistance for the visually impaired, robotics, human-human and human-robot interaction and law enforcement. Several surveys [10][39] have been published describing various techniques for activity classification, and heart rate and sleep sensing by wearable sensors. Compared to static cameras, there have been relatively less work as well as much fewer datasets focusing on wearable cameras, egovision and combination of wearable camera data with other sensor modalities.

Existing works focusing on egocentric videos differ in terms of the types of objects and activities that they detect. There have been methods focusing only on hand detection from egocentric videos [40]. Other approaches employing egocentric video data either classify the activities observed by the camera [41][42] or the activities of the person wearing the camera [43][44][45][46][47][48]. Ryoo and Matthies [42] present a method to recognize what activities others are performing to the observer or the person or robot wearing the camera. Pirsiavash and Ramanan [45] presented a dataset of egocentric videos covering Activities of Daily Living (ADL), and reported 40.6% accuracy over 18 classes. McCandless and Grauman [46] presented a method for activity recognition by learning the spatio-temporal partitions. They used the same ADL dataset, and reported 38.7% accuracy over 18 classes. Lu and Grauman [49] presented a method for story-driven video summarization and tested it on the ADL dataset [45]. Moghimi et al. [48] presented a method for activity detection using RGB-D egocentric videos. Nguyen et al. [50] provided a survey and review of the egocentric vision systems for the recognition of activities of daily living. It was concluded that the performance of current systems is far from satisfactory.

The aforementioned approaches, which focus on egocentric videos, are based on only a single sensor modality, namely the camera. There have been even fewer approaches that combine egocentric video data with IMU data [51][52][53][54][55]. Zhan et al. [52] use a smartphone attached on top of safety goggles to collect video and 3-axis acceleration data. They use optical flow vectors from camera data and classify 12 activities, including walking, going upstairs/downstairs, sitting, standing, drinking and writing. Windau and Itti [53] also use both IMU and camera data from a prototype eyeglass setup. They extract GIST features from camera data to perform indoor/outdoor classification. They report 81.5% accuracy for classifying 20 activities including lying down, walking, jogging, biking, running, playing cello, playing piano, computer work, folding laundry and driving car. However, both of these methods still focus on activities that can be classified by only accelerometer data. In other words, they do not perform detection of objects in the scene, and do not focus on activity types involving interactions with different types of objects, which cannot be classified by only accelerometer data. In earlier work, Spriggs et al. [54] used the CMU Multi-Modal Activity (CMU-MMAC) database [56], and presented a method for temporal segmentation and activity classification, focusing on recipe preparation, by extracting the GIST features from the egocentric video data. They reported 57.8% as the highest performance.

Different from the eyeglass setup, Conti et al. [51] employed the various sensors on a smartwatch to perform context classification over only 5 classes (morning preparation, walking outdoors, public transportation, in the car and in the office).

It should be emphasized that many activities can be very close to each other in the "activity space", in other words, can be very similar, such as using a spoon versus using a fork. In this case, adding another sensor modality, namely the camera, and detecting objects become even more important to identify activities involving interactions with various-sized objects. The problem gets much harder for fine-grained classification of activities. As mentioned above, most of the existing work does not focus on fine-grained activity classification. On the other hand, the relatively small number of existing works on fine-grained classification have reported lower accuracies.

1.3.3 Autonomously and Simultaneously Refining Deep Neural Network Parameters by a Bi-Generative Adversarial Network Aided Genetic Algorithm

There have been works focusing on optimizing neural network architectures. Most of the proposed approaches are based on the GAs [57], or evolutionary algorithms, which are heuristic search algorithms. Benardos and Vosniakos [58] proposed a methodology for determining the best neural network architecture based on the use of a GA and a criterion that quantifies the performance and the complexity of a network. In their work, they focus on optimizing four architecture decisions, which are the number of layers, the number of neurons in each layer, the activation function in each layer, and the optimization function. Magnier and Haghighat [59] presented an optimization methodology based on a combination of a neural network and a multi-objective evolutionary algorithm. The methodology was used for the optimization of thermal comfort and energy consumption in a residential house. Leung et al. [60] presented the tuning of the structure and parameters of a neural network using an improved GA. Ritchie et al. [61] proposed a method to automate neural network architecture design process for a given dataset by using genetic programming. Islam et al. [62] employed a genetic algorithm for finding the optimal number of neurons in the input and hidden layers. They apply their approach to power load prediction task and report better results than a manually designed neural network. Yet, their approach is used to optimize only the number of neurons for input and hidden layers, and optimization of other important design decisions such as the number of layers or activation function type are not discussed. Stanley and Miikkulainen [63] presented the NEAT algorithm for optimizing neural networks by evolving topologies and weights of relatively small recurrent networks. In a recent work, Miikkulainen et al. [64] proposed CoDeepNEAT algorithm for optimizing deep learning architectures through evolution by extending existing neuroevolution methods to topology, components and hyperparameters.

The genetic algorithm-based optimization uses a given set of blueprints and models, i.e. it performs a finite search over a discrete set of candidates. Thus, GAs, in general, cannot generate unseen configurations, and they can only make a combination of preset parameters. GAs are good at searching better solutions from limited possibilities, such as type of layers and activation functions. However, it cannot search for a solution which is not defined before. In addition, the complexity of GAs increases significantly when the number of choices increases to large scale. Rylander [65] has shown that the generations needed for convergence increases exponentially with the node size.

Apart from the genetic algorithms, Bergstra and Bengio [66] have proposed random search for hyper-parameter optimization, and stated that randomly chosen trials are more efficient for hyperparameter optimization than trials on a grid. Yan and Zhang [67] optimized architectures' width and height with growing running time budget through submodularity and suparmodularity.

Generative Adversarial Networks (GANs) [68] are one of the important milestones in deep learning research. In contrast to CNNs, which extract rich and dense representations of the source domain, and may eventually map source instances into some classes, GANs generate instances of the source domain from small noise. They employ deconvolution operators, or transposed convolutions, to generate N-D instances from 1-D noise. GAN's power comes from the competition with the discriminator, which decides whether the generated instance belongs to the source domain. Discriminator acts like the police who is trying to intercept counterfeit money, where in this case the generator is the counterfeiter. Generator and discriminator are trained together until discriminator cannot distinguish the generated instances from the instances in the source domain. GANs have been adapted in many applications [69, 70, 71, 72, 73].

1.3.4 Efficient Human Activity Classification From Egocentric Videos Incorporating Actor-Critic Reinforcement Learning

There have been many methods for human activity classification, which rely on third-person video data [32, 33, 74, 35, 36] from static cameras watching activities of person(s). Compared to human activity video datasets obtained from static cameras, there has been much less video data from egocentric cameras. Similarly, compared to works that use static cameras installed in the environment, there has been relatively less work using egocentric videos, meaning providing the first-person view from wearable cameras.

Heilbron et al. [38] presented the ActivityNet, which is a large-scale video benchmark for human activity understanding, and proposed a method based on 3D Convolutional Neural Networks (CNNs). In this video dataset, majority of videos are not egocentric. Karpathy et al. [32] proposed a method for large-scale video classification, and presented results on the UCF-101 Action Recognition Dataset [37], which mostly contains third-person videos. Instead of using 2D CNN and LSTM, different approaches have been presented using 3D CNNs for learning spatiotemporal features [74]. Montes et al. [35] use a 3D CNN together with LSTM to achieve temporal activity detection in untrimmed videos. Instead of using LSTM, Buch et al. [36] use 3D CNN together with Gated Recurrent Units on videos from a third-person view.

Many approaches have been developed, which employ deep neural networks to perform human activity classification. In addition to the networks having deeper structures, higher resolution image data needs to be processed in many cases. This increases the computational complexity. Thus, researchers have also focused on speeding up the processing [75, 76, 77, 78]. However, these models are mostly tailored to particular network structures, and may not generalize well to new architectures. Minh et al. [79] introduced a recurrent neural network-based model to represent visual attention, and applied it to the image classification task and a simple game.

Reinforcement learning (RL) provides a mathematical framework for learning or deriving policies that map situations (i.e. states) into actions with the goal of maximizing an accumulative reward [80]. Unlike supervised learning, in RL the agent (i.e. learner) learns the policy for decision making through interactions with the environment. The goal of the agent is to maximize the cumulative reward by taking the optimal action at each time step according to the current state while considering the trade-off between explorations and exploitations. The combination of conventional Q-learning and deep neural network, i.e. Deep Q-network (DQN) [81], provides a breakthrough in deep reinforcement learning (DRL). However, the neural network in DQN needs to accumulate enough samples of values, and the data needed for its training can either come from a model-based simulation or from actual measurement [82]. Originally developed by DeepMind, the DRL provides a promising data-driven, adaptive technique in handling large state space of complicated control problems [83]. The actor-critic deep reinforcement learning [84] has overcome difficulties in learning control policies of systems with continuous state and action space, which provides a potential solution for efficient real-time processing of video clips in our case.

1.3.5 Cross-task Transferability of Adversarial Examples with Dispersion Reduction

Recent progress in adversarial machine learning has brought the weaknesses of deep neural networks (DNNs) into the spotlight, and drawn the attention of researchers working on security and machine learning. Given a deep learning model, it is easy to generate adversarial examples (AEs), which are close to the original input, but are easily misclassified by the model [85, 86]. More importantly, their effectiveness sometimes transfers, which may severely hinder DNN-based applications especially in security critical scenarios [87, 88, 89]. While such problems are alarming, little attention has been paid to the threat model of commercially deployed vision-based systems, wherein deep learning models across different tasks are assembled to provide fail-safe protection against evasion attacks. Such a threat model turns out to be quite different from those models that have been intensively studied by aforementioned research.

Cross-task threat model. Computer vision (CV) based detection mechanisms have been deployed extensively in security-critical applications, such as content censorship and authentication with facial biometrics, and readily available services are provided by cloud giants through APIs (e.g., Google Cloud Vision [90]). The detection systems have long been targeted by evasive attacks from cybercriminals, and it has resulted in an arm race between new attacks and more advanced defenses. To overcome the weakness of deep learning in an individual domain, real-world CV systems tend to employ an ensemble of different detection mechanisms to prevent evasions. Underground businesses embed promotional contents such as URLs into porn images with sexual content for illicit online advertising or phishing. A detection system, combining Optical Character Recognition (OCR) and image-based explicit content detection, can thus drop posted images containing either suspicious URLs or sexual content to mitigate evasion attacks. Similarly, a face recognition model that is known to be fragile [91] is usually protected by a liveness detector to defeat spoofed digital images when deployed for authentication. Such ensemble mechanisms are widely adopted in real-world CV deployment.

To evade detection systems with uncertain underlying mechanisms, attackers turn to generating adversarial examples that transfer across CV tasks. Many adversarial techniques on enhancing transferability have been proposed [92, 89, 87, 88]. However, most of them are designed for image classification tasks, and rely on task-specific loss functions (e.g., cross-entropy loss), which limits their effectiveness when transferred to other CV tasks.

1.3.6 Robust Analysis of Multiple Object Tracking Based on Automatic Driving

Since the first physical adversarial examples against traffic sign classifier demonstrated by Eykholt et al. [93], several work in adversarial machine learning [94, 95, 96, 97, 98, 99] have been focused on the visual perception task in autonomous driving, and more specifically, the object detection models. To achieve high attack effectiveness in practice, the key challenge is how to design robust attacks that can survive distortions in real-world driving scenarios such as different viewing angles, distances, lighting conditions, and camera limitations. For example, Lu et al. [96] shows that AEs against Faster-RCNN [100] generalize well across a sequence of images in digital space, but fail in most of the sequence in physical world; Eykholt et al. [94] generates adversarial stickers that, when attached to stop sign, can fool YOLOv2 [101] object detector, while it is only demonstrated in indoor experiment within short distance; Chen et al. [99] generates AEs based on expectation over transformation techniques, while their evaluation shows that the AEs are not robust to multiple angles, probably due to not considering perspective transformations [98]. It was not until recently that physical adversarial attacks against object detectors achieve a decent success rate (70%) in fixed-speed (6 km/h and 30 km/h) road test [98].

MOT aims to identify objects and their trajectories in video frame sequence. With the recent advances in object detection, tracking-by-detection [102] has become the dominant MOT paradigm, where the detection step identifies the objects in the images and the tracking step links the objects to the trajectories (ie., trackers). Such paradigm is widely adopted in autonomous driving systems today [1, 2, 3, 4, 5, 6, 7]. A per-track Kalman filter [1, 2, 103, 104, 105] is used to maintain the state model, which operates in a recursive predict-update loop: the predict step estimates current object state according to a motion model, and the update step takes the detection results $detc|_t$ as measurement to update its state estimation result $track|_t$.

The association between detected objects with existing trackers is formulated as a bipartite matching problem [106, 103, 104] based on the pairwise similarity costs between the trackers and detected objects, and the most commonly used similarity metric is the spatial-based cost, which measures the overlapping between bounding boxes, or bboxes [1, 107, 108, 106, 103, 104, 109, 105, 110, 111]. To reduce errors in this association, an accurate velocity estimation is necessary in the Kalman filter prediction [112, 113]. Due to the discreteness of camera frames, Kalman filter uses the velocity model to estimate the location of the tracked object in the next frame in order to compensate the object motion between frames.

1.3.7 Practical Pruning for Adversarial Training

Pruning has served as an important technique for removing redundant structure in neural networks [114, 115, 116, 117]. Properly pruning can reduce cost in computation and storage without harming performance. However, pruning was until recently only used as a post-processing procedure, while pruning at initialization was believed ineffective [115, 116]. Recently, [118] proposed the lottery ticket hypothesis, showing that for a deep neural network there exist sub-networks, when trained from certain initialization obtained by pruning, performing equally or better than the original model with commensurate convergence rates. Such pairs of sub-networks and initialization are called winning tickets.

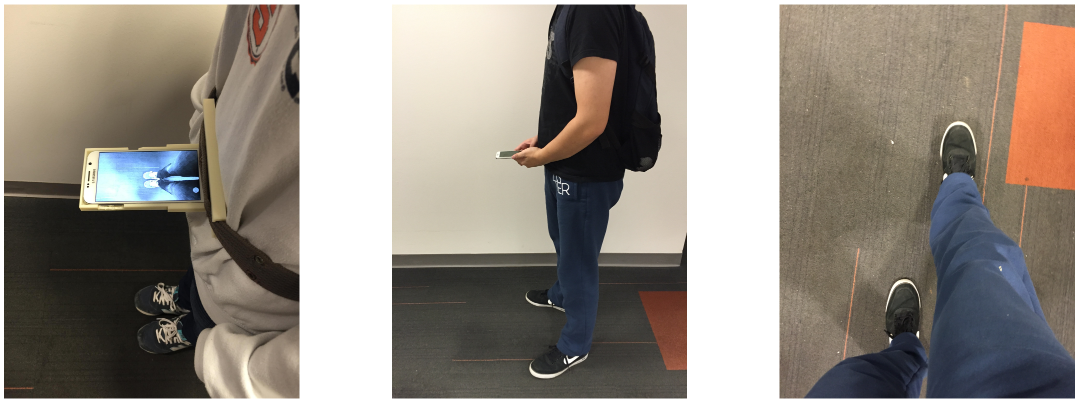
This phenomenon indicates it is possible to perform pruning at initialization. However, finding winning tickets still requires iterative pruning and excessive training. Its high cost limits the application of winning tickets.

Although [118] shows that winning tickets converge faster than the corresponding full models, it is only observed on small networks, such as a convolutional neural network (CNN) with only a few convolution layers. In chapter 10, we show that for a variety of model architectures, there consistently exist such sub-networks that converge significantly faster when trained from certain initialization after pruning. We call these *boosting tickets*.

Chapter 2

Footstep Counting and Traveled Distance Calculation by Mobile Devices

This chapter studies the autonomous and robust system for counting footsteps, tracking and calculating stride length, and computing the total traveled distance by using camera as well as accelerometer and gravity sensors of a smart phone or a Google™ glass. To provide higher precision, as opposed to using a preset step or stride length, the system incorporates data from different sensor modalities to track and calculate the step length for each step. In addition, if camera is tilted significantly, the angle data obtained from the gravity sensor is used to account for camera geometry and increase the precision of the calculated step length. The system performs well even when feet of multiple people are visible in the image, and also across subjects with varying walking pace, heights and stride lengths. If a smart phone is being used, the subjects can hold the phone as they normally would when they read e-mails, or browse pages on the web. A holder is also designed and printed, by using a 3D printer, for hands-free usage and longer walking experiments. The images captured by the rear-facing camera of the smart-phone are used for the method. Moreover, we used a Google™ glass as a different use scenario. Different usage styles, and example images from a smart phone and Google™ glass can be seen in Fig 2.1.



(a) (b) (c)

Figure 2.1: (a) Experimental setup showing the subject with the phone holder, (b) the user holding the phone, (c) image from the camera while walking.

Various experiments are performed with a smart phone and a Google glass™. With smart phone experiments 15-20 different subjects either carried the phone by hand or used the holder around the waist. The Google ™ glass experiments are performed with 10 subjects. The system is compared with the accelerometer-based step counter apps. In different sets of experiments, subjects walk continuously, they stopped and started walking again, and also followed a zig-zag-like pattern. For comparison purposes, the subjects carry three mobile devices simultaneously at different body locations. More specifically, two smart phones are carried in front pant pockets and in a backpack, and these phones run readily available, accelerometer-based apps for step counting and distance calculation. The third phone is carried in a holder or held by hand to capture data for the method as seen in Fig. 2.1(a) and (b). The method has been compared with accelerometer-based methods in terms of both counted steps and the traveled distance. The results show that the method provides the lowest average error rate in number of steps taken and the distance traveled compared to commercially available, accelerometer-based step counters and apps.

2.1 Methodology

The flow diagram of the method is shown in Fig. 2.2. The details of the step counting and traveled distance calculation will be described in Sections 2.1.1 and 2.1.2, respectively.

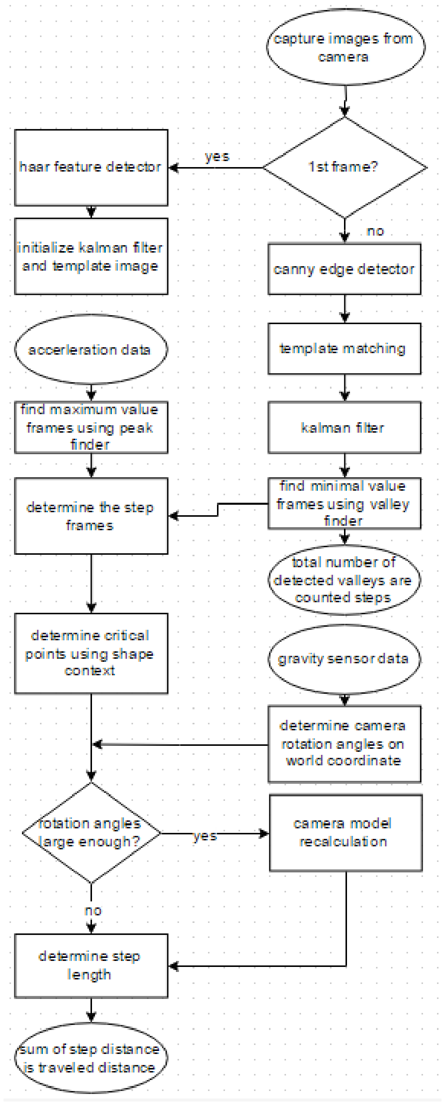

capture images from camera
1st frame?
yes
haar feature detector
initialize kalman filter and template image
no
canny edge detector
template matching
acceleration data
find maximum value frames using peak finder
kalman filter
find minimal value frames using valley finder
determine the step frames
total number of detected valleys are counted steps
determine critical points using shape context
gravity sensor data
determine camera rotation angles on world coordinate
rotation angles large enough?
yes
camera model recalculation
no
determine step length
sum of step distance is traveled distance

Figure 2.2: Flow diagram

2.1.1 Step Counting

We have trained a Haar-based detector for feet. Initially, only for the first frame, Haar-based detection is used to detect the feet, and obtain bounding boxes around them. The sizes and locations of the bounding boxes are used to initialize the matching template size, search region size and the Kalman filter tracker. More specifically, the template size is set to be the same as the bounding box size. The search region size, to be used in the following frames, is set to be twice the size of the bounding box for each feet.

For subsequent images, edges are detected first by using the Canny edge detection algorithm. Several example images obtained after edge detection can be seen in Fig. 2.3(b) and 2.4(b) for different floor surfaces. Then, the binary template, shown in Fig. 2.5, is used to perform template matching only in the determined search regions. Since much smaller search regions are used, instead of the entire image, for template matching, the computational efficiency of the method is increased. Figures 2.3(d) and 2.4(d) show the search regions. Figures 2.3(c) and 2.4(c) show the normalized value of correlation coefficient at each pixel location in the search regions, wherein brighter pixels correspond to higher values, and the two brightest spots are the locations of two detected feet. The detected locations are marked in green in Figures 2.3(d) and 2.4(d).

As mentioned above, a Kalman filter tracker is created in the first image by using the bounding boxes around the feet detected by the Haar-based detector. The feet locations are then tracked in the subsequent frames. If feet locations cannot be detected by using template matching, then the locations are updated by using the predictions from the tracker. The detected and tracked locations for the feet are shown in green and red, respectively, in Fig. 2.3(d) and 2.4(d). Then, the x-coordinates of the feet locations are saved. As seen in the flow diagram in Fig. 2.2, we then find the valleys in the x-coordinate values by using a peak detection algorithm by Yoder [119]. The valley points for left and right foot can be seen in Fig. 2.7(a) and 2.7(b), respectively. The number of steps taken is set to be the total number of detected valley points.

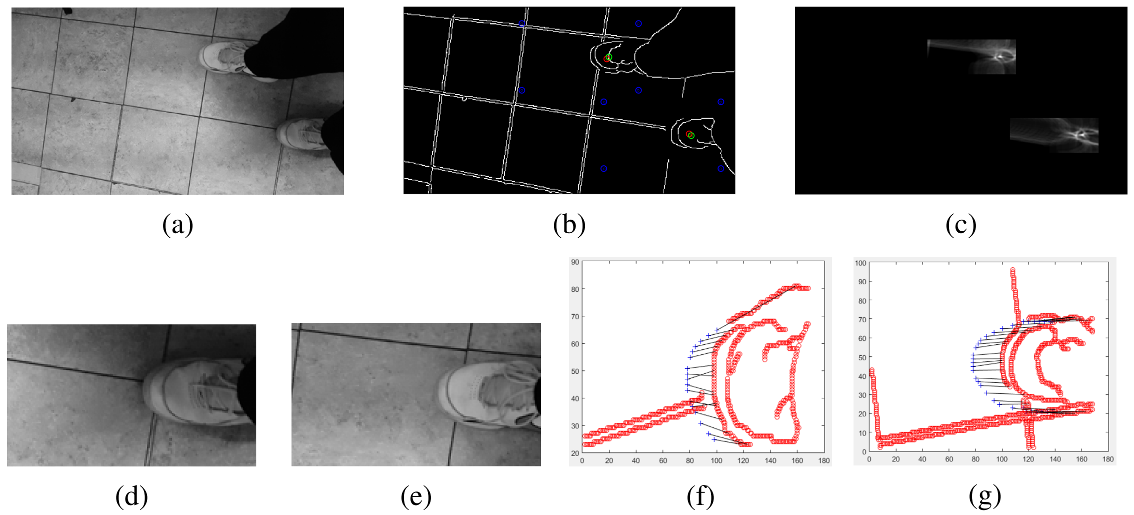


Figure 2.3: (a) Gray scale image, (b) detected edges, (c) correlation coefficient values, (d,e) feet subimages, (f,g) shape context.

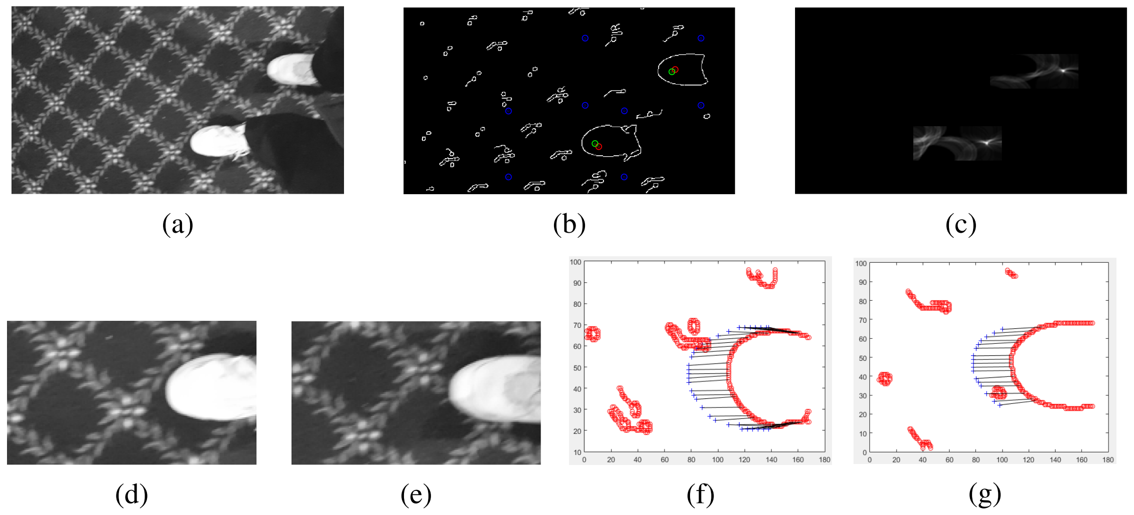


Figure 2.4: (a) Gray scale image, (b) detected edges, (c) correlation coefficient values, (d,e) feet subimages, (f,g) shape context.

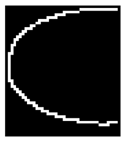


Figure 2.5: The template used for matching.

2.1.2 Computing the Total Traveled Distance

In our method, instead of using preset stride lengths and assuming equal-length steps, we calculate the length of each step by incorporating data from the camera, the accelerometer and the gravity sensor.

For better accuracy in calculating the step length, it is very important to precisely determine the start/end frames of a step. The valley locations detected as described in Section 2.1.1 may not exactly correspond to actual start/end of a step. In order to address this challenge, we incorporate information from the accelerometer. First, the peaks of the magnitude of the accelerometer data are found as shown in Fig. 2.8. This, by itself, does not solve the problem either, since accelerometer data can be noisy at times and not all the peaks in the magnitude of the accelerometer data correspond to actual steps. Thus, we combine the valley information obtained from the camera data with the information from the accelerometer data. For a valley in x-coordinates of feet locations, the closest peak, in time, of the accelerometer data is found, and the corresponding frame number is used as the beginning/end frame of a step. This approach prevents the method being affected by the noisy peaks in the accelerometer data.

After the start/end of a frame is determined, the next stage is to find the tips of each foot on those frames so that the step length can be calculated. For this, shape context [120] is applied to the output of the edge detection. For higher precision, full size images (1920×1080) of these frames are used. The detected tip locations can be seen in Fig.2.3(f) and 2.4(f).

Before computing the step length, camera rotation angle obtained from the gravity sensor is checked. This angle is used to determine if a significant change happened in the camera orientation or not. More specifically, if the rotation angle is greater than 10 degrees, which is an empirically determined threshold, then the camera geometry model is used to incorporate this change, and the image locations of extracted tip points are transformed to the world coordinate system as described below.

If the angles obtained from the gyroscope are θ' and ϕ', we use $\theta = \theta' - \frac{\pi}{2}$, $\phi = \phi' - \frac{\pi}{2}$ for the rotation of the camera along the x-axis and y-axis, respectively. Then, the image coordinates can be transformed to world coordinates by using the rotation matrix.

Since the image captured by the camera is assumed to be of a ground plane, a 2D planar transformation is used. The relationship between the camera coordinate system and the world coordinate systems is given by:

$$C = \begin{pmatrix} R_\phi R_\theta & T \end{pmatrix} \begin{pmatrix} W^T & 1 \end{pmatrix}^T \tag{2.1}$$

where

$$C = \begin{pmatrix} X_C \\ Y_C \\ Z_C \end{pmatrix}, W = \begin{pmatrix} X_W \\ Y_W \\ Z_W \end{pmatrix}, \qquad T = \begin{pmatrix} X_T \\ Y_T \\ Z_T \end{pmatrix},$$

$$R_\theta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}, \; R_\phi = \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix}.$$

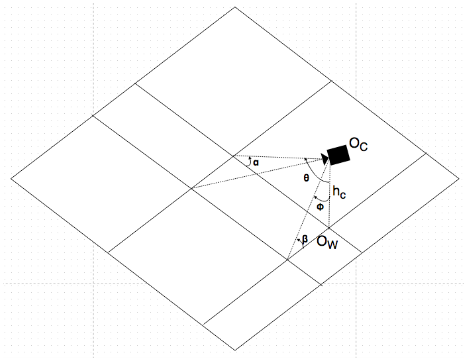


Figure 2.6: camera geometry.

This step is especially important when the camera is hold by hand.

In this case, we are using camera geometry to let camera be perpendicular to the ground with known hight H_C. Because we only need distance of tips of feet, the original point is irrelevant to the result. For convenience, we let $Z_W = 0, X_T = Y_T = 0, Z_T = -H_C$, where H_C is the camera height. Thus,

$$\begin{pmatrix} X_W \\ Y_W \\ H_C \end{pmatrix} = R_\theta^{-1} R_\phi^{-1} C.$$

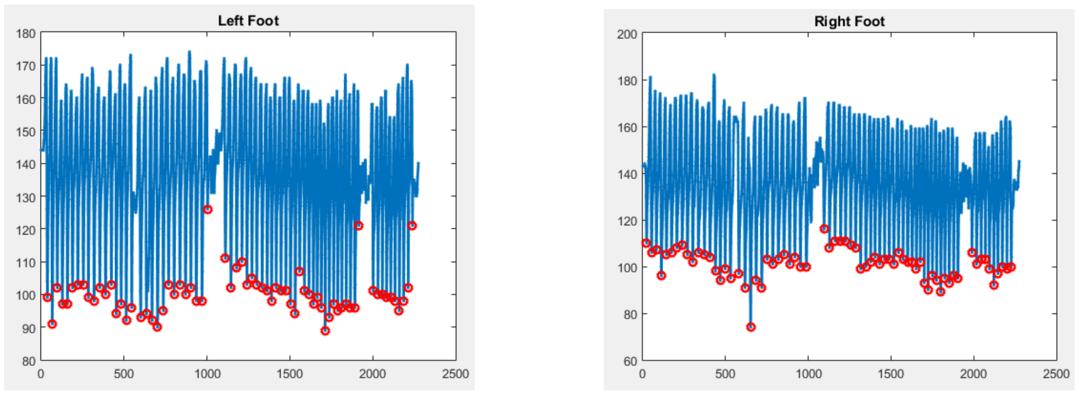
By substituting and simplifying the equations above, the transformation from image coordinates to world coordinates can be expressed as:

$$X_W = H_C \frac{x \sin\phi - f \cos\phi}{x \sin\theta \cos\phi - y \cos\theta + f \sin\theta \sin\phi},$$

$$Y_W = H_C \frac{x \cos\theta \cos\phi + y \sin\theta + f \cos\theta \sin\phi}{x \sin\theta \cos\phi - y \cos\theta + f \sin\theta \sin\phi}.$$

where f is the focal length of the camera, x and y are point location in image.

With known the height of smart phone H_C, X_W and Y_W can be calculated and then stride length is obtained by calculating distance in y direction which is along the walking direction between tips of feet.



(a) (b)

Figure 2.7: Valleys in the x-coordinates of (a) left foot and (b) right foot.

2.2 Experimental Results

In table 2.1 and 2.2, we compared method with and without camera geometry to accelerometer based cell-phone app.

Subjects	Ground Truth	Method		Front pocket-accelerometer		Backpack-accelerometer	
	No. of Steps	Counted	Error	Counted	Error	Counted	Error
Subject 1	295	307	4.07 %	352	19.32%	269	8.81 %
Subject 2	151	159	5.30 %	157	3.97%	170	12.58 %
Subject 3	356	365	2.53 %	242	32.02%	201	43.54 %
Subject 4	278	283	1.80 %	215	22.66%	166	40.29 %
Subject 5	176	184	4.55%	259	47.16%	156	11.36 %
Subject 6	153	163	6.54%	174	13.73%	213	39.22 %
Subject 7	363	374	3.03 %	280	22.87%	243	33.06 %
Subject 8	217	221	1.84 %	201	7.37%	239	10.14 %
Subject 9	230	240	4.35%	188	18.26%	205	10.87 %
Subject 10	255	264	3.53%	198	22.35%	271	6.27 %
			Avg. err 3.75%		Avg. err 20.97%		Avg. err 21.67%

Table 2.1: Step Counting Results Part-1

Subjects	Ground Truth		Method with camera geometry		Method without camera geometry		Front pocket-accelerometer		Backpack-accelerometer	
	Distance (feet)	Distance (mile)	Distance (feet)	Error	Distance (feet)	Error	Distance (mi)	Error	Distance (mi)	Error
Subject 1	344	0.065	330.6	3.90%	328.1	4.62%	0.14	115.38 %	0.11	69.23 %
Subject 2	309	0.059	300.7	2.69%	296.8	3.95%	0.06	1.69 %	0.07	18.64 %
Subject 3	335	0.063	311.2	7.10%	297.4	11.22%	0.10	58.73 %	0.08	26.98 %
Subject 4	282	0.053	273.5	3.01%	269.1	4.57%	0.08	50.94 %	0.07	32.08 %
Subject 5	294	0.056	284.0	3.40%	281.6	4.22%	0.10	78.57 %	0.06	7.14 %
Subject 6	317	0.060	307.8	2.90%	297.9	6.03%	0.07	16.67 %	0.08	33.33 %
Subject 7	302	0.057	299.2	0.93%	297.0	1.66%	0.11	92.98 %	0.10	75.44 %
Subject 8	290	0.055	276.9	4.52%	270.3	6.79%	0.08	45.45 %	0.10	81.82 %
Subject 9	316	0.060	305.9	3.20%	298.2	5.63%	0.08	33.33 %	0.08	33.33 %
Subject 10	323	0.061	312.7	3.19%	306.4	5.14%	0.08	31.15 %	0.11	80.33 %
				Avg. err 3.48%		Avg. err 5.38%		Avg. err 52.49%		Avg. err 45.83%

Table 2.2: Traveled Distance Results Part-1

In table 2.3 and 2.4, we compared method without camera geometry to accelerometer

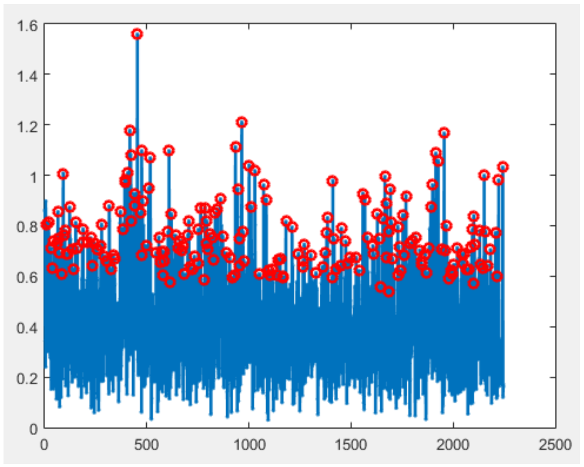


Figure 2.8: Magnitude of the accelerometer data.

based cell-phone app.

Subjects	Ground Truth	Method		Front pocket-accelerometer		Backpack-accelerometer	
	No. of Steps	Counted	Error	Counted	Error	Counted	Error
Subject 1	258	266	3.10 %	212	17.83%	220	14.73 %
Subject 2	257	262	1.95 %	238	7.39%	294	14.40 %
Subject 3	254	263	3.54 %	199	21.65%	185	27.17 %
Subject 4	330	334	1.21 %	183	44.55%	230	30.30 %
Subject 5	180	171	5.00%	240	33.33%	193	7.22 %
Subject 6	282	274	2.84%	84	70.21%	68	75.89 %
Subject 7	227	236	3.96 %	225	0.88%	213	6.17 %
Subject 8	277	280	1.08 %	207	25.27%	271	2.17 %
Subject 9	300	289	3.67%	57	81.00%	195	35.00 %
Subject 10	284	279	1.76%	216	23.94%	245	13.73 %
Subject 11	207	198	4.35%	160	22.71%	199	3.86 %
Subject 12	228	221	3.07%	105	53.95%	100	56.14 %
Subject 13	244	237	2.87%	208	14.75%	252	3.28 %
Subject 14	272	279	2.57%	259	4.78%	75	72.43 %
Subject 15	238	230	3.36%	174	26.89%	225	5.46 %
			Avg. err 2.96%		Avg. err 29.94%		Avg. err 24.53%

Table 2.3: Step Counting Results Part-2

Subjects	Ground Truth		Method		Front pocket-accelerometer		Backpack-accelerometer	
	Distance (feet)	Distance (mile)	Distance (feet)	Error	Distance (mi)	Error	Distance (mi)	Error
Subject 1	301.2	0.057	298.10	1.03%	0.08	40.35 %	0.08	40.35 %
Subject 2	300.6	0.057	299.29	0.44%	0.09	57.89 %	0.09	57.89 %
Subject 3	300.3	0.057	295.14	1.72%	0.08	40.35 %	0.07	22.81 %
Subject 4	335.0	0.063	321.76	3.95%	0.06	4.76 %	0.08	26.98 %
Subject 5	336.7	0.064	304.58	9.54%	0.09	40.63 %	0.08	25.00 %
Subject 6	330.0	0.063	307.23	6.90%	0.03	52.38%	0.03	52.38%
Subject 7	327.4	0.062	312.67	4.50%	0.09	45.16 %	0.09	45.16 %
Subject 8	325.8	0.062	316.03	3.00%	0.08	29.03 %	0.11	77.42 %
Subject 9	330.9	0.063	318.33	3.80%	0.02	68.25%	0.08	26.98 %
Subject 10	324.8	0.062	307.26	5.40%	0.09	45.16%	0.10	61.29 %
Subject 11	319.9	0.061	308.51	3.56%	0.06	1.64%	0.08	31.15 %
Subject 12	323.7	0.061	305.89	5.50%	0.04	34.43%	0.04	34.43 %
Subject 13	330.2	0.063	315.57	4.43%	0.08	26.98%	0.10	58.73 %
Subject 14	332.1	0.063	315.09	5.12%	0.10	58.73%	0.03	52.38 %
Subject 15	311.6	0.059	304.53	2.27%	0.07	18.64%	0.09	52.54 %
				Avg. err 4.08%		Avg. err 37.63%		Avg. err 44.37%

Table 2.4: Traveled Distance Results Part-2

In table 2.5 and 2.6, we compared method without camera geometry to accelerometer based cell-phone app, in case objects stop and start walk again several times during experiments.

In table 2.7 and 2.8, we compared method with camera geometry while objects wearing a cell-phone holder to accelerometer based cell-phone app.

In fig. 2.9, we prepared the case multiple feet appear in camera while walking. The results show that due to local template matching, the extra foot does not have influence on

Subjects	Ground Truth	Method		Front pocket-accelerometer		Backpack-accelerometer	
	Steps	Counted	Error	Counted	Error	Counted	Error
Subject 1	275	297	8.00%	340	23.64%	326	18.55%
Subject 2	254	242	4.72%	207	18.50%	299	17.72%
Subject 3	203	191	5.91%	204	0.49%	261	28.57%
Subject 4	220	213	3.18%	217	1.36%	67	69.55%
Subject 5	233	225	3.43%	76	67.38%	236	1.29%
Subject 6	216	206	4.63%	180	16.67%	253	17.13%
Subject 7	289	283	2.08%	209	27.68%	164	43.25%
Subject 8	240	238	0.83%	91	62.08%	157	34.58%
Subject 9	277	269	2.89%	243	12.27%	260	6.14%
Subject 10	232	220	5.17%	205	11.64%	199	14.22%
			Avg. err 4.09%		Avg. err 24.17%		Avg. err 25.10%

Table 2.5: Step count results for the experiment involving stopping and starting during walks.

Subjects	Ground Truth		Method		Front pocket-accelerometer		Backpack-accelerometer	
	Distance(feet)	Distance (mile)	Distance (feet)	Error	Distance (mi)	Error	Distance (mi)	Error
Subject 1	325.5	0.062	344.7	5.90%	0.14	125.81%	0.13	109.68%
Subject 2	323.9	0.061	318.7	1.61%	0.08	31.15%	0.12	96.72 %
Subject 3	318.9	0.060	303.6	4.80%	0.08	33.33%	0.12	100.00 %
Subject 4	320.2	0.061	311.2	2.81%	0.09	47.54%	0.03	50.82 %
Subject 5	330.7	0.063	314.8	4.81%	0.03	52.38%	0.09	42.86 %
Subject 6	319.4	0.060	373.3	16.88%	0.07	16.67%	0.10	66.67 %
Subject 7	334.1	0.063	316.1	5.39%	0.09	42.86%	0.07	11.11 %
Subject 8	324.5	0.061	313.8	3.30%	0.04	34.43%	0.06	1.64 %
Subject 9	328.3	0.062	312.2	4.90%	0.10	61.29%	0.11	77.42 %
Subject 10	311.6	0.059	295.4	5.20%	0.08	35.59%	0.08	35.59 %
				Avg. err 5.56%		Avg. err 48.10%		Avg. err 59.25%

Table 2.6: Traveled distance results for the experiment involving stopping and starting during walks.

the algorithm.

2.3 Conclusion

Most of the available devices and approaches for step counting rely only on accelerometer data, and thus are prone to over-counting. We have presented an autonomous and robust method for counting footsteps, and tracking and calculating stride length by using both accelerometer and camera data from smart phones or a Google$^{\text{TM}}$glass. To provide higher precision, instead of using a preset step and/or stride length, the presented method calculates the distance traveled with each step by using the camera data. The presented method has been compared with the commonly-used accelerometer-based step counter applications (apps). The results show that the presented method provides a significant increase in accu-

Subjects	Ground Truth	Method		Front pocket-accelerometer		Backpack-accelerometer	
	No. of Steps	Counted	Error	Counted	Error	Counted	Error
Subject 1	1830	1749	4.43 %	1165	36.34%	1373	24.97 %
Subject 2	2023	1987	1.78 %	1349	33.32%	996	50.77 %
Subject 3	1996	1916	4.01 %	1433	28.21%	1505	24.60 %
Subject 4	2011	1936	3.73 %	2679	33.22%	916	54.45 %
Subject 5	1759	1707	2.96%	778	55.77%	1327	24.56 %
Subject 6	1700	1635	3.82%	1521	10.53%	1299	23.59 %
Subject 7	1861	1809	2.79 %	2300	23.59%	1090	41.43 %
Subject 8	1728	1631	5.61 %	1517	12.21%	1444	16.44 %
Subject 9	1838	1778	3.26%	996	45.81%	1919	4.41 %
Subject 10	1830	1764	3.61%	1209	33.93%	1363	25.52 %
Subject 11	1949	1880	3.54 %	2596	33.20%	860	55.87 %
Subject 12	2067	1953	5.52 %	1247	39.67%	695	66.38 %
Subject 13	1899	1807	4.84 %	886	53.34%	2308	21.54 %
Subject 14	1818	1734	4.62 %	1541	15.24%	1404	22.77 %
Subject 15	1800	1760	2.22%	1398	22.33%	2137	18.72 %
Subject 16	2044	1959	4.16%	692	66.14%	1002	50.98 %
Subject 17	1703	1608	5.58 %	1416	16.85%	1923	12.92 %
Subject 18	1895	1803	4.85 %	1478	22.01%	1384	26.97 %
Subject 19	1767	1700	3.79%	1503	14.94%	1993	12.79 %
Subject 20	1906	1866	2.10%	2640	38.51%	2670	40.08 %
			Avg. err 3.86%		Avg. err 31.76%		Avg. err 30.98%

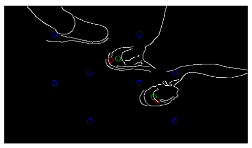
Table 2.7: Step Counting Results From holder

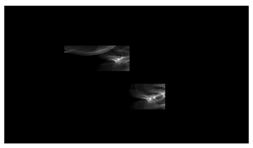
Subjects	Ground Truth		Method with camera geometry		Method without camera geometry		Front pocket-accelerometer		Backpack-accelerometer	
	Distance (feet)	Distance (mile)	Distance (feet)	Error	Distance (feet)	Error	Distance (mi)	Error	Distance (mi)	Error
Subject 1	2000	0.38	1942.6	2.87%	328.1	4.62%	0.48	25.13 %	0.56	47.48 %
Subject 2	2000	0.38	1978.0	1.10%	296.8	3.95%	0.55	44.74 %	0.41	7.89 %
Subject 3	2000	0.38	1930.2	3.49%	297.4	11.22%	0.58	53.92 %	0.61	61.65 %
Subject 4	2000	0.38	1986.8	0.66%	269.1	4.57%	1.09	187.76 %	0.37	1.61 %
Subject 5	2000	0.38	1926.0	3.70%	281.6	4.22%	0.32	16.43 %	0.54	42.53 %
Subject 6	2000	0.38	1894.1	5.30%	297.9	6.03%	0.62	63.37 %	0.53	39.53 %
Subject 7	2000	0.38	1912.2	4.39%	297.0	1.66%	0.94	147.05 %	0.44	17.08 %
Subject 8	2000	0.38	1862.5	6.88%	270.3	6.79%	0.62	62.94 %	0.59	55.10 %
Subject 9	2000	0.38	1936.0	3.20%	298.2	5.63%	0.41	6.98 %	0.78	106.12%
Subject 10	2000	0.38	1940.3	2.99%	306.4	5.14%	0.49	29.86 %	0.56	46.40 %
Subject 11	2000	0.38	1932.0	3.40%	328.1	4.62%	1.06	178.84 %	0.35	7.63 %
Subject 12	2000	0.38	1910.7	4.47%	296.8	3.95%	0.51	33.94 %	0.28	25.35 %
Subject 13	2000	0.38	1932.1	3.40%	297.4	11.22%	0.36	4.83 %	0.94	147.91%
Subject 14	2000	0.38	1927.9	3.61%	269.1	4.57%	0.63	65.52 %	0.57	50.81 %
Subject 15	2000	0.38	1922.6	3.87%	281.6	4.22%	0.57	50.16 %	0.87	128.54%
Subject 16	2000	0.38	1948.2	2.59%	297.9	6.03%	0.28	25.67 %	0.41	7.63 %
Subject 17	2000	0.38	1847.9	7.61%	297.0	1.66%	0.58	52.09 %	0.78	106.55%
Subject 18	2000	0.38	1887.8	5.61%	270.3	6.79%	0.60	58.75 %	0.56	48.66 %
Subject 19	2000	0.38	1874.1	6.30%	298.2	5.63%	0.61	61.44 %	0.81	114.07%
Subject 20	2000	0.38	1939.4	3.03%	306.4	5.14%	1.08	183.57 %	1.09	186.79%
				Avg. err 3.92%		Avg. err 4.98%		Avg. err 67.65%		Avg. err 62.47%

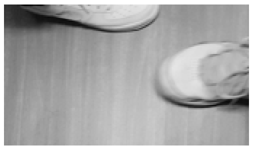
Table 2.8: Traveled Distance Results from holder

racy, and has the lowest average error rate both in number of steps taken and the distance traveled compared to commercially available, accelerometer-based step counters and apps.

(a)


(b)


(c)


(d)

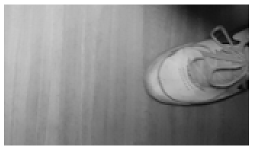

(e)

Figure 2.9: (a) Gray scale image, (b) detected edges, (c) correlation coefficient values, (d,e) feet subimages, (f,g) shape context.

Chapter 3

Human Activity Classification from Wearable Devices with Cameras

In this chapter, we use accelerometer and camera data from a smart phone, and focus on the type of activities, which cannot be classified by just accelerometer data, and require detection of objects in the environment from camera data. Without loss of generality, we present experiments on differentiating between sitting on a sofa, sitting on a chair, and walking through doorways. Only accelerometer data can be enough to detect walking, but it is not enough to detect that the person is walking through a doorway, and thus changing rooms. Similarly, camera data becomes necessary to detect whether the person sat on a sofa or a chair. In our proposed approach, objects in images are detected by using an Aggregate Channel Features (ACF)-based detector. Multi-class SVMs are trained to classify motion-related activities, and detect approaching different objects. Overall precision and recall rates of 95% and 89% are achieved, respectively.

The rest of this chapter is organized as follows: The proposed approach is described in Section 3.1. Experimental results are presented in Section 3.2, and the chapter is concluded in Section 3.3.

3.1 Methodology

Our proposed method employs both camera and accelerometer data obtained from a smart phone. A flow diagram of the proposed approach is provided in Fig. 3.1. Images from camera are used to (i) compute optical flow vectors, and (ii) perform object detection via an ACF-based detector [121]. Different features are extracted from the accelerometer data to obtain a 14D vector. Then, a 3D vector is obtained based on the y component of the optical flow vectors, and concatenated to the 14D vector to obtain a 17D feature vector. A multi-class SVM is trained and employed to differentiate between motion-related activities of walking, sitting from standing position, standing from sitting position, and being immobile.

The ACF-based detector is trained to detect chairs, sofas and doors/doorways. It provides bounding boxes around the detected objects. The features obtained from these bounding boxes together with the label describing the detected object type are saved as a vector. These vectors are used to train another multi-class SVM to detect approaching a sofa, chair or door. Then, a Hidden Markov Model (HMM) is built to be used in the final stage to detect whether the person carrying the smart phone is sitting on a chair, sitting on a sofa or walking through a door. More details about these steps are described in the following subsections.

3.1.1 Feature extraction

Accelerometer data

A segment length of 10 frames is used for accelerometer data. Mean and mean of the absolute values are used to measure the average acceleration. In addition, variance and movement intensity are obtained from the accelerometer data. Movement intensity at time t is $\sqrt{a_x(t)^2 + a_y(t)^2 + a_z(t)^2}$, where $a_x(t)$, $a_y(t)$ and $a_z(t)$ are the accelerations along the x, y and z axes, respectively, at time t. Accelerometer data is transformed to the frequency domain by using FFT, and main frequency and energy are obtained as described in [53].

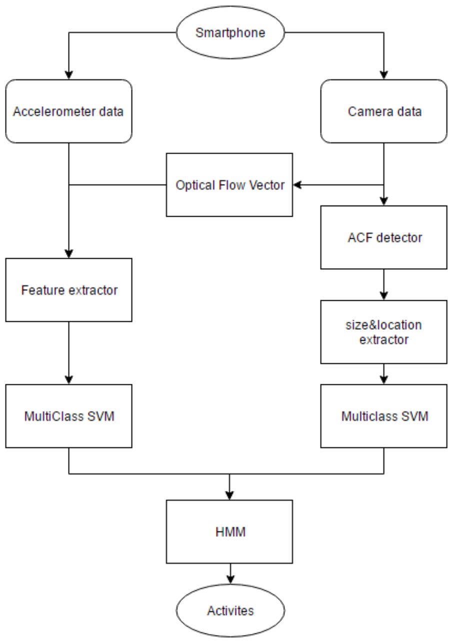


Figure 3.1: The flow diagram of the proposed approach.

As seen in detail in Table 3.1, a 14D vector is obtained from the features obtained from the accelerometer data.

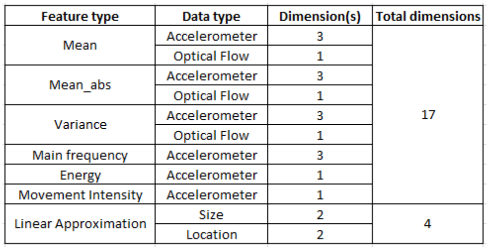
Camera data

As for camera data, dense optical flow vectors are extracted from each frame. In our approach, rather than using both components of optical flow vectors, we only use the vertical component for robustness. When both components are used, other people, passing in front of the person wearing the camera, cause distractions.

First, an image is divided into nine regions. In each region, the average of vertical components of flow vectors is computed, and the maximum value M_{of} among the 9 average values is obtained. The mean, mean of absolute value and variance of M_{of} are calculated as features, in a segment length of 5 frames, to get a 3D vector. This vector is concatenated with the 14D vector from the accelerometer data to obtain a 17D feature vector to be used in the classification of the activities of walking, sitting from standing position, standing from sitting position, and being immobile.

In addition to the vertical optical flow components, other features are also extracted from camera data. More specifically, an ACF-based detector is trained to detect sofas, chairs and doors/doorways in images. The output is a bounding box around the detected object. Areas and the y-coordinates of the centroid of the bounding boxes are extracted for 5 consecutive frames. A linear model is used to fit a line to these values over five frames, and the parameters of these lines are saved to obtain a 4D vector. The label describing the detected object type is added to this to obtain a 5D vector to be used in the classification of approaching a sofa, chair or door.

Table 3.1: Description of different extracted features.



Feature type	Data type	Dimension(s)	Total dimensions
Mean	Accelerometer	3	17
	Optical Flow	1	
Mean_abs	Accelerometer	3	
	Optical Flow	1	
Variance	Accelerometer	3	
	Optical Flow	1	
Main frequency	Accelerometer	3	
Energy	Accelerometer	1	
Movement Intensity	Accelerometer	1	
Linear Approximation	Size	2	4
	Location	2	

3.1.2 Classification

The 17D feature vector obtained from accelerometer data, and vertical components of optical flow vectors is used to train a multi-class SVM to classify activities of walking, sitting from standing position, standing from sitting position, and being immobile.

The 5D vector obtained from the outputs of ACF-based detector is used to train another multi-class SVM to classify activities of approaching a chair, a sofa or a door. In addition, an HMM is built involving four states, and 10 observations as seen in Fig. 3.2. The states are (1) sitting on chair, (2) sitting on sofa, (3) walking through door, and (4) none. Both classification results from the multi-class SVMs are combined to be used as the observation in the HMM.

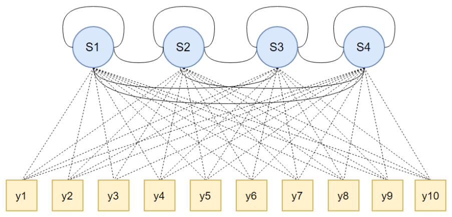


Figure 3.2: The built HMM.

3.1.3 Training

For training the two multi-class SVMs and the HMM, a 30min video is captured from a subject wearing the smart phone as seen in Fig. 3.3. The subject performs a total of 193 activities consisting of sitting on a chair, sitting on a sofa, and walking through doors.

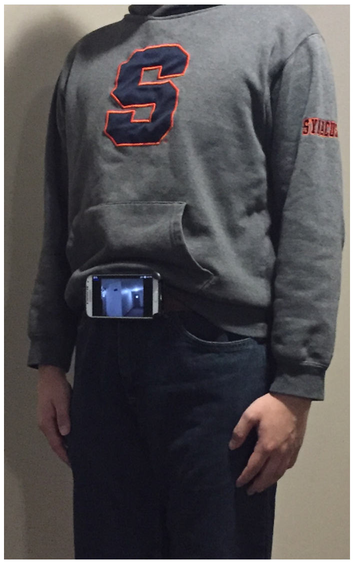


Figure 3.3: Experimental setup.

3.2 Experimental Results

Experiments have been performed with seven different subjects in an indoor environment. During a period of about 10 min, each subject sits on a chair, sits on a sofa, and walks through doors several times. More specifically, there are 100, 90, 80, 101, 88, 89 and 120 activities performed by the seven subjects. The smart phone is carried on a belt around the waist, and the camera faces forward as seen in Fig. 3.3. The captured image size is 640×480 pixels. The precision and recall rates are calculated by using the following definitions:

$$\text{Precision} = \frac{tp}{tp + fp}, \qquad \text{Recall} = \frac{tp}{tp + fn},$$

where tp, fp and fn refer to the number of true positives, false positives and false negatives, respectively.

The overall precision and recall values, across all seven subjects, for all three activity types are plotted in Fig. 3.4. The precision and recall values for each subject can be seen in Fig. 3.5. The variation of the precision and recall values can be seen in Fig. 3.6(a) and 3.6(b), respectively. Example images from different test videos showing the detection of chairs, sofas and doorways by the ACF-based detector are shown in Figures 3.7 and 3.8.

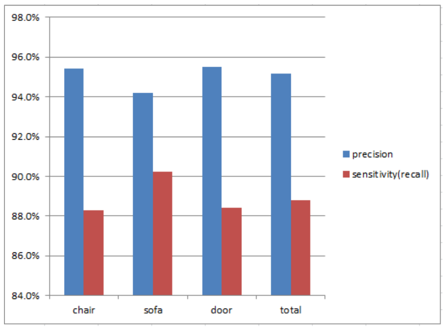


Figure 3.4: The overall precision and recall values across all subjects.

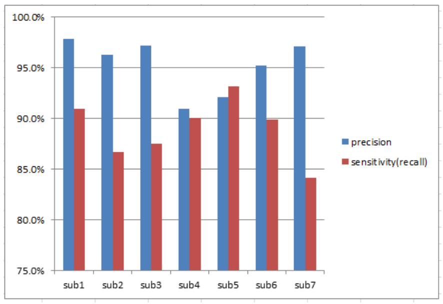
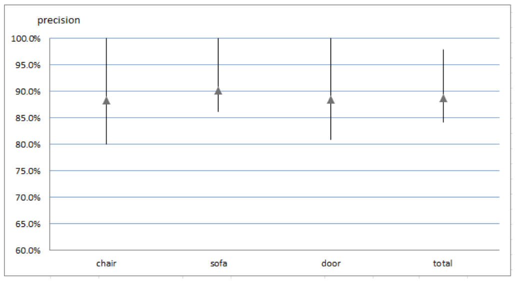
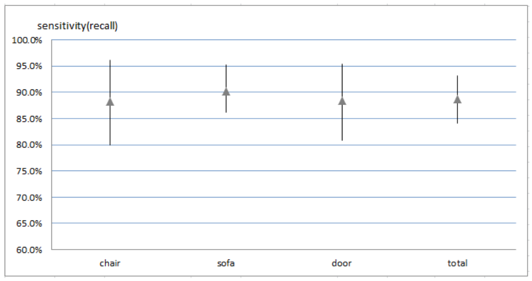


Figure 3.5: The precision and recall values for each subject.

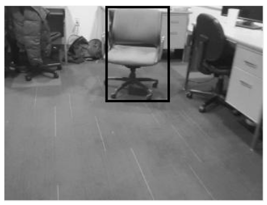


(a)

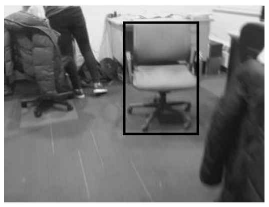


(b)

Figure 3.6: Variation of the (a) precision, and (b) recall values.

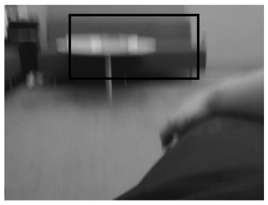


(a)

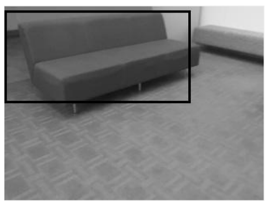


(b)

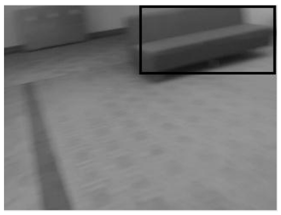
(c)



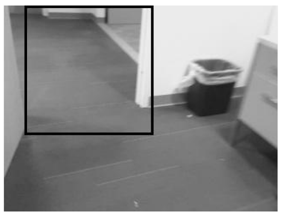
(d)



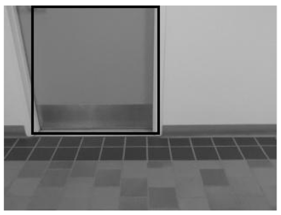
(e)



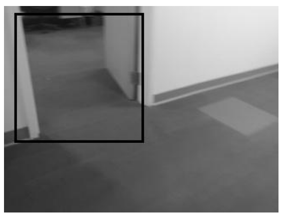
(f)



(g)



(h)



(i)

Figure 3.7: Example images from a test video showing the detection of chairs (a-c), sofas (d-f) and doorways (g-i).

3.3 Conclusion

In this chapter, we have presented a new approach, using accelerometer and ego-vision data from a smart phone, for activity classification. We have focused on types of activities that cannot be differentiated by just accelerometer data, since they are more complex, and require context and detection of objects in the environment from camera data. Without loss

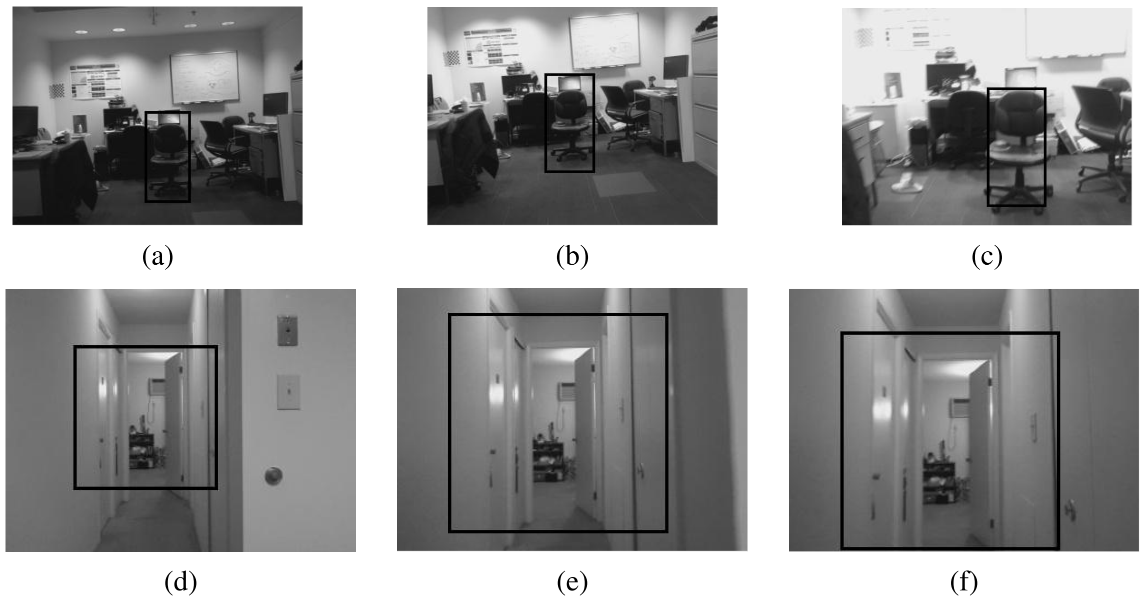


Figure 3.8: Example images from another test video showing the detection of chairs (a-c) and doorways (d-f).

of generality, we have presented results on differentiating between sitting on a sofa, sitting on a chair, and walking through a door. The activity types can be increased by detecting different types of objects in the scene, and modifying the HMM. The proposed method achieves overall precision and recall rates of 95% and 89%, respectively.

Chapter 4

Human Activity Classification Incorporating Egocentric Video And Inertial Measurement Unit Data

This chapter studies a robust and autonomous activity classification method that leverages data from multiple wearable sensor modalities to differentiate between activities, which are similar in nature, with a level of accuracy that would be impossible by each sensor alone. We employ egocentric camera data together with the data from four IMU sensors on body, provided in the CMU Multi-Modal Activity (CMU-MMAC) Database [56]. In this chapter, we propose a new model architecture, which incorporates Capsule Networks (CapsNets) [122] for feature extraction. Our method, instead of using a single CapsNet, employs multiple CapsNets for consecutive images, and then uses a convolutional LSTM to build a recurrent CapsNet (RecCapsNet). The LSTM framework is employed both on IMU data and egocentric camera data to capture the temporal aspect of actions. Without loss of generality, the following classes have been extracted from the videos in the CMU-MMAC dataset: cracking eggs, beating eggs, pouring oil, pouring cake mix/flour, stirring in a bowl and using fridge. Example images from the ego-vision camera can be seen in Fig. 4.2. The method resulted in very promising results, achieving overall recall and precision rates of 86.19% and 85.75%, respectively. We also present results of using each sensor modality by itself, which show that the approach provides 19.47% and 39.34% increase in accuracy compared to using only egocentric camera data and only IMU data, respectively. Since this is an example of fine-grained classification, using multiple modalities provides a significant increase in performance compared to single-modality results. We also performed a comparison of using CapsNets versus VGG16 features, and detailed results are provided in Sec.r̃efsec:exp. This approach can readily be extended to classify more activity types, by detecting different types of objects in the scene.

4.1 Methodology

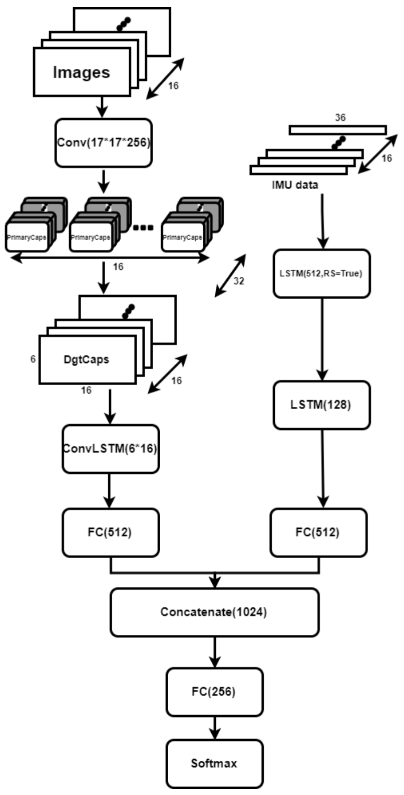


Figure 4.1: Details of the architecture.

We propose a new model architecture, shown in Fig. 4.1, which is based on CapsNets and LSTM. The inputs to the network are egocentric camera images and IMU data. Capsule Network has been introduced by Sabour et al. [122] to explore spatial relationships between features, and it achieved state-of-the-art performance on the MNIST database. A capsule is a group of neurons, and capsule outputs are in vector form. The length of the vector corresponds to the detection probability of a feature, and the direction of the vector corresponds to the state of the feature. Dynamic routing is used to determine the destination of a capsule's output. In this chapter, instead of using a single image with the original CapsNet, we propose a Recurrent CapsNet (RecCapsNet), which takes a sequence of images as input, and then implement a 2D Convolutional LSTM (convLSTM) [123] layer to capture the temporal aspect.

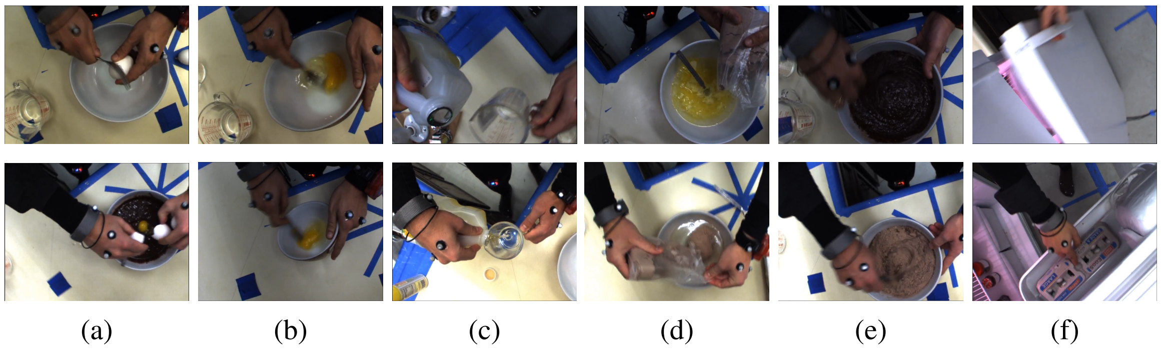


Figure 4.2: Example images of CMU-MMAC dataset. Columns (a) cracking egg, (b) beating eggs, (c) pouring oil, (d) pouring a bag of cake mix, (e) stirring in a bowl, (f) using fridge.

First an image is sent to a ReLU convolutional layer containing 256 convolution kernels with a stride of 1 and ReLU activation. Then, the output ($20 \times 20 \times 256$) is sent to 32 primary capsules. This is done for 16 consecutive images in a video clip with 50% overlap. Each primary capsule has 8 convolutional units. The next layer DigitCaps is 6×16, having one 16 dimensional capsule for each of the six classes that we have. Since we are processing 16 consecutive frames, the output after DigitCaps layer is $6 \times 16 \times 16$ (Fig. 4.1). We then apply a ConvLSTM layer, followed by a fully connected (FC) layer for the analysis of video data.

On the other hand, IMU data (for again 16 consecutive time frames) is fed into a two-layer LSTM model followed by a FC layer to extract feature vectors.

For the decoder part, we apply 16 sub-decoders to each image frame. Each sub-decoder has the same structure with the decoder of the original CapsNet except the sigmoid output is 1296 (36×36). Given the ground truth label, the decoder regenerates a 16 frame image sequence which has the same size with the image input. The description of different input types, and details of the network architecture will be provided below.

4.1.1 Input Data

The network architecture takes an image sequence of F consecutive frames and IMU data with the same length of F. In our experiments, F is 16. The image frame is 36×36, and the IMU data vector has 36 components concatenated from the four IMU sensors in the CMU-MMAC dataset. Each IMU sensor contributes nine entries from accelerometer, gyroscope and magnetometer measurements.

4.1.2 Processing of IMU Data Sequence

As mentioned above, the IMU data features are extracted by a 2-layer LSTM. The first layer has 512 units with return sequence, and the second layer has 128 units. For the first layer LSTM for the IMU data, $D = 512$, $N = 36$, and $t \in [1, 16]$. The output of this layer is $[O_{t=1}, O_{t=2}, ..., O_{t=16}]$. For the second layer LSTM, $D = 128$, $N = 512$, and $t \in [1, 16]$. The output of this layer is $O_{t=16}$. The LSTM layers are followed by a fully connected layer having 512 neurons. The fully connected layer is added at this point in order to make the IMU feature vector length the same as the length of image feature vectors, and also to let both vectors have equal weights after concatenation.

4.1.3 Processing of Egocentric Video Data

Recurrent neural networks have been successfully used with time series data, and for human activity detection using IMU sensors. In our approach, we use CapsNets on consecutive image frames, and apply ConvLSTM to capsule vectors to propose a new RecCapsNet. The Digit Capsule outputs are fed into a ConvLSTM to extract temporal features. This is followed by a FC layer to have the features equally weighted as IMU data features. The output of each of the ConvLSTM is of length 6×16. The FC layer has 512 neurons.

4.1.4 Classification

Feature vector outputs for IMU and video data are concatenated to form a vector of size 1024, which is then fed to a FC layer with 256 neurons. This is followed by a softmax classifier. The output of the model is the confidence score for each class proposal.

4.2 Experimental Results

For the experiments, we used the CMU-MMAC dataset, which contains data from multi-modal sensors, including camera and IMU data, monitoring activity of subjects performing cooking and food preparation tasks. In our experiments, we used the video data from the egocentric camera view, and the wired IMU data. We down-sampled the IMU data to make the measuring frequency the same with that of egocentric camera (30 Hz). Images are resized to a size of 36×36, and processed in groups of 16 frames with 50% overlapping. Then, we aligned/synchronized the IMU with camera data. Without loss of generality, the following classes were extracted from the videos: cracking eggs, beating eggs, pouring oil, pouring a bag of cake mix/flour, stirring in a bowl and using fridge. Example images from the ego-vision camera can be seen in Fig. 4.2. The system was trained on 8 videos, and tested on activities extracted from one long video.

We also performed a comparison of our method of using RecCapsNets versus using VGG16 features. In other words, we still used both camera and IMU data, but instead of employing RecCapsNet, we extracted image features from 16 consecutive image frames, via convolutional layers of CNN-based VGG16 [124] without the top layers. The results are presented in Table 4.1. As can be seen, using our RecCapsNet achieves an average accuracy of 84.4%, and increases accuracy by 18.8% compared to using VGG16 features.

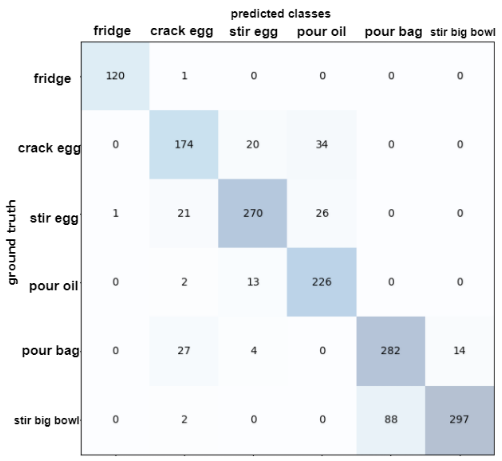
Moreover, to provide comparison and show the improvement obtained by using multiple sensor modalities, we also obtained results by using each sensor modality by itself, namely by using only IMU data and only camera data. As can be seen in Table 4.1, the approach provides 39.34%, 25.62% and 19.47% increase in accuracy compared to using only IMU data, only egocentric camera data with VGG16 features and only egocentric camera data with CapsNet features, respectively. Since this is an example of fine-grained classification, using multiple modalities provides a significant increase in performance compared to single-modality results.

Spriggs et al. [54] and Soran et al. [125] also used the CMU-MMAC dataset, and reported accuracies of 57.8% and 54.62%, respectively. However, a direct comparison would not be commensurate, since they either employ hand-crafted features and different sets of sensor modalities, including static cameras [125], or the frame and class annotations are different.

Sensor Modality	Method	Accuracy
IMU only	LSTM	45.06%
Camera only	VGG16	58.78%
	CapsNet	64.93%
Camera and IMU	VGG16 & LSTM	65.60%
	RecCapsNet & LSTM	**84.40%**

Table 4.1: Accuracy rates for different modalities and methods

The confusion matrix obtained with the method is provided in Fig. 4.3, which also includes the number of instances of each action. For instance, in the test video, there were 121 instances of 16-frame video clips of using the fridge, and 387 instances of 16-frame video clips of stirring in a bowl. The recall and precision values for each of the six activity classes are shown in Fig. 4.4 and 4.5, respectively. It is not surprising that "using the fridge" class has the highest precision and recall among the others, since it is the furthest in the activity space from the other activities of dealing with ingredients and bowls. “Stirring in a bowl” is sometimes confused by “pouring a bag of cake mix/flour”, since stirring comes after pouring, and images seen by the camera are similar for these actions.



ground truth \ predicted classes	fridge	crack egg	stir egg	pour oil	pour bag	stir big bowl
fridge	120	1	0	0	0	0
crack egg	0	174	20	34	0	0
stir egg	1	21	270	26	0	0
pour oil	0	2	13	226	0	0
pour bag	0	27	4	0	282	14
stir big bowl	0	2	0	0	88	297

Figure 4.3: The confusion matrix showing the actual versus predicted classes together with the number of instances of each action.

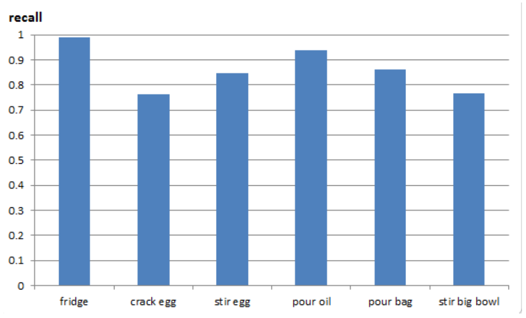


Figure 4.4: The recall values for the activity classes.

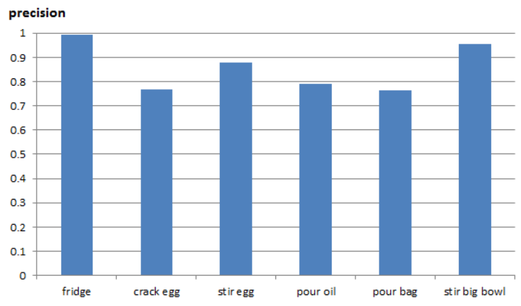


Figure 4.5: The precision values for the activity classes.

4.3 Conclusion

We have presented a robust and autonomous activity classification method that leverages data from multiple sensor modalities to differentiate between activities, which are similar in nature, with a level of accuracy that would be impossible by each sensor alone. The different modalities are egocentric video and IMU sensor data from wearable devices. We have a new model architecture, which incorporates Capsule Networks (CapsNets) for feature extraction. Instead of using a single CapsNet, multiple CapsNets are employed for consecutive images, and then a convolutional LSTM is used to build a recurrent CapsNet. The LSTM framework is employed both on IMU data and egocentric camera data to capture the temporal aspect of actions. Without loss of generality, results have been presented for classifying activities of cracking eggs, beating eggs, pouring oil, pour bag of cake mix/flour, stirring in a bowl and using fridge. The method have provided promising results achieving overall recall and precision rates of 86.19% and 85.75%, respectively. This approach can be readily extended to classify more activity types, by detecting different types of objects in the scene.

Chapter 5

Autonomous Human Activity Classification from Wearable Multi-Modal Sensors

This chapter studies the usage of egocentric video, IMU data and recurrent Capsule Networks for activity classification. The work proposed in this chapter is different and improved compared to our previous work [126] in multiple ways including the following: (i) for the work in [126], we used manually set values for all the network parameters. In contrast, in this chapter, we propose a GA-based approach to autonomously and systematically set various network parameters, rather than using manual and empirical settings; (ii) the experiments in [126] were performed for only 6-label classification. In this chapter, we provide a much more comprehensive evaluation by performing classification for 9 as well as 26 activities; (iii) in this chapter, we provide a more detailed description of The method and a comparison between the performances obtained with the manually preset network parameters, and the parameters determined by our proposed GA-based approach.

Experiments have been performed on the CMU-MMAC dataset to perform 9- and 26-label classification, and The method, using autonomously set network parameters, has provided very promising results, achieving overall accuracies of 86.6% and 77.2%, respectively. We also used each sensor modality alone, and obtained their individual accuracies, showing that The approach, combining both modalities, provides increased accuracy compared to using only egovision data and only IMU data.

The rest of this chapter is organized as follows: The approach is described in detail in Section 5.1. Experimental results are presented in Section 5.2, and the chapter is concluded in Section 5.3.

5.1 Methodology

We present a new model architecture to process first-person, also known as egocentric, images and IMU data. The architecture can be seen in Fig. 5.1. It is composed of our proposed recurrent CapsNet (for processing images), an LSTM network (for processing IMU data), and fully connected layers. In addition, we also propose and apply a GA-based approach to autonomously and simultaneously optimize multiple parameters of our network architecture. These parameters are shown in parentheses with red color in Fig. 5.1. For instance, the parameters for the fully connected layers and the primary capsules are examples of the parameters autonomously set by our proposed approach. More details about these parameters will be provided in Sec. 5.1.1.

Sabour et al. [122] introduced the Capsule Networks (CapsNets) to explore spatial relationships between features, and reported state-of-the-art performance on the MNIST database. CapsNets [122] were used for image classification on individual images, whereas our goal is to perform fine-grained activity classification from video data. Thus, in this chapter, instead of using a single image with the original CapsNet, we propose a Recurrent CapsNet (RecCapsNet), which takes a sequence of images as input. We implement a 2D Convolutional LSTM (convLSTM) [123] layer to extract features and capture the temporal aspect. For robustness, we use multiple digit/class layers instead of using only a single digit layer as was done in [122]. In order to prevent gradient vanishing, we remove the squash function for digit layers and implement ReLu activation function instead.

As seen in Fig. 5.1, 16 consecutive images are passed through a 2D convolutional layer separately. The size of each input image is 36×36. Then, the output for each image is sent to multiple primary capsules, the number of which is determined by our GA. When 16 consecutive images are formed, 50% overlap is used throughout the video. The number of convolutional units for each primary capsule is also determined by the GA. The output from the primary capsule layer is then sent through two digit/class layers, whose parameters are set by the GA. We then apply a Convolutional LSTM layer, followed by a fully connected (FC) layer, for the analysis of the egocentric video data.

For the decoder part, we apply 16 sub-decoders to each image frame. Each sub-decoder has the same structure with the decoder of the original CapsNet except the sigmoid output is 1296 (36×36). In other words, the decoders are implemented to generate the same size as the input images. Given the ground truth label, the decoder regenerates a 16 frame image sequence which has the same size as the image input.

As for the IMU data, similar to the images, data from 16 consecutive time frames are used. Each of the 16 IMU data vectors has 36 components obtained by concatenating data from the four IMU sensors. Each IMU sensor contributes nine entries from accelerometer, gyroscope and magnetometer measurements. The time stamps are provided for camera and IMU data in the CMU-MMAC dataset. To align the camera and IMU data, for a given camera image, the IMU time stamp that is closest to the camera time stamp is found. The IMU data is fed into an LSTM model to extract feature vectors, which are then sent to the FC layer(s). The outputs of the fully connected layer for video data and the fully connected layer for the IMU data are concatenated, and the concatenated features are then fed into another FC layer for classification. The number of neurons for this FC layer is also set by the GA, and it is denoted by $Para\_FC\_merge$ and shown in red in Fig. 5.1. This FC layer is followed by a softmax classifier. The output of the model is the confidence scores for each class proposal.

Next, in Sec. 5.1.1, we will describe the details of the algorithm that we propose to refine the various parameters of this architecture by using a Genetic Algorithm.

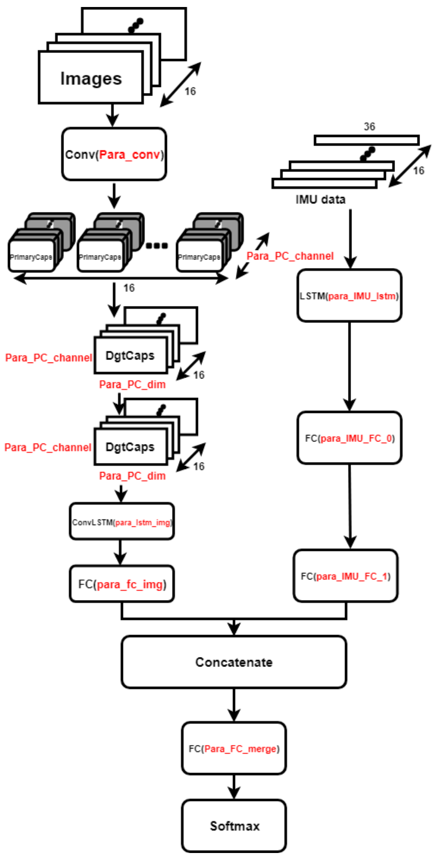


Figure 5.1: Details of The architecture.

5.1.1 Autonomously and Simultaneously Refining the Network Parameters

The overall structure of The method to refine the network parameters is shown in Fig. 5.2. In this approach, a GA is used to make a decision from a set of discrete choices. The decisions by the GA include the choice of the activation function and the optimizer; whether or not to use batch normalization, dropout and max-pooling; the number of convolutional layers and dense layers; the kernel size and stride, the number of LSTM units, etc. The

Table 5.1: Parameters Autonomously Chosen by the GA

optimizers	{"adam", "rmsprop", "adagrad", "adadelta"}
activation functions	{"relu", "leaky relu", "sigmoid", "tanh"}
batch normalization	{True, False}
dropout	{True, False}
max pooling	{True, False}
kernel size	{3, 6, 9}
kernel stride	{1, 2, 3}
number of conv filters	{32, 64, 128 ... 512}
number of dense neurons	{32, 64, 128, 256}
number of lstm units	{16, 32, 64 ... 256}
dimension of capsules	{2, 4, 8, 16}
number of primary channels	{16, 32, 64}
number of conv layers	{3,6}
number of dense layers	{1,3}
number of LSTM layers	{1,3}

complete set of parameters together with the discrete set of values that they can take are shown in Table 5.1.

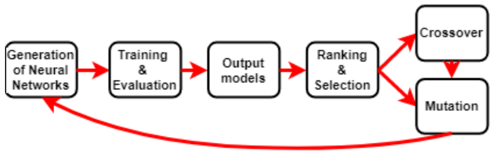


Figure 5.2: The structure of The Genetic Algorithm

The parameter set of the network i for the GA has the following form:

$$\begin{aligned} p_i^{GA} =& [prm_i^{conv}, prm_i^{PC}, prm_i^{LSTM_{img}}, prm_i^{FC_{img}}, \\ & prm_i^{LSTM_{IMU}}, prm_i^{FC^0_{IMU}}, prm_i^{FC^1_{IMU}}, \\ & prm_i^{FC_{merge}}, prm_i^{O}] \\ & i \in \{1, 2, ..., n_m\} \end{aligned} \tag{5.1}$$

where PC and FC denote primary capsule and fully connected layer, respectively, n_m is the number of network models in the population and

$$\begin{aligned} prm_i^{conv} =& [1/0 \text{ (conv. lyr exists or not)}, \text{no. of filters}, \\ & \text{kernel size}, \text{stride}, \text{activation func.}, \\ & 1/0 \text{ (for batch norm.)}], \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.2}$$

$$\begin{aligned} prm_i^{PC} =& [\text{capsule dim.}, \text{num of chan.}, \\ & \text{kernel size}, \text{stride}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.3}$$

$$\begin{aligned} prm_i^{LSTM_{img}} =& [1/0 \text{ (LSTM lyr exists or not)}, \text{no. of units}, \\ & \text{activation func.}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.4}$$

$$\begin{aligned} prm_i^{FC_{img}} =& [1/0 \text{ (dense lyr exists or not)}, \text{no. of neurons}, \\ & \text{activation func.}, 1/0 \text{(for dropout)}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.5}$$

$$\begin{aligned} prm_i^{LSTM_{IMU}} =& [1/0 \text{ (LSTM lyr exists or not)}, \text{no. of units}, \\ & \text{activation func.}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.6}$$

$$\begin{aligned} prm_i^{FC_{IMU^{0/1}}} =& [1/0 \text{ (dense lyr exists or not), no. of neurons,} \\ & \text{activation func., } 1/0 \text{(for dropout)}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.7}$$

$$\begin{aligned} prm_i^{FC_{merge}} =& [1/0 \text{(dense lyr exists or not), no. of neurons,} \\ & \text{activation func., } 1/0 \text{(for dropout)}] \\ & i \in \{1, 2, ..., n_m\}, \end{aligned} \tag{5.8}$$

$$\begin{aligned} prm_i^{O} =& [\text{type of optimizer}] \\ & i \in \{1, 2, ..., n_m\}. \end{aligned} \tag{5.9}$$

Initial Population

The first generation of the networks, N^1, is generated randomly such that $N^1 = \{N_1, N_2, ..., N_{n_m}\}$, where n_m is the number of models. This is done by choosing the values of parameters, in (5.2) through (5.9), randomly, from the possible choices. The value of n_m was set to be 10 in our experiments.

Evaluation

Each generated network model N_i ($i \in \{1, ..., n_m\}$) is evaluated by the fitness function $f(N_i)$, which is a measure of the accuracy of each model. Models with better performance will return higher values. Thus, $E = \{E_1, E_2, ..., E_{n_m}\}$, will hold the fitness scores $E_i = f(N_i)$, where $i \in \{1, 2, ..., n_m\}$.

Selection

In the selection part, t-many top-ranked models are selected from the $sorted(E)$, and r-many models are selected randomly from the rest of the network models. Then, d-many models are dropped in order to prevent over-fitting and getting stuck at a local optimum. The remaining selected models are the parent models (P), which will be used to create new models for the next generation.

Crossover and Mutation

Crossover is applied to generate n_m-many child network models from the parents. As opposed to always choosing two parents randomly from the parent pool, we associate a counter C_P with each parent P, and initialize it to zero. This counter is incremented by one each time a parent is used for crossover. First, two parents are selected randomly from the $t + r - d$ many parents. A new 'child' network is generated from the parents via crossover, and the counters of the parents are incremented by one. Then, two parents, whose counter is still zero, are selected randomly from the parent pool. Another network is generated from them via crossover, and the counters of the parents are incremented. If there is only one network model left with counter equal to zero, and the number of children is still less than n_m, then this model is chosen as one of the parents, and the other parent is chosen randomly from the rest of the models who have a counter value of one. If there are no more parents left with counter equal to zero, and the number of children is still less than n_m, then two parents, whose counter is one, are picked randomly, and their counter is incremented to two after crossover. This process is repeated until the number of children models reaches n_m.

The crossover between parent models a and b is performed, as illustrated in Fig. 5.3, by using a single-point crossover. As seen from equations (5.1) through (5.9), there are a total of 33 parameters in each parent vector. An index number ind is picked randomly between 1 and 33. Parameters to the left of ind from the parent a vector and to the right of ind from the parent b vector are selected to compose the child vector. In other words, parameters 1 through ind, and $ind+1$ through 33 are selected from parents a and b, respectively, to form the child vector.

After all the n_m-many child networks are obtained via crossover, the mutation is performed. From each vector, k-many indices are chosen randomly to perform mutation. The values of the parameters corresponding to the chosen indices are randomly changed to one of the possible choices shown in Table 5.1. For instance, if the random number corresponds to the dimension of the capsules, then its value is chosen randomly from $\{2, 4, 8, 16\}$. The value of k was chosen to be 3 in our experiments.

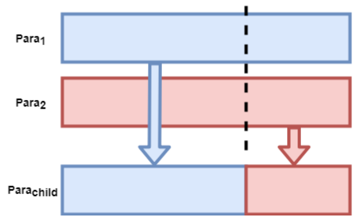


Figure 5.3: Crossover process for the GA

Then, the entire process is repeated by using this new population. The pseudo code for GA-based parameter setting is provided in Algorithm 1. After the parameters are set autonomously by the GA-based approach, the network model is generated as described in the pseudo code in Algorithm 2.

Algorithm 1 The Genetic Algorithm

```
Randomly initialize $n_m$ models for population $N^1$.
while $i^{th}$ iteration do
    Train and evaluate $N_1^i, N_2^i, ..., N_{n_m}^i$ by fitness function $f(N_j^i)$ and obtain scores $E$.
    Select $t$ top scored networks $N_{top} = N^i(argmax(E))$
    Randomly choose $r$ networks $N_{rand}$ from the rest of population $N^i$
    Merge $N_{top}$ and $N_{rand}$ and then drop $d$ networks ($N_{drop}$)
    Form $N_{parent} = (N_{top} \bigcup N_{rand}) - N_{drop}$
    Choose parents from $N_{parent}$ for crossover and generate $n_m$ new networks and add them to $N^{i+1}$
    Perform mutation on k-many elements of vectors in $N^{i+1}$.
end
```

Algorithm 2 Network model generation from the GA-set parameters

```
Input: Genetic representative vector $\mathcal{L}$; vector prototype $p^{GA} = [prm_{video}, prm_{IMU}, prm_{merge}]$ shown in eq.(1);
$prm_{video} = [prm_i^{conv}, prm_i^{PC}, prm_i^{LSTM_{img}}, prm_i^{FC_{img}}]$
$prm_{IMU} = [prm_i^{LSTM_{IMU}}, prm_i^{FC^0_{IMU}}, prm_i^{FC^1_{IMU}}]$
$prm_{merge} = [prm_i^{FC_{merge}}]$
Input: video input shape $S_v$; IMU input shape $S_{IMU}$;
Output: output model $M$

$input_{video} \leftarrow Placeholder(shape = S_v)$
for $i = 0$ to $length(input_{video}) - 1$ do
    $offset \leftarrow 0$
    $M_i \leftarrow Sequential(input_{video}[i])$
    for $idx = 0$ to $length(prm_{video})$ do
        Build layer $L_{idx}$ from $L[offset : offset + len(prm_{video}[idx])]$
        $M_i \overset{+}{\leftarrow} L_{idx}$
        $offset += length(prm_{video}[idx])$
    end
    $output^i_{video} = M_i(input^i_{video})$
end
$input_{IMU} \leftarrow Placeholder(shape = S_{IMU})$
$M_{IMU} \leftarrow Sequential(input_{IMU})$
for $idx = 0$ to $len(prm_{IMU})$ do
    Build layer $L_{offset}$ from $L[offset : offset + length(prm_{IMU}[idx])]$
    $M_{IMU} \overset{+}{\leftarrow} L_{idx}$
    $offset += length(prm_{IMU}[idx])$
end
$output_{IMU} = M_{IMU}(input_{IMU})$
obtain concatenate layer $L_{concat}$ from $output_{video}$ and $output_{IMU}$
$M_{merge} \leftarrow Sequential(L_{concat})$
for $idx = 0$ to $length(prm_{merge})$ do
    Build layer $L_{idx}$ from $L[offset : offset + len(prm_{merge}[idx])]$
    $M_{merge} \overset{+}{\leftarrow} L_{idx}$
    $offset += len(prm_{merge}[idx])$
end
$output_{merge} = M_{merge}(L_{concat})$
return $Model([input_{video}, input_{IMU}], output_{merge})$
```

5.2 Experimental Results

5.2.1 Experimental Setup

We have used the CMU-MMAC dataset [56] for the experiments. This dataset contains data from multi-modal sensors monitoring human subjects preparing food. A kitchen was built and 25 subjects were recorded cooking five different recipes, namely brownies, pizza, sandwich, salad, and scrambled eggs. The sensor modalities used for data collection include three high-resolution static cameras, two low-resolution static cameras, one wearable camera, five microphones, and IMUs. In our experiments, we used the egocentric (egovision) camera data (from the wearable camera) and the wired IMU data. We resized the image frames from the camera to 36×36 pixels, and processed 16 consecutive frames each time with 50% overlapping. We down-sampled the IMU data to make the measuring frequency the same with the egocentric camera (30 Hz). Then, we synchronized/aligned the IMU data with camera data.

We performed two sets of experiments with different number of classes. More specifically, we performed 9-class labeling and 26-class labeling by using 9 and 26 different activity classes, respectively. The activities used for the 9-label classification are:
A_9= {'fridge(open or close)', 'taking/beating eggs', 'pouring into big bowl', 'pouring into cup', 'stirring in a bowl', 'taking bowl', 'taking baking pan', 'taking measuring cup', 'twisting cap (on or off)'}.
Example images for these nine classes can be seen in Fig. 5.9.

The activities used for the 26-label classification are:
A_{26} = {'closing fridge', 'cracking egg', 'opening brownie bag', 'opening fridge', 'pouring big bowl into a pan', 'pouring brownie bag into a bowl', 'pouring oil into a cup', 'pouring water into a bowl', 'pouring water into a cup', 'putting pan into oven', 'putting cooking spray/pam into cupboard', 'spraying cooking oil', 'stirring in a bowl', 'switching on', 'taking baking pan', 'taking bowl', 'taking brownie box', 'taking eggs', 'taking fork', 'taking

big cup', 'taking small cup', 'taking cooking spray', 'twisting cap off', 'twisting cap on', 'walking to the counter', 'walking to the fridge'}.

As can be seen, especially for the 26-class case, the activities involved are very close in the 'activity space', and this fine-grain classification is a very challenging problem.

A total of 10 videos from subjects 07, 08, 09, 12 and 13 (2 videos per subject) have been used for training and testing. Videos from each subject were randomly divided so that 70%, 20%, 10% of the samples were allocated for training, validation and testing, respectively.

We also compared our results with two other works [54][125], which also use the same CMU-MMAC dataset. All the results are presented below in Section 5.2.2.

5.2.2 Results and Discussion

As mentioned above, we performed 9-class and 26-class labeling in our experiments. In each scenario, we first performed classification with manually preset network parameters, and then with the parameters determined autonomously by our GA-based approach described above. Preset parameters were obtained by choosing the parameter configuration that resulted in the best performance after multiple trials. For both 9-class and 26-class labeling, the preset parameters (corresponding to equations (1) through (9)) are:

$$\begin{aligned}&[1, 256, 9, 1, 0, 0, 8, 32, 9, 2, 1, 256, 0, 1, 128, 0, 0, 1, 128, 0, 1,\\&128, 0, 0, 1, 32, 0, 0, 0, 128, 0, 0, 0]\end{aligned} \tag{5.10}$$

The parameters determined autonomously by our proposed GA-based approach are:

$$\begin{aligned}&[1, 32, 6, 3, 0, 1, 16, 32, 3, 1, 0, 32, 0, 0, 128, 1, 0, 0, 64, 1, 1,\\&64, 0, 0, 0, 256, 1, 0, 0, 32, 0, 0, 0]\end{aligned} \tag{5.11}$$

and

$$[1, 64, 9, 3, 0, 1, 8, 64, 6, 2, 0, 256, 1, 0, 64, 1, 0, 0, 64, 1, 1,$$
$$256, 0, 0, 0, 128, 1, 0, 0, 64, 0, 0, 0] \tag{5.12}$$

for 9-class and 26-class classification, respectively. For instance, in both cases, the GA-based approach results in less number of filters for the convolutional layers (32 and 64 instead of 256), and less number of neurons for the fully connected merge layer. The overall accuracies from these experiments are summarized in Table 5.2, wherein the accuracy is the ratio of all correctly classified instances to the total number of instances. As can be seen, when we use our proposed GA-based approach to autonomously set the various parameters of the network, this provides higher accuracy for both 9-class and 26-class labeling. Thus, the remainder of the results are presented for when the parameters are set with our GA-based approach.

Table 5.2: Overall accuracies for the 9- and 26-class labeling with and without using The GA-based parameter setting

	9-class		26-class	
	Preset parameters	GA-based parameters	Preset parameters	GA-based parameters
Accuracy	84.2%	**86.6%**	75.7%	**77.2%**

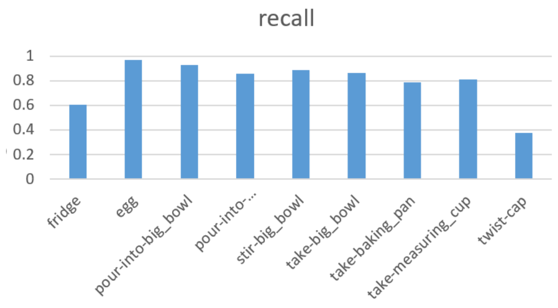


Figure 5.4: The recall values for each of the 9 classes.

The recall and precision values for each class, for the 9-class case, are shown in Figures 5.4 and 5.5, respectively. For the 26-class case, the recall and precision values for each

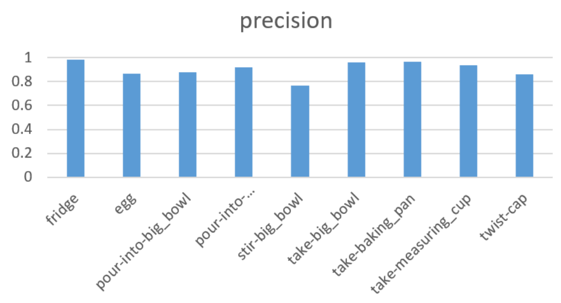


Figure 5.5: The precision values for each of the 9 classes.

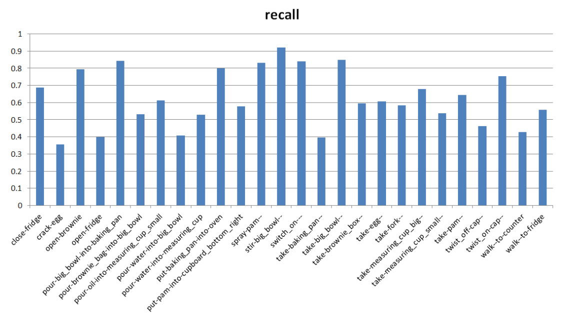


Figure 5.6: The recall values for each of the 26 classes.

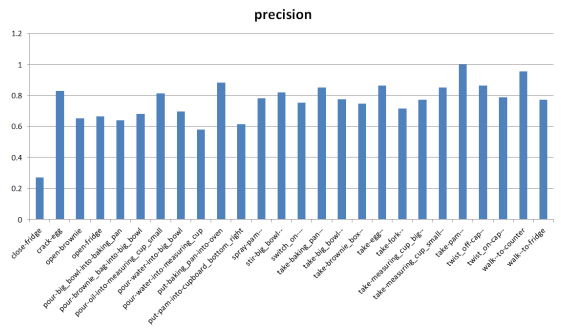
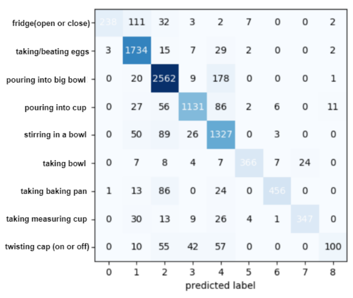
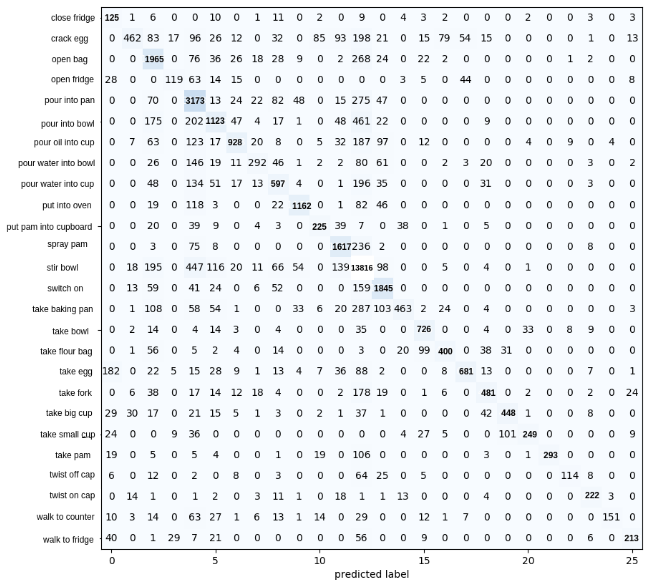


Figure 5.7: The precision values for each of the 26 classes.

class are shown in Figures 5.6 and 5.7, respectively. The confusion matrices for the 9-class and 26-class activity classification are shown in Figures 5.8(a) and 5.8(b), respectively. As can be seen from the precision-recall graphs and the confusion matrices, when subjects interact with larger objects, and subject movements are faster, higher accuracy is achieved compared to the slower movements and interacting with smaller objects. For instance, it is harder to detect 'twisting cap on' and 'twisting cap off' actions, since the cap is always occluded by hand in the camera view. As another example, actions such as 'cracking egg'



(a) 9-class labeling



(b) 26-class labeling

Figure 5.8: Confusion matrices showing the correct versus predicted classes together with the number of instances of each activity for (a) 9-class and (b) 26-class activity classification.

are also harder to classify, since the egg is much smaller than the bowl.

In addition, as expected, higher overall precision and recall rates are achieved for 9-class labeling, since activities are much closer to each other and harder to differentiate for the 26-class labeling case. In Fig. 5.10, we show example images for the activities that are confused with each other in the 26-class labeling case (based on the confusion matrix in Fig. 5.8(b)). These images illustrate once more the difficulty of performing very fine-grained activity classification. The first row shows taking a small cup (on the left) vs. big cup (on the right). The second row shows walking to the fridge (on the left) vs. closing the fridge (on the right), and finally the third row shows pouring into pan (on the left) vs. putting the pan into the oven (on the right). As can be seen, these are very similar looking activities, and The approach still provides very promising results for the 26-class labeling. More discussion and comparison will be provided below.

After setting the various network parameters by our GA-based approach, we then performed a comparison of our proposed Rec-CapsNets method with using VGG16 features.

For this comparison, instead of employing The RecCapsNet, we extracted image features from 16 consecutive image frames by using the convolutional layers of the CNN-based VGG16 [124] without the top layers. We also used CapsNet on individual frames. We used the same dataset splitting, described above, for all compared methods. The results are summarized in Table 5.3 for 9-label classification. As can be seen, using our proposed RecCapsNet provides a higher accuracy than using the VGG16 features. Moreover, to show the improvement provided by using multiple sensor modalities, we also obtained results by using each sensor modality by itself, namely by using only IMU data and only camera data. As can be seen in Table 5.3, The approach provides 29.07%, 20.29% and 19.16% increase in accuracy compared to using only IMU data, only egocentric camera data with VGG16 features and only egocentric camera data with CapsNet features, respectively.

The above comparison was performed for 26-label classification as well, and the results are summarized in Table 5.4. Similar to the 9-class case, using our proposed RecCapsNet together with LSTM on IMU data provides a higher accuracy than using the VGG16 features. In addition, The approach provides 28%, 19.5% and 25.2% increase in accuracy compared to using only IMU data, only egocentric camera data with VGG16 features and only egocentric camera data with CapsNet features, respectively. For Tables 5.3 and 5.4, the parameters used for each approach are as follows:

$$\begin{aligned} \text{LSTM (for IMU data): } & \text{LSTM (128)} \rightarrow \text{LSTM(64)} \rightarrow \\ & \text{FC(128)} \rightarrow \text{FC(64)} \\ \text{VGG16: } & \text{parameters from [124]} \\ \text{CapsNet: } & \text{parameters from [122]} \end{aligned}$$

For The method (in Tables 5.3 and 5.4), we used the parameters in equations (5.11) and (5.12), respectively, which were autonomously set by our GA-based approach.

Since this is fine-grained classification, using multiple modalities provides a significant increase in performance compared to single-modality results.

We also compared our results with two other works [54][125], which also use the same CMU-MMAC dataset. Soran et al. [125] does not use IMU data, and either employ egocentric camera data or combine egocentric camera data with static camera data. From only the egocentric camera data, Soran et al. [125] report an accuracy of 37.92% for 28 classes. With our approach, when we exclude the IMU data, we obtain an accuracy of 67.48% and 52% for 9-class and 26-class labeling, respectively. Thus, The approach provides much higher performance.

Spriggs et al. [54], on the other hand, report an accuracy of 57.8% over 29 classes when using IMU data and egocentric camera data together. Our accuracy over 26 classes listed in 5.8(b) is 77.2%. In order to make the comparison more commensurate, we performed an additional experiment. More specifically, we have trained and tested our proposed method with the same 29 classes used in [54]. The accuracy we obtained is 83.03% for the 29-class labeling. These results are summarized in Table 5.5.

Overall, our proposed method provides a significant improvement without relying on the static cameras watching the targets, which could also be important to alleviate the pri-

Table 5.3: Accuracy rates from different modalities and approaches for 9-label classification

Sensor Modality	Method	Accuracy
IMU only	LSTM	57.57%
Camera only	VGG16	66.35%
	CapsNet	67.48%
Camera and IMU	VGG16 & LSTM	82.97%
	RecCapsNet & LSTM (Proposed)	**86.64%**

Table 5.4: Accuracy rates from different modalities and approaches for 26-label classification

Sensor Modality	Method	Accuracy
IMU only	LSTM	49.2%
Camera only	VGG16	57.7%
	CapsNet	52%
Camera and IMU	VGG16 & LSTM	74.6%
	RecCapsNet & LSTM (Proposed)	**77.2%**

vacy concerns. In addition, using The GA-based approach not only provides a way to systematically set the network parameters, but also improves the performance further compared to using the manually set parameters.

Table 5.5: Comparison of different approaches

Method	Sensor Modality	No. of classes	Accuracy
Soran et al. [125]	ego. cam.	28	37.92%
Ours	ego. cam	26	**52%**
Ours	ego. cam & IMU	26	77.2%
Spriggs et al. [54]	ego. cam & IMU	29	57.8%
Ours	ego. cam & IMU	29	**83.03%**

5.3 Conclusion

We have presented a robust and autonomous method to perform fine-grain activity classification by leveraging data from multiple sensor modalities, more specifically egocentric video and IMU sensor data from wearable devices. In contrast to many CNN-based approaches, we have proposed to use a capsule network to obtain features from egocentric video data. Instead of using a single CapsNet, multiple CapsNets are employed for consecutive images, and then a convolutional LSTM is used to build a recurrent CapsNet. The LSTM framework is employed both on IMU data and egocentric camera data to capture the temporal aspect of actions, which span a time window. Moreover, we proposed a GA-based approach to autonomously and systematically set the various parameters of our network architecture. It has been shown that using The GA-based approach increases the accuracy compared to using the manually set parameters. Results have been presented for 9-label, 26-label and 29-label classification. The method has provided promising results, achieving an overall accuracy of 86.6% 77.2%, and 83.03% for 9-label, 26-label and 29-label classification, respectively. This approach can be readily extended to classify more activity types. As future work, we will incorporate a generative adversarial network-based approach to increase the range of parameters that can be chosen autonomously.

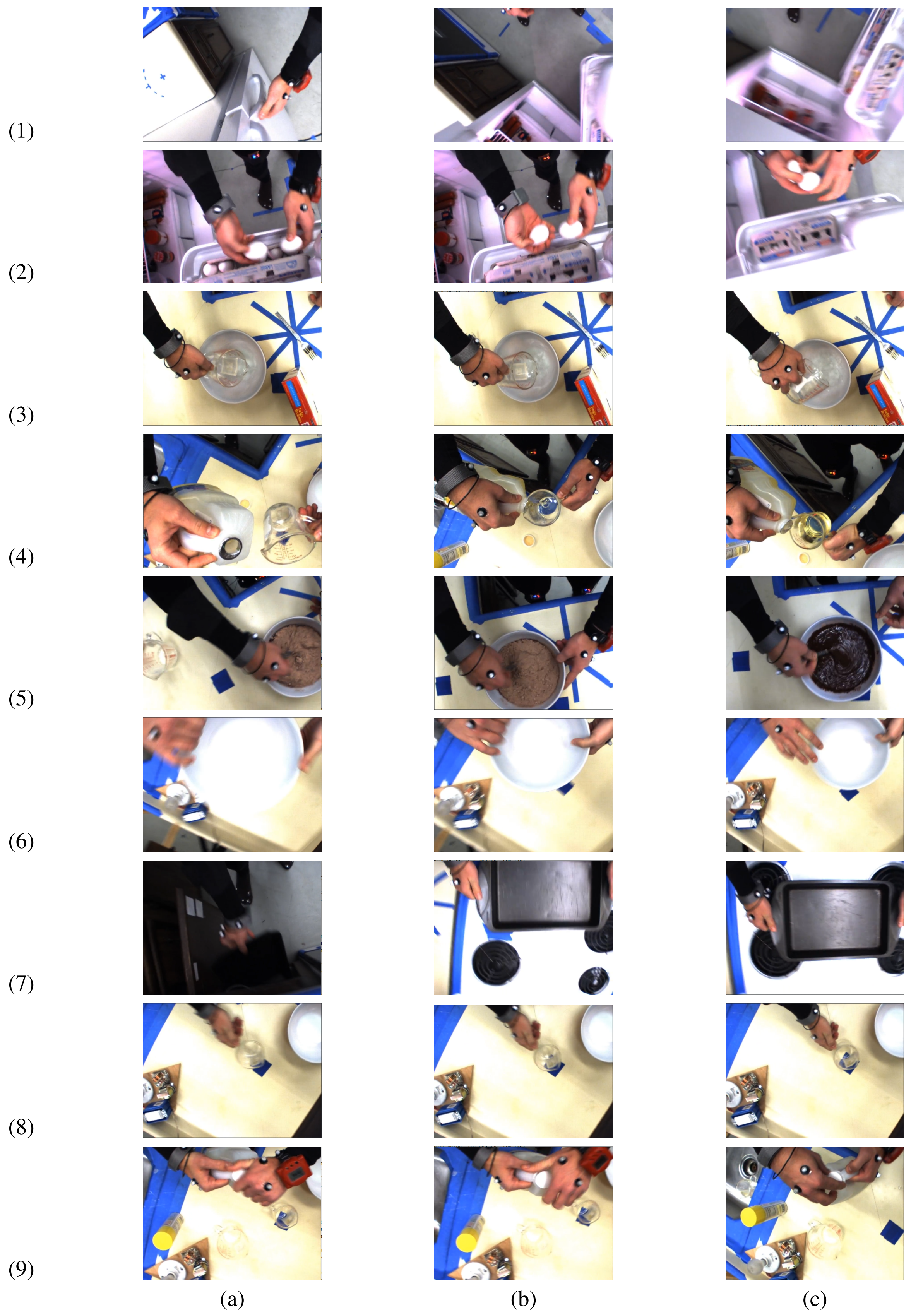


Figure 5.9: Example images of the 9 activity classes from the CMU-MMAC dataset. Rows: (1) using fridge, (2) taking eggs, (3) pouring into big bowl, (4) pouring into a measuring cup, (5) stirring in a big bowl, (6) taking bowl, (7) taking baking pan, (8) taking measuring cup, (9) twisting cap (on or off). Columns (a), (b) and (c) show images from the beginning, middle and end of each activity.

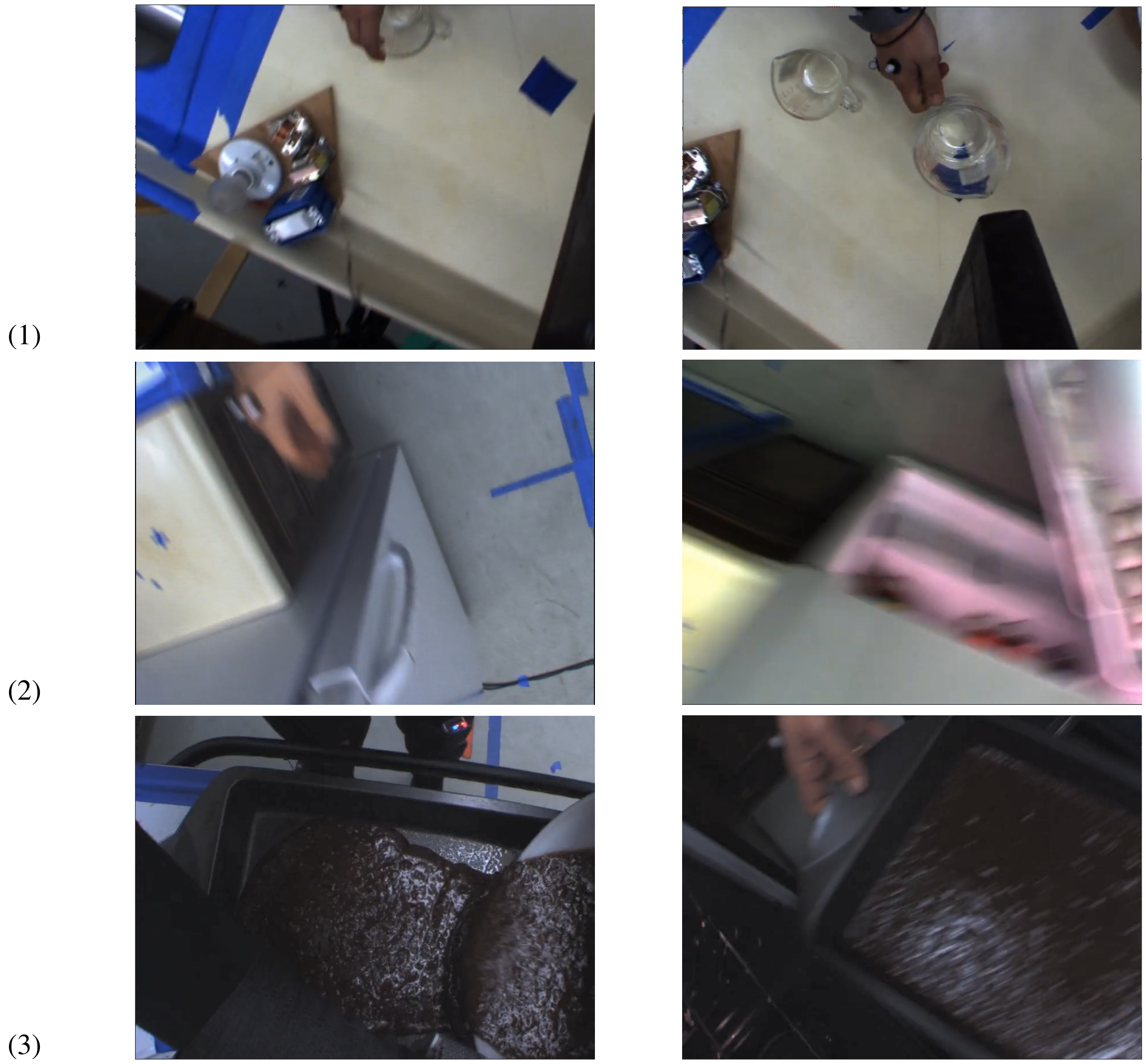


Figure 5.10: Examples images of challenging cases causing confusion. Rows: (1) taking a small cup (on the left) vs. big cup (on the right), (2) walking to fridge (on the left) vs. closing fridge (on the right), (3) pouring into pan (on the left) vs. putting the pan into oven (on the right).

Chapter 6

Efficient Human Activity Classification From Egocentric Videos Incorporating Actor-Critic Reinforcement Learning

This chapter studies a novel approach to significantly reduce the computational cost of human activity classification from egocentric videos while maintaining the accuracy at the same level. We leverage the actor-critic model of RL, and apply it to optical flow data to determine how to move a bounding box in x and y directions to maximize the reward, and find an optimal region of interest. The bounding box is used for clipping a portion of the image. We also propose to use one shallow and one deeper convolutional neural network to process the original image and the clipped image region, respectively. This overall proposed architecture will henceforth be referred to as the Deep-Shallow Network. We compared our method with another approach, using 3D convolutional networks for activity recognition, on the recently released Dataset of Multimodal Semantic Egocentric Video. The results will be presented in Sec. 6.2.

6.1 Methodology

The overall Deep-Shallow Network, shown in Fig. 6.1, is composed of a shallow network, a deeper network and an image clipper. Both shallow and deep feature extractors are 3D convolutional neural networks (CNNs). The shallow feature extractor takes the original images as input, and uses relatively larger kernels and fewer layers to extract environment features from the larger original image. On the other hand, the deep feature extractor takes the clipped image regions as input, and uses smaller kernels to extract activity features. The details of the shallow and deep network models can be seen in Fig. 6.2.

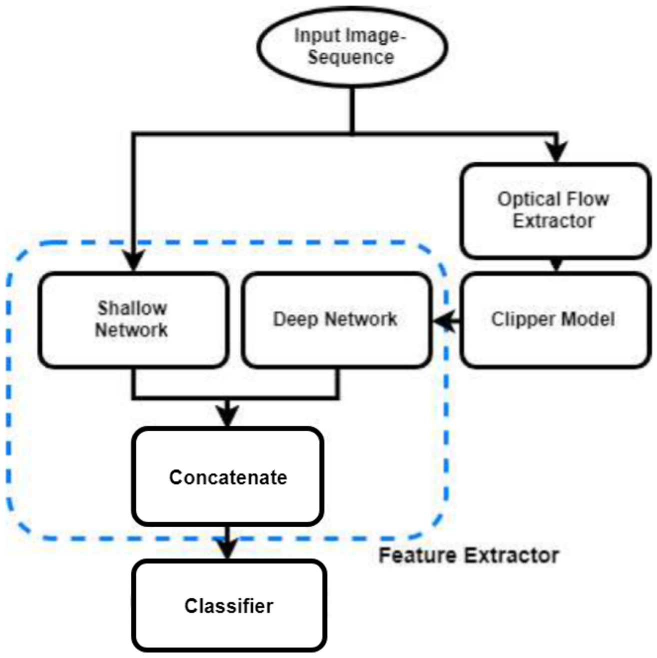
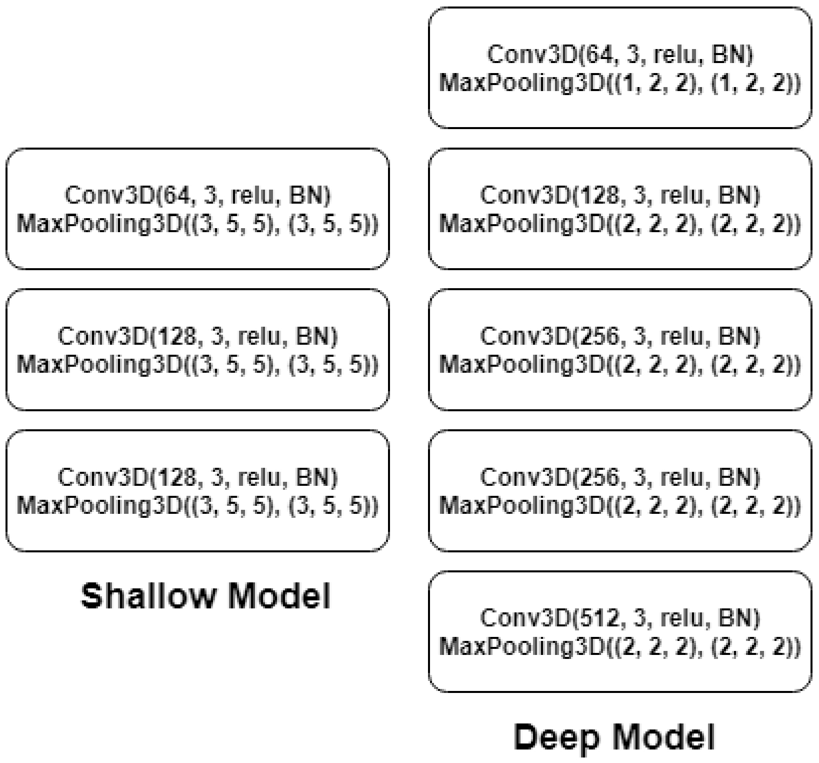


Figure 6.1: Overall structure of the proposed Deep-Shallow Network.

The image clipper is trained based on the actor-critic model of RL. The input of the actor-critic model is the optical flow data extracted from the original images. Extracted features from the shallow and deep networks are concatenated, and followed by fully connected layers to obtain classification results.

As a result of reducing the complexity of the network structure, and only processing the regions of interest with a deep network, the proposed Deep-Shallow Network can signifi-


Conv3D(64, 3, relu, BN)
MaxPooling3D((3, 5, 5), (3, 5, 5))
Conv3D(128, 3, relu, BN)
MaxPooling3D((3, 5, 5), (3, 5, 5))
Conv3D(128, 3, relu, BN)
MaxPooling3D((3, 5, 5), (3, 5, 5))
Shallow Model
Conv3D(64, 3, relu, BN)
MaxPooling3D((1, 2, 2), (1, 2, 2))
Conv3D(128, 3, relu, BN)
MaxPooling3D((2, 2, 2), (2, 2, 2))
Conv3D(256, 3, relu, BN)
MaxPooling3D((2, 2, 2), (2, 2, 2))
Conv3D(256, 3, relu, BN)
MaxPooling3D((2, 2, 2), (2, 2, 2))
Conv3D(512, 3, relu, BN)
MaxPooling3D((2, 2, 2), (2, 2, 2))
Deep Model
Figure 6.2: Deep and shallow network model details.

cantly increase the processing speed, while maintaining the same level of accuracy with a state-of-the-art 3D CNN network.

6.1.1 Clipper Model Trained with Deep Reinforcement Learning

The main goal of the proposed approach is to reduce the computational complexity without sacrificing accuracy. A deep reinforcement learning (DRL)-based approach is adopted in this work to train the image clipper, which determines the location of the region of interest by moving a fixed-size bounding box. The height and width of the bounding box is half of the original image size. We build a standard reinforcement learning [83] setup up to derive the correlation between each state-action pair (s, a) of the system under control and its value function $Q(s, a)$ in discrete decision epochs. At each decision epoch t_k of the processing, the agent, i.e. the video frame at t_k, is at state s_k, and performs inference using deep neural network to select action a_k according to the policy π. We define the control actions as (Δ_x, Δ_y) with real values, which represent offsets of the bounding box in x and y directions, respectively. Since our problem has continuous output space, an actor-critic-based DRL [81] is adopted. Under a certain policy π, the value of $Q(s, a)$ estimates the accumulated discounted reward of each state-action pair:

$$Q(s,a) = \mathbf{E}(\Sigma_{k=0}^{\infty} \lambda^k r_k(s_k, a_k) | s_0 = s, a_0 = a)) \tag{6.1}$$

where r_k is the total reward observed at decision epoch t_k. To accelerate learning, and avoid oscillations or divergence in the parameters, we employ an experience replay and target network [84]. The experience replay updates the weights of the target network θ' based on learned network weights θ by:

$$\theta' = \tau\theta + (1-\tau)\theta', \qquad \tau \ll 1 \tag{6.2}$$

The actor model is a feed-forward neural network composed of three fully-connected hidden layers with rectified linear units (ReLU) as the activation function. It is used to predict the optimal action based on the current state $\mathbb{S}_t$. The number of neurons in fully connected hidden layers are 64, 128 and 128, respectively. The output layer size is 2 providing the horizontal and vertical offsets for the bounding box.

The critic model is another feed-forward neural network that evaluates the state and action pair, and the evaluation is used by the actor model to update its control policy in particular gradient direction. The critic model has two hidden layers. The first layer contains two separate fully-connected structures and the number of hidden neurons in each is 32. The addition of outputs from the first hidden layer is fed into the second layer which has 64 hidden neurons. The inputs of the critic model are $\mathbb{S}_t$ and $\mathbb{A}_t$, and the output is a single value $Q(\mathbb{S}_t, \mathbb{A}_t)$. The actor-critic framework is shown in Fig. 6.3.

During training, the actor model is trained using pair data $(\mathbb{S}_t, \mathbb{A}_t)$ to predict the optimal action $\mathbb{A}_t$ based on current agent state $\mathbb{S}_t$. Next agent state $\mathbb{S}_{t+1}$ is calculated through environment based on $\mathbb{A}_t$ and is used to predict optimal $\mathbb{A}_{t+1}$ by actor model. The critic model evaluates the resulting $\{\mathbb{S}_{t+1}, \mathbb{A}_{t+1}\}$ pair by predicting a Q-value to fine-tune action prediction. Therefore, the weights in actor model are updated by the gradient between actor and critic model, using chain rule $dQ/dW_{actor} = dQ/dW_{critic} \times dW_{critic}/dW_{actor}$. W_{actor} and W_{critic} indicate the weights of $\underline{actor}$ and $\underline{critic}$ models, respectively.

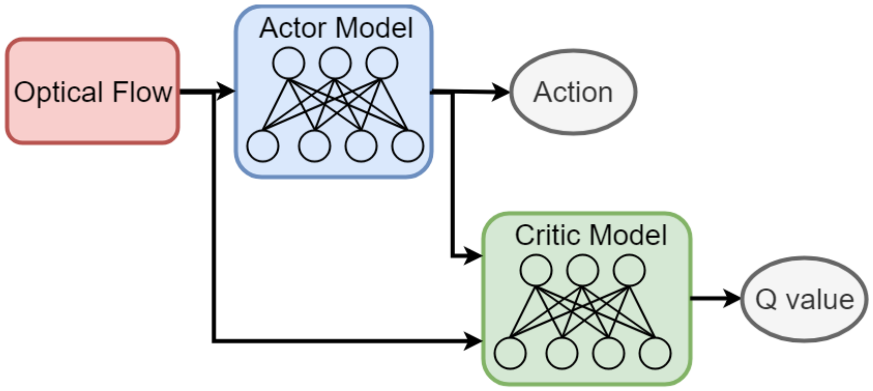


Figure 6.3: Actor-critic based clipper model.

Example images from four different egocentric videos are shown in Fig. 6.4 together with the bounding boxes placed via the actor critic model. The first, second and third rows show frames $(t - 10)$, t and $(t + 10)$, respectively. As can be seen, the box placed by the actor-critic model moves inside the image to determine a focus region of interest.

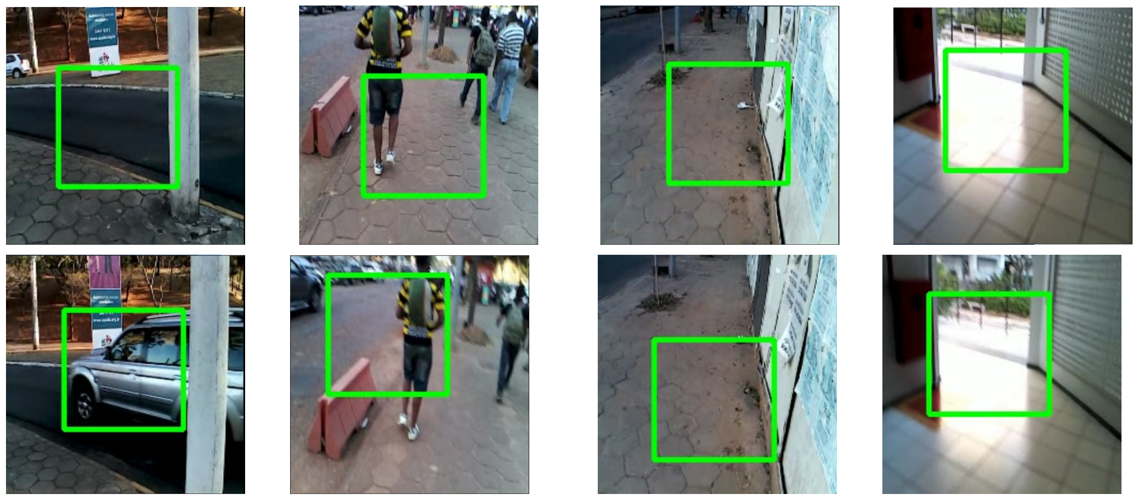


Figure 6.4: Examples showing the autonomously placed bounding boxes. 1st and 2nd rows show frames (t-10) and t, respectively.

6.2 Experimental Results

We compared our proposed Deep-Shallow network with a commonly used 3D CNN [74], which will be referred to as C3D. We have used a recently released dataset called Dataset of Multimodal Semantic Egocentric Video (DoMSEV) [127]. DoMSEV contains 80 hours of egocentric video covering a wide range of activities. The videos were recorded using either a GoPro Hero camera or a built setup composed of a 3D Inertial Movement Unit (IMU) attached to the Intel Realsense R200 RGB-D camera. The activities performed while recording include walking, running, standing, browsing, driving, biking, eating, cooking, eating, observing, in conversation, playing, and shopping. We selected 11 videos (8 Tourism and 3 Daliy life videos), and five activities ($walking, running, standing, in\ conversation, browsing$) as labels. We segment the videos into video clips of 60 frames with 50% overlapping. Then, we randomly separate 80% of data for training and 20% for testing. 20% of the training data is used for validation. The curves of training and validation loss of our Deep-Shallow model are shown in Fig. 6.5. The loss and reward curves of the actor-critic-based clipper model are shown in Fig. 6.6 and Fig. 6.7, respectively.

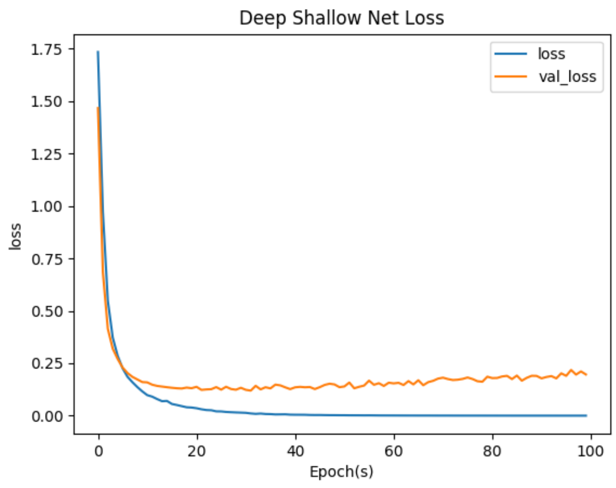


Figure 6.5: Deep-Shallow network training loss.

As mentioned above, we compared the performances of the method and the traditional C3D in terms of speed and accuracy. For all the video clips (60 frame duration) in one

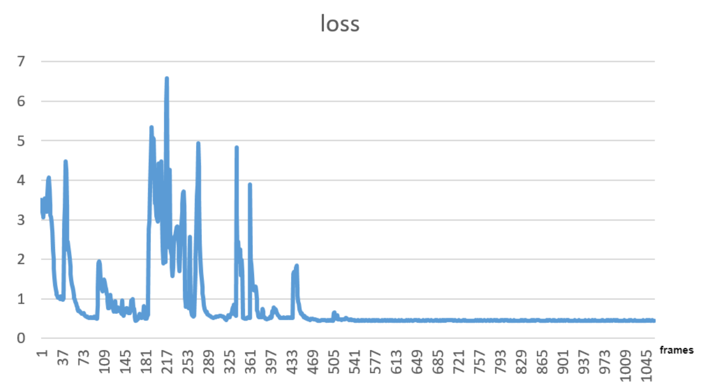


Figure 6.6: Actor model training loss.

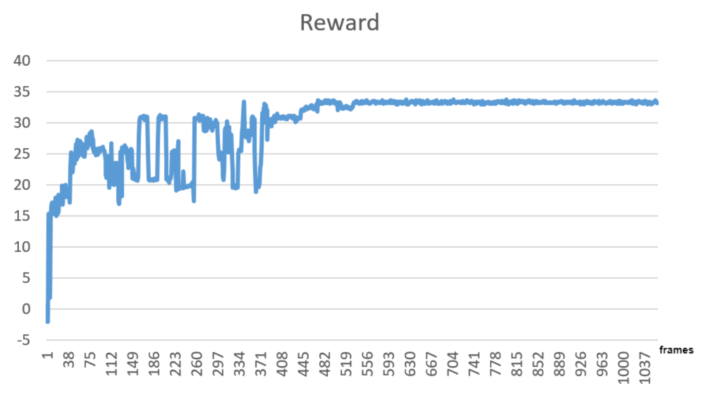


Figure 6.7: Actor model training reward.

video, we measured how long it takes to process them, and took the average. As shown in Table 6.1, the average processing time per clip is 906 ms for C3D, while the average processing time per clip is 576 ms for the proposed Deep-Shallow model. In other words, our proposed model is 36.4% faster than the C3D as seen in Fig. 6.8.

The precision and recall values for each class are shown in Fig. 6.9 and Fig. 6.10, respectively. The average precision of the C3D and the proposed Deep-Shallow network are 0.72 and 0.71, respectively. The average recall of the C3D and the method are 0.75 and 0.74, respectively. As seen in Table 6.1, the C3D and the proposed approach achieves 74% and 72.9% overall accuracy, respectively. In summary, the proposed approach provides a significant increase in processing speed with only 1.1% decrease in the accuracy.

	C3D	Deep-Shallow
Avg. process. time/clip	906 ms	576 ms
Overall accuracy	0.740	0.729

Table 6.1: Comparison table

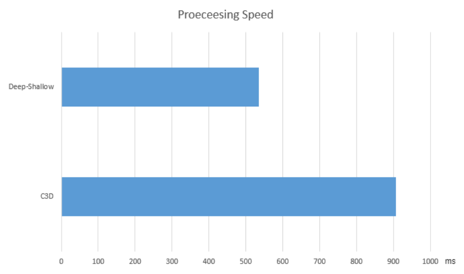


Figure 6.8: Processing speed comparison

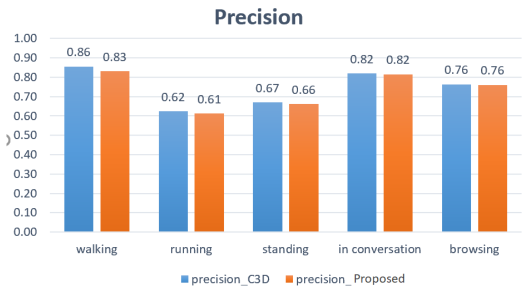


Figure 6.9: Precision values for each activity class

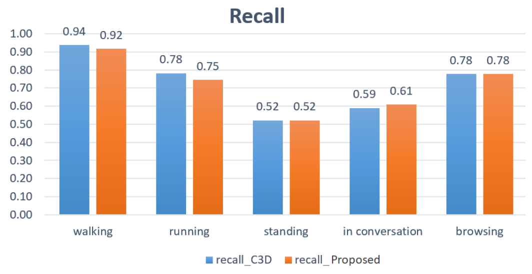


Figure 6.10: Recall values for each activity class

6.3 Conclusion

We have presented a novel method to efficiently perform human activity classification from egocentric videos by incorporating actor-critic model of reinforcement learning. Actor-critic reinforcement learning allows placing a bounding box on a region of interest, and clipping that region. Then, only the clipped region is processed through a deeper network, while the entire image is processed by a shallow one. This strategically reduced complexity of the network structure provides significant increase in the processing speed, while maintaining the same level of accuracy with a state-of-the-art 3D CNN network.

Chapter 7

Autonomously and Simultaneously Refining Deep Neural Network Parameters by a Bi-Generative Adversarial Network Aided Genetic Algorithm

This chapter focuses on optimizing the network architecture and its different parameters for any neural network model. We propose a novel and systematic way, which employs a revised generative adversarial networks, referred to as Bi-GAN, together with a Genetic Algorithm (GA). The method can autonomously refine the number of convolutional layers, the number and size of filters, number of dense layers, and number of neurons; decide whether to use batch normalization and max pooling; choose the type of the activation function; and decide whether to use dropout or not.

In this chapter, to move from exploration to exploitation, we propose a novel and systematic method that autonomously and simultaneously optimizes multiple parameters of any deep neural network by using a GA aided by our proposed bi-generative adversarial network (Bi-GAN). In contrast to traditional GANs, our proposed Bi-GAN involves two generators, and two different models compete and improve each other progressively with a GAN-based strategy to optimize the networks during GA evolutions. The Bi-GAN allows the autonomous exploitation and choice of the number of filters and number of neurons from a large range of values. Our proposed approach can be used to autonomously refine the number of convolutional layers, the number and size of filters, number of dense layers, and number of neurons; decide whether to use batch normalization and max pooling; choose the type of the activation function; and decide whether to use dropout or not, to improve the accuracy of different deep neural network architectures.

For this work, without loss of generality, we tested the performance of our approach by using the ModelNet database, and compared it with the 3D Shapenets and two GA-only methods. The results show that the presented approach can simultaneously and successfully optimize multiple neural network parameters, and achieve increased accuracy even with shallower networks. The rest of this chapter is organized as follows: The method is described in Sec. 7.1. The experimental results are presented in Sec. 7.2, and the chapter is concluded in Sec. 7.3.

7.1 Methodology

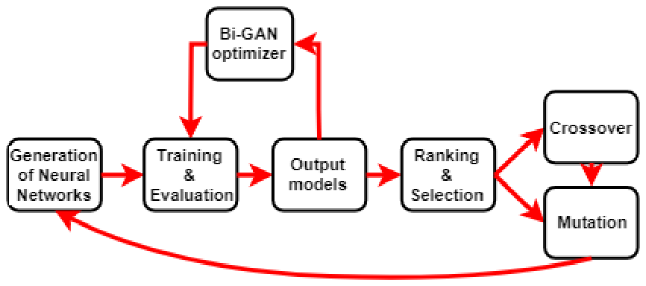


Figure 7.1: Proposed Bi-GAN aided GA network for refining deep neural network parameters.

The overall structure of the method is shown in Fig. 7.1. It involves a GA aided by a

Bi-GAN. While Bi-GAN is used to set/optimize the parameters from a large set covering a large range, GA is applied to make discrete decision from a small set of choices. The decisions the GA makes include the choice of the activation function; whether or not to use batch normalization, dropout and max pooling; number of convolutional layers and dense layers; and the kernel size of convolutional layers. Table 4.1 shows the set of parameters, and the discrete set of values that they can take.

Table 7.1: Parameter Choices

Activation function	{“relu”, “leaky relu”, “sigm.”, “tanh”}
Batch norm.	{True, False}
Dropout	{True, False}
Max pooling	{True, False}
Num. of conv layers	{1,2,3}
Num. of dense layers	{1,2,3}
Kernel size	{3,5}

The parameter set of the network i for the GA has the following form:

$$\begin{aligned} p_i^{GA} =& [prm_i^{conv^1}, prm_i^{conv^2} ..., prm_i^{conv^C}, \\ & prm_i^{dense^1}, prm_i^{dense^2} ..., prm_i^{dense^D}], \\ & i \in \{1, 2, ..., n_m\} \end{aligned} \tag{7.1}$$

where C and D are the *maximum* number of possible convolutional layers and dense layers, respectively, n_m is the number of network models in the population and

$$\begin{aligned} prm_i^{conv^j} =& [1/0 \text{ (conv. layer exists or not), kernel size,} \\ & \text{activation func.}, 1/0 \text{ (for batch norm.),} \\ & 1/0 \text{ (for max pooling)}], \\ & i \in \{1, 2, ..., n_m\} \\ & j \in \{1, 2, ..., C\} \end{aligned} \tag{7.2}$$

$$
\begin{aligned}
prm_i^{dense^k} =& [1/0 \text{ (dense layer exists or not)}, \\
& \text{activation func.}, 1/0 \text{ (for batch norm.)}, \\
& 1/0 \text{(for dropout)}], \\
& i \in \{1, 2, ..., n_m\} \\
& k \in \{1, 2, ..., D\}.
\end{aligned}
\tag{7.3}
$$

For the discrete set of parameters given in Table 4.1, the values of C and D are 3.

First, n_m-many models are randomly initialized for the first network population N^1. This is done by choosing the values of convolutional and dense parameters, in (7.2) and (7.3), randomly, from the possible choices. Then, the number of filters for the convolutional layers, and the number of neurons for the fully connected layers are determined by the Bi-GAN as will be described in Section 7.1.1.

Then, the n_m models are trained, and evaluated to obtain their accuracy scores. Based on the accuracy scores, the GA is applied as detailed in Section 7.1.2.

7.1.1 Bi-GAN network

We propose a novel and modified generative adversarial network (GAN), referred to as Bi-GAN, to find the optimal network parameters, that have a large range of values. The proposed Bi-GAN network for refining different neural network parameters is shown in Fig. 7.2. It is composed of a generative part, an evaluation part and a discriminator. In contrast to traditional GANs, there are two generators (G_1 and G_2), two evaluators (E_1 and E_2), and one discriminator (D). The input to the two generators is Gaussian noise $z \sim p_{noise}(z)$. On the other hand, the input to the evaluators is the training data $x \sim p_{data}(x)$.

As will be discussed in more detail below, the generators G_1 and G_2 have the same network structure. From input noise $p_{noise}(z)$, G_1 and G_2 generate the input network parameters $G_1(z)$ and $G_2(z)$ to be used and evaluated by E_1 and E_2, respectively. E_1 and E_2 have the structure of the neural network whose parameters are being optimized or refined.

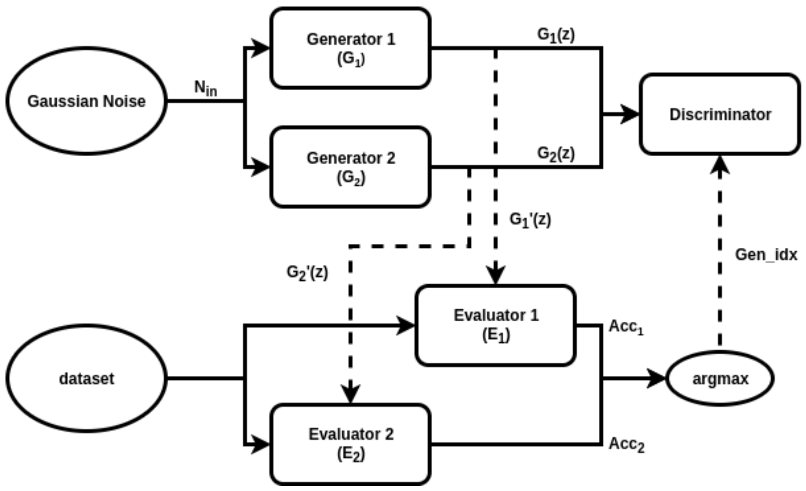


Figure 7.2: Proposed Bi-GAN incorporating two generators and one discriminator.

They calculate the classification accuracy on the training data x. $E_i(x, G_i(z)), i\epsilon\{1, 2\}$ represents the classification accuracy obtained by the evaluator E_i when the parameters $G_i(z)$ are used. The generator resulting in higher accuracy is marked as more accurate generator G_a, and the other generator is marked as G_b, where $a \in 1, 2, b =!a$.

We define the discriminator D as a network, which is used for binary classification between better and worse generator. $G_1(z)$ and $G_2(z)$ are fed into the discriminator D, and the ground truth label about which is the better generator comes from the evaluators. The discriminator D provides the gradients to train the worse performing generator.

Generative part

The two generators G_1 and G_2 have the same neural network structure shown in Fig. 7.3. Their input is a Gaussian noise vector z, and their outputs are $G_1(z)$ and $G_2(z)$. As seen in Fig. 7.3, generators are composed of fully connected layers with leaky relu activations. At the output layer, $tanh$ is employed so that $G_i^j(z) \in (-1, 1)$, where $j \in \{1, 2, ..., length(G_i(z))\}$ and $i \in \{1, 2\}$. Then, the range of $G_i(z)$ is changed from $(-1, 1)$ to (pm_{min}^j, pm_{max}^j) by using

$$G_i(z)' = [G_i(z) \times \frac{pm_{max} - pm_{min}}{2} + \frac{pm_{max} + pm_{min}}{2}]. \tag{7.4}$$

In (7.4), pm_{max} and pm_{min} are preset maxima and minima values, which are defined empirically based on values that a certain parameter can take, so that the value of the refined parameters can only change between pm_{max} and pm_{min}. For instance, in the case of the number of neurons for a fully connected layer, pm_{max} and pm_{min} are 4000 and 10, respectively. The re-scaled values $G_1(z)'$ and $G_2(z)'$ are then used as the parameters of evaluator networks. The length of $G_i(z)$ is determined by the number of network parameters that are refined, and is set at the generator network's last fully connected layer.

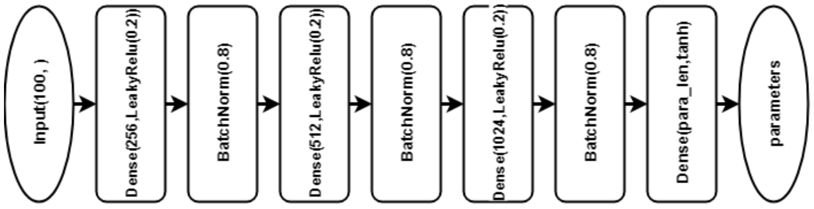


Figure 7.3: Generator network

Generators are trained/improved by the discriminator, which is a binary classifier used to differentiate the results from generator outputs $G_1(z)$ and $G_2(z)$. Labels "a" and "b" represent the generators with higher accuracy and lower accuracy results, respectively. The generator, which has the worse performance and is labeled by "b", is trained by stochastic gradient descent (SGD) from the discriminator to minimize $log(1 - D(G_b(z))$ by using

$$\nabla_{G_b} \frac{1}{m} \sum_{j=1}^{m} log(1 - D(G_b(z^{(j)}))), \tag{7.5}$$

where m is the number of epochs.

When $G'_a(z)$ becomes equal to $G'_b(z)$ for two consecutive iterations, the weights of G_b will be re-initialized to default random values. The purpose of this step is to prevent the optimization stopping at a local maxima and also prevent the vanishing tanh gradient problem.

Evaluation part

As mentioned above, one of the strengths of the proposed approach is that it can be used to refine/optimize parameters of different deep neural network structures. In other words, the evaluator networks have the same structure as the neural network whose parameters are being optimized or refined.

Evaluator networks are built by using the parameters $G_1(z)'$ and $G_2(z)'$ provided by the generators. The training data $x \sim p_{data}(x)$ is used to evaluate these network models. We employ an early stopping criteria. More specifically, if no improvement is observed in c epoches, the training is stopped.

We then obtain the accuracies $acc_i = \mathbb{E}_{x \sim p_{data}(x)} E_i(x)$, $i = \{1, 2\}$. Let a be the value of i resulting in higher accuracy, and $b =!a$. Then “a” is used as the ground truth label for the discriminator, which marks the generator with better parameters, and trains the worse generator G_b.

Discriminator

We define the discriminator D as a network (seen in Fig. 7.4, whose output is a scalar softmax output, which is used for binary classification between better generator and worse generator. $G_1(z)$ and $G_2(z)$ are fed into the discriminator D, and the ground truth label about which is better generator comes from the evaluators. Let $D(G(z))$ represent the probability that $G(z)$ came from the more accurate generator G_a rather than G_b. We train D to maximize the probability of assigning the correct label to the outputs $G_1(z)$ and $G_2(z)$ of both generators. Moreover, we simultaneously train the worse generator G_b to minimize $log(1 - D(G_b(z))$. The whole process can be expressed by:

$$min_{G_a} max_D \mathbb{E}_{z \sim p_z(z)}(log(D(G_a(z))) + log(1 - D(G_b(z)))), \tag{7.6}$$

where, $a = argmax_{i=\{1,2\}}(\mathbb{E}_{x \sim p_{data}(x)} E_i(x))$, $b =!a$.

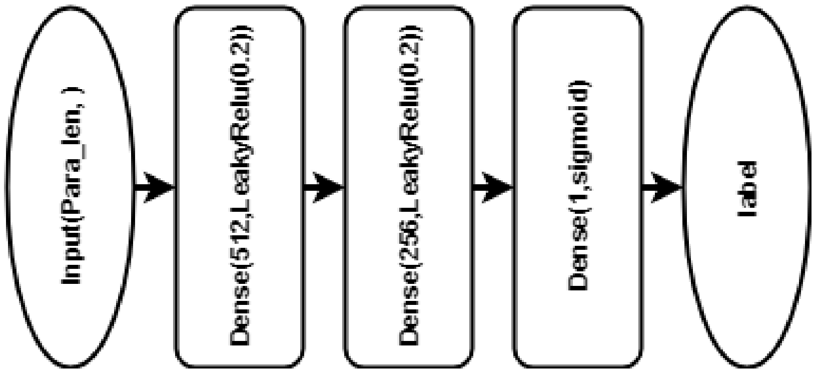


Figure 7.4: Discriminator network

The pseudo code for the proposed Bi-GAN is provided in Algorithm 1.

Algorithm 3 Bi-GAN Algorithm.

```
while in the iterations do
    Generate $m \times 2$ noise samples $\{Z_1^{(1)}, Z_1^{(2)}, ..., Z_1^{(m)}\}$ and $\{Z_2^{(1)}, Z_2^{(2)}, ..., Z_2^{(m)}\}$ from Gaussian white noise
    while j in range(m) do
        Build evaluators $E_1^{(j)}$ and $E_2^{(j)}$ based on parameters from $G_1(Z_1^{(j)})$ and $G_2(Z_2^{(j)})$
        Calculate $acc_1^j$ and $acc_2^j$ from $\mathbb{E}_{x \sim p_{data}(x)} E_i(x)$
        End if no acc. impr. after $c$ epoches
    end
    Calculate mean value $acc_i = (1/m) \sum_{j=1}^{m} acc_i^j, i = \{1, 2\}$
    Find $G_a$ as $G_{argmax(acc_1, acc_2)}$ and $G_b$ as the other one.
    Update Discriminator by SGD: $\bigtriangledown_D \frac{1}{m} \sum_{j=1}^{m} (log(D(G_a(z^{(j)}))) + log(1 - D(G_b(z^{(j)}))))$
    Update Generator $G_b$ by SGD: $\bigtriangledown_{G_b} \frac{1}{m} \sum_{j=1}^{m} log(1 - D(G_b(z^{(j)})))$
end
```

7.1.2 Genetic Algorithm

As mentioned above, we use a GA to make discrete decisions from a set of choices shown in Table 4.1. Within each GA evolution, our proposed Bi-GAN is used to set/optimize the values of the number of filters for the convolutional layers and the number of neurons for the dense layers. Then, the network models are trained and evaluated to obtain their accuracy scores. Based on the accuracy scores, the GA is applied.

Initial Population

The first generation of the networks, N^1, is generated randomly such that $N^1 = \{N_1, N_2, ..., N_{n_m}\}$, where n_m is the number of models. This is done by choosing the values of convolutional and dense parameters, in (7.2) and (7.3), randomly, from the possible choices.

Bi-GAN optimization

Our proposed Bi-GAN is used, as described in Sec. 7.1.1, to update the number of neurons for the fully connected layers, and the number of filters for the convolutional layers, of the n_m network models.

Evaluation

After the number of neurons for the fully connected layers, and the number of filters for convolutional layers are determined by Bi-GAN, each generated network model N_i ($i \in \{1, ..., n_m\}$) will be evaluated by the fitness function $fitness = f(N_i)$, which is a measure of the accuracy of each model. Models with better performance will have higher values. Thus, $E = \{E_1, E_2, ..., E_{n_m}\}$, will hold the fitness scores $E_i = f(N_i)$, where $i \in \{1, 2, ..., n_m\}$.

Selection

In the selection part, t-many top ranked models are selected from the $sorted(E)$ and r-many models are selected randomly from the rest of the network models. Then, d-many models are dropped in order to prevent over-fitting and getting stuck at a local optimum. The remaining selected models are the parent models (P), which will be used to create new models for the next generation.

Crossover and Mutation

Crossover is applied to generate n_m-many child network models from the parents. The choice of parents is performed as follows: Instead of always choosing two parents randomly from the parent pool, we associate a counter C_P with each parent P, and initialize it to zero. This counter is incremented by one each time a parent is used for crossover. First, two parents are selected randomly from the $t+r-d$ many parents. A new 'child' network is generated from the parents via crossover, and the counters of the parents are incremented by one. Then, two parents, whose counter is still zero, are selected randomly from the parent pool. Another network is generated from them via crossover, and the counters of the parents are incremented. If there is only one network model left with counter equal to zero, and the number of children is still less than n_m, then this model is chosen as one of the parents, and the other parent is chosen randomly from the rest of the models who have a counter value of one. If there are no more parents left with counter equal to zero, and the number of children is still less than n_m, then two parents, whose counter is one, are picked randomly, and their counter is incremented to two after crossover. This process is repeated until the number of children models reaches n_m.

The crossover between parent models a and b is performed as illustrated in Fig. 7.5. First two integers (ID$_1$ and ID$_2$) are picked randomly between 1 and C and 1 and D, respectively. Then, the parameters of the child network is set so that

$$\begin{aligned} p^{GA}_{child} = [& prm_a^{conv^1}, prm_a^{conv^2}, ..., prm_a^{conv^{ID_1}}, \\ & prm_b^{conv^{ID_1+1}}, ..., prm_b^{conv^C}, \\ & prm_a^{dense^1}, prm_a^{dense^2}, ..., prm_a^{dense^{ID_2}}, \\ & prm_b^{dense^{ID_2+1}}, ..., prm_b^{dense^D}]. \end{aligned} \tag{7.7}$$

After all the n_m-many child networks are obtained via crossover, 20% of the population is chosen randomly to perform mutation. As seen in (7.2) and (7.3), there are five different

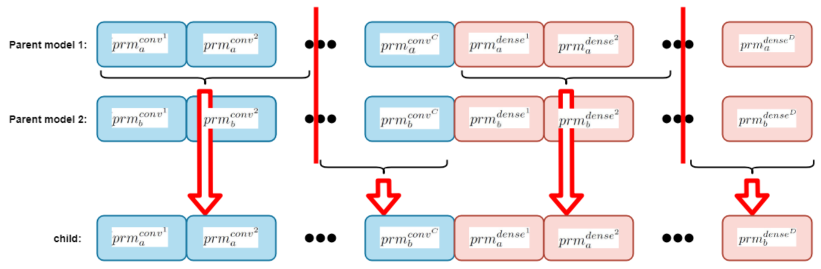


Figure 7.5: Crossover.

convolutional layer parameters, and four different dense layer parameters. Thus, there are $5*C+4*D$-many possible parameters that can be mutated. An integer is picked randomly between 1 and $5*C+4*D$, and the corresponding parameter type is chosen randomly from the possible choices in Table 4.1. For instance, if the random number corresponds to the filter size parameter, then its value is chosen randomly from $\{3,5\}$.

Then, the entire process is repeated by using this new population, updating the number of neurons and the number of filters for each network model in the population by using our propose Bi-GAN, and so on. The pseudo code for the entire process is provided in Algorithm 4.

Algorithm 4 GA-BiGAN algorithm.

```
Randomly initialize $n_m$ models for population $N^1$.
while $i^{th}$ iteration do
    Update hyper parameters by Bi-GAN (Alg. 3).
    Train and evaluate $N_1^i, N_2^i, ..., N_{n_m}^i$ by fitness function $f(N_j^i)$ and obtain scores $E$.
    Select $t$ top scored networks $N_{top} = N^i(argmax(E))$
    Randomly choose $r$ networks $N_{rand}$ from the rest of population $N^i$
    Merge $N_{top}$ and $N_{rand}$ and then drop $d$ networks ($N_{drop}$)
    Form $N_{parent} = (N_{top} \bigcup N_{rand}) - N_{drop}$
    Choose parents from $N_{parent}$ for crossover and generate $n_m$ new networks and add them
    to $N^{i+1}$
    Choose 20% of the networks in $N^{i+1}$, and perform mutation on them.
end
```

7.2 Experimental Results

Without loss of generality, we have applied the proposed approach on a 3D convolutional neural network by using the voxelized version of ModelNet40 dataset, which contains 3D CAD models of 40 object classes. Wu et al. [128] voxelized each object from the ModelNet at 12 different orientations (around gravity axis) for data augmentation. Voxel grids are $30 \times 30 \times 30$, and every object is fitted into this range. We used this pre-voxelized version of the dataset in our experiments. The dataset contains 40 subfolders for different objects. Each of these 40 subfolders contains 2 subfolders for training and testing. Train:test ratio differs for each object, but overall train:test ratio is around 3:1. Some example voxelized objects from the ModelNet40 dataset are shown in Fig. 7.6.

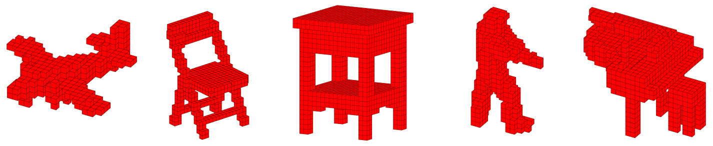


Figure 7.6: Sample voxelized objects from ModelNet40 dataset.

We have compared our proposed approach with two other approaches, which are based only on GAs and referred to as small-set GA, and large-set GA. The small-set GA is a basic genetic algorithm with limited number of choices. Large-set GA is given a larger set of choices for the number of neurons, and the number of filters. As for the activation function, batch normalization, dropout and max pooling decisions, the number of convolutional and dense layers, and the kernel size, the parameter choices are the same as in Table 4.1 for both small- and large-set GA. The difference between small- and large-set GA is the parameter choices for the number of neurons and the number of filters. For the small-set GA these choices are as follows:

Num. of neurons:	{16,32,64,128,256,512,1024,2048,4096}
Num. of kernels:	{1,4,16,64,256}

For the large-set GA, the number of neurons can be any integer between 16 and 4096, and the number of filters can be any integer between 1 and 256. In our proposed approach, the number of neurons and the number of filters are determined by our proposed Bi-GAN method from this continuous range.

These three approaches were run by using the same data, for the same amount of time to compare their performances. The parameters used in Algorithm 1 and 2 are as follows: For the Bi-GAN part, $m = 100$, and $c = 5$. For the GA part, the parameters used are $n_m = 25$, $t = 4$, $r = 2$ and $d = 1$. The results are summarized in Table 7.2. Same population size was used for all the GAs. As can be seen, our proposed approach provides the highest accuracy, and performs better compared to only GA-based approaches. Figure 7.7 shows the accuracy of each method over time. The method determines the number of neurons and the number of filters without requiring a discrete set of choices.

Table 7.2: Accuracy values obtained with different networks

	Accuracy
ShapeNet model	0.8417
Small-set GA	0.8294
Large-set GA	0.3641
method	**0.8520**

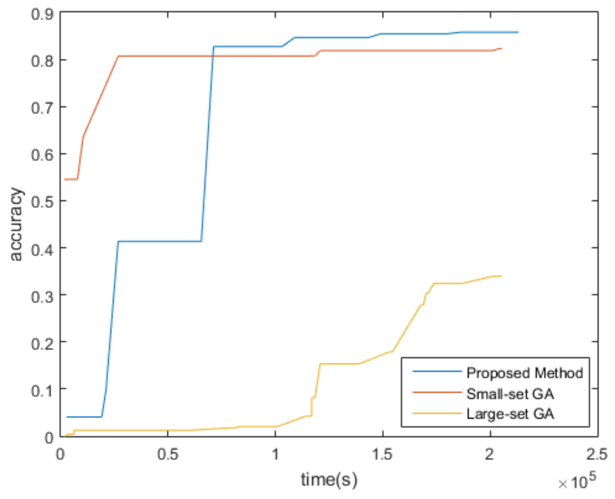


Figure 7.7: Accuracy of different network refinement approaches over time.

Another important point is the performance comparison with respect to Shapenets [128], which is a hand-crafted network with four convolutional layers, and two dense layers. Table

7.2 also includes the accuracy obtained when the Shapenet [128] model is used. As can be seen, by autonomously refining the network parameters, our proposed approach provides higher accuracy than the manually designed model by using a shallower network. The parameter values that each of the three approaches (method, small-set GA and large-set GA) ends up choosing/using are provided in Table 7.4 (the first entry is the filter size). As can be seen, the method achieves this higher accuracy of 85.2% by using only three convolutional layers as opposed to the four-convolutional-layer Shapenet model.

Iterations	5-pops		10-pops		20-pops	
	Small-set GA	Proposed	Small-set GA	Proposed	Small-set GA	Proposed
10	0.5813	0.6077	0.5945	0.6563	0.6409	0.7945
20	0.6969	0.7601	0.6183	0.7798	0.7689	0.7984
50	0.7626	0.8124	0.8069	0.8197	0.8113	0.8429

Table 7.3: Comparison of the method with the Small-set GA for different population sizes.

Table 7.4: Final Parameter Values

Approach	Final Parameter Values
Sm-set GA	[C1: 5,‘Relu’, NO Batch norm,Maxpool,64], [C2: 5,‘Relu’,No Batch,Max pool,64], [C3: 3,‘Relu’,0,1,256], [D1: Relu, No Batch, Dropout, 256], [D2: Leaky relu, no batch, no dropout, 64]
Lrg-set GA	[C1: 5,’Leaky Relu’,No Batch n., max pool,30], [C2: 3,‘Relu’,No Batch, Max pool, 31], [C3: 5,‘Relu’,No batch, No max pool, 9], [D1: Leaky Relu, Batch norm, Dropout, 31]
Prop.Meth.	[C1: 5,’Relu’,No Batch n,max pool,80], [C2: 3,‘Relu’,No Batch,Max pool, 105], [C3: 3,‘Leaky Relu’,No batch n.,Max pool,202], [D1: relu, no batch, no dropout, 601], [D2:leaky relu, no batch, dropout, 240]

Since small-set GA performs better than the large-set GA, in the remainder of the experiments, we compared our method with the small-set GA. We have tried three more population sizes while keeping the other parameter choices same as before. The results are summarized in Table 7.3. As can be seen, the method provides the higher accuracy rates for all different population sizes. Figures 7.8 and 7.9 show the change in accuracy and loss of these approaches, respectively, with each evolution (when population size is 20). As can be seen, the method performs better during its evolutions.

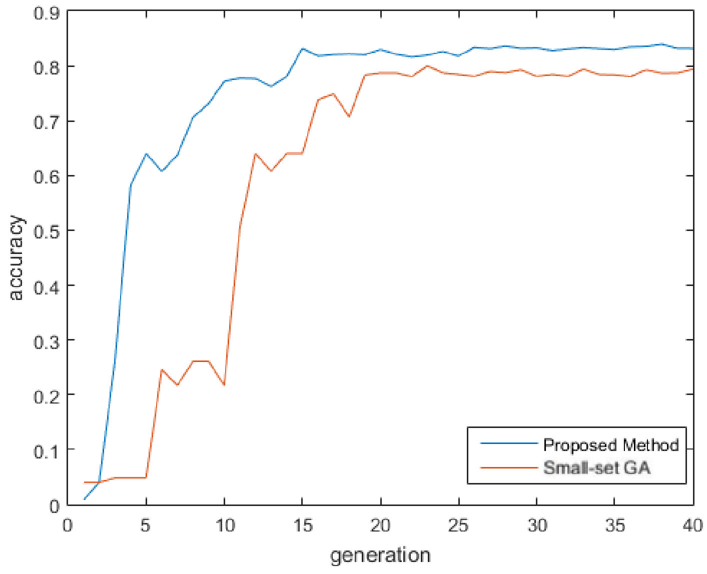


Figure 7.8: Accuracy with every evolution of the method and the Small-set GA.

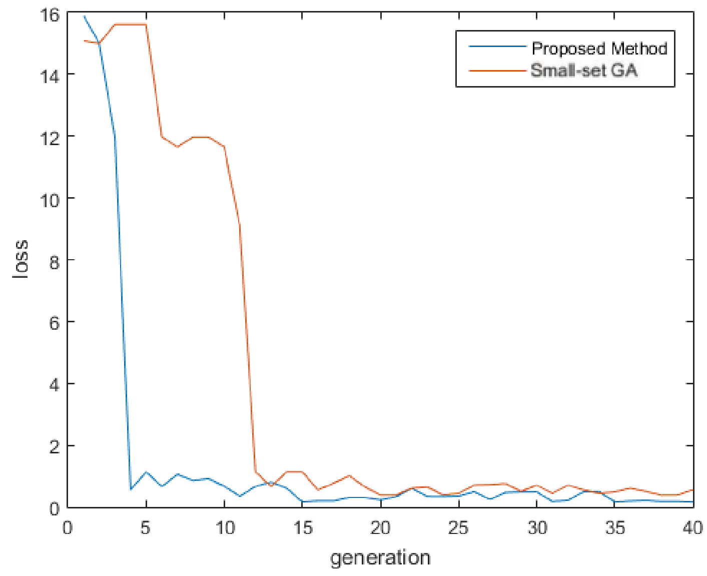


Figure 7.9: Loss with every evolution of the method and the Small-set GA.

We also performed one more experiment, where we let the maximum number of convolutional layers to be 5. In this experiment, the method achieved an accuracy of 86.62%.

7.3 Conclusion

In this chapter, we have presented a novel and systematic method that autonomously and simultaneously optimizes multiple parameters of any given deep neural network by using a genetic algorithm (GA) aided by a novel Bi-Generative Adversarial Network (GAN) with two generators, which is referred to as Bi-GAN. The proposed Bi-GAN allows the autonomous exploitation and choice of the number of neurons, for the fully-connected layers, and number of filters for the convolutional layers, from a large range of values. Our proposed approach can be used to autonomously refine the number of convolutional layers and dense layers, number and size of kernels, and the number of neurons; choose the type of the activation function; and decide whether to use dropout and batch normalization or not, to improve the accuracy of different deep neural network architectures. Without loss of generality, the method has been tested with the ModelNet database, and compared with the 3D Shapenets and two GA-only methods. The results show that the presented approach can simultaneously and successfully optimize multiple neural network parameters, and achieve increased accuracy even with shallower networks.

Chapter 8

Enhancing Cross-task Transferability of Adversarial Examples with Dispersion Reduction

To provide a strong baseline attack to evaluate the robustness of DNN models under the aforementioned threat model, we propose a new succinct method to generate adversarial examples, which transfer across a broad class of CV tasks, including classification, object detection, semantic segmentation, explicit content detection, and text detection and recognition. Our approach, called Dispersion Reduction (**DR**) and illustrated in Fig. 8.1, is inspired by the impact of "contrast" on an image's perceptibility. As lowering the contrast of an image would make the objects indistinguishable, we presume that reducing the "contrast" of an internal feature map would also degrade the recognizability of objects in the image, and thus could evade CV-based detections.

We use dispersion as a measure of "contrast" in feature space, which describes how scattered the feature map of an internal layer is. We empirically validate the impact of dispersion on model predictions, and find that reducing the dispersion of internal feature map would largely affect the activation of subsequent layers. Based on additional observation that lower layers detect simple features [129], we hypothesize that the low level features extracted by early convolution layers share many similarities across CV models. By reducing the dispersion of an internal feature map, the information that is in the feature output becomes indistinguishable or useless, and thus the following layers are not able to obtain any useful information no matter what kind of CV task is at hand. Thus, the distortions caused by dispersion reduction in feature space, are ideally suited to fool any CV model, whether designed for classification, object detection, semantic segmentation, text detection, or other vision tasks.

Based on these observations, we present and build the **DR** as a strong baseline attack to evaluate model robustness against black box attacks, which generate adversarial examples using simple and readily-available image classification models (e.g., VGG-16, Inception-V3 and ResNet-152), whose effects extend to a wide range of CV tasks. We evaluate DR attack on both popular open source detection and segmentation models, as well as commercially deployed detection models on four Google Cloud Vision APIs: classification, object detection, SafeSearch, and Text Detection (see §8.2). ImageNet, PASCAL VOC2012 and MS COCO2017 datasets are used for evaluations. The results show that DR attack causes larger drops on the model performance compared to the state-of-the-art attacks (MI-FGSM [88], DIM [89] and TI [130]) across different tasks. We hope our finding to raise alarms for real-world CV deployment in security-critical applications, and our simple yet effective attack to be used as a benchmark to evaluate model robustness.

8.1 Methodology

To construct AEs against a target model, we first establish a source model as the surrogate, to which we have access. Conventionally, the source model is established by training with examples labeled by the target model. That is, the inputs are paired with the labels generated from the target model, instead of the ground truth. In this way, the source model

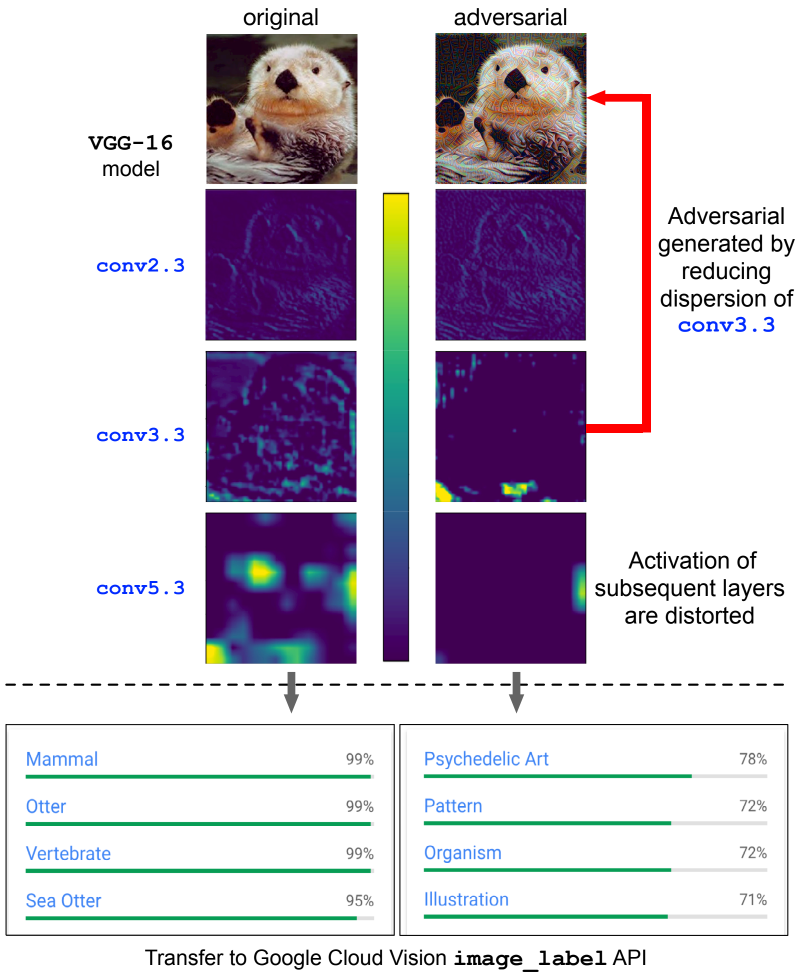


Figure 8.1: DR attack targets on the dispersion of feature map at a specific layer of feature extractors. The adversarial example generated by minimizing dispersion at `conv3.3` of VGG-16 model also distorts feature space of subsequent layers (e.g., `conv5.3`), and its effectiveness transfers to commercially deployed GCV APIs.

mimics the behavior of the target model. When we construct AEs against the source model, they are likely to transfer to the target model due to such connection.

In our framework, although a source model is still required, there is no need for training new models or querying the target model for labels. Instead, a pretrained public model could simply serve as the source model due to the strong transferability of the AEs generated via our approach. For example, in our experiments, we use pretrained VGG-16, Inception-v3 and Resnet-152, which are publicly available, as the source model f. With

Algorithm 5 Dispersion reduction attack

```
Input: A classifier $f$, original sample $x$, feature map at layer $k$; perturbation budget $\epsilon$
Input: Attack iterations $T$, learning rate $\ell$.
Output: An adversarial example $x'$ with $\| x' - x \|_\infty \leqslant \epsilon$
1: procedure DISPERSION REDUCTION
2:     $x'_0 \leftarrow x$ for $t = 0$ to $T - 1$ do
3:
        end
        Forward $x'_t$ and obtain feature map at layer $k$:
              $\mathcal{F}_k = f(x'_t)|_k$                                   (8.1)
4:     Compute standard deviation of $\mathcal{F}_k$: $g(\mathcal{F}_k)$
5:     Compute its gradient $w.r.t$ the input: $\bigtriangledown_x g(\mathcal{F}_k)$
6:     Update $x'_t$ by applying Adam optimization:
              $x'_t = x'_t - Adam(\bigtriangledown_x g(\mathcal{F}_k), \ell)$          (8.2)
7:     Project $x'_t$ to the vicinity of $x$:
              $x'_t = clip(x'_t, x - \epsilon, x + \epsilon)$                  (8.3)
8:     return $x'_t$
```

f as the source model, we construct AEs against it. Existing attacks perturb input images along gradient directions $\bigtriangledown_x J$ that depend on the definition of the task-specific loss function J, which not only limits their cross-task transferability but also requires ground-truth labels that are not always available. To mitigate these issues, we present dispersion reduction (DR) attack that formally defines the problem of finding an AE as an optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}'} g(f(\mathbf{x}', \theta)) \\ &s.t. \parallel \mathbf{x}' - \mathbf{x} \parallel_{\infty} \leqslant \epsilon \end{aligned} \tag{8.4}$$

where $f(\cdot)$ is a DNN classifier with output of intermediate feature map, and $g(\cdot)$ calculates the dispersion. The DR attack, detailed in Algorithm 5, takes a multi-step approach that creates an AE by iteratively reducing the dispersion of an intermediate feature map at layer k. Dispersion describes the extent to which a distribution is stretched or squeezed, and there can be different measures of dispersion, such as the standard deviation, and gini coefficient [131]. In this work, we choose standard deviation as the dispersion metric due to its simplicity, and denote it by $g(\cdot)$.

To explain why reducing dispersion could lead to valid attacks, we present a similar argument used in [132]. Consider a simplified model where $f(\mathbf{x}) = \mathbf{a} = (a1, \ldots, a_n)$ is the intermediate feature, and $\mathbf{y} = \mathbf{W}\mathbf{a}$ is an affine transformation of the feature (we omit the constant $\mathbf{b}$ for simplicity), resulting the final output logits $\mathbf{y} = (y_1, \ldots, y_k)$. In other words, we decompose a DNN classifier into a feature extractor $f(\cdot)$ and an affine transformation. Suppose the correct class is c, the logit y_c of a correctly classified example should be the largest, that is $\mathbf{w}_c^{\top}\mathbf{a} >> \mathbf{w}_i^{\top}\mathbf{a}$ for $i \neq c$, where $\mathbf{w}_i$ is the ith row of $\mathbf{W}$. This indicates $\mathbf{w}_c$ and $\mathbf{a}$ are highly aligned.

On the other hand, suppose our attack aims to reduce the standard deviation of the feature $\mathbf{a}$. The corresponding adversarial examples $\mathbf{x}'$ leads to a perturbed feature

$$\begin{aligned} f(\mathbf{x}') = \mathbf{a}' &\approx \mathbf{a} - \alpha \frac{\partial}{\partial \mathbf{a}} Std(\mathbf{a}) \\ &= \mathbf{a} - 2\alpha(\mathbf{a} - \bar{a}\mathbf{1})/(\sqrt{n-1} Std(\mathbf{a})) \end{aligned} \tag{8.5}$$

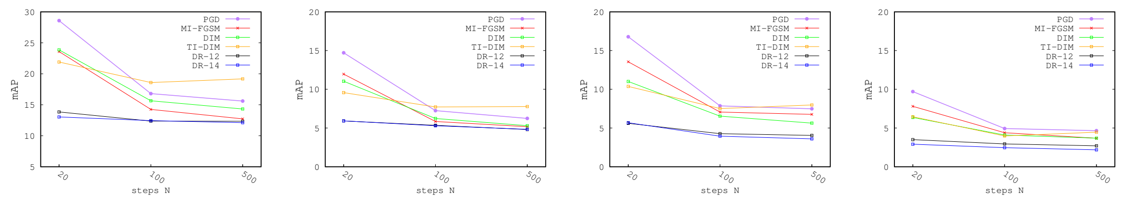
Where α depicts the magnitude of the perturbation on $\mathbf{a}$, $\bar{a}$ is the average of the entries of $\mathbf{a}$, and $\mathbf{1}$ is a column vector with 1 in each entry. Therefore, the change of the logit y_c due to adversarial perturbation is essentially

$$
\begin{aligned}
\Delta y_c &= -2\alpha(\mathbf{w}_c^\top \mathbf{a} - \mathbf{w}_c^\top \mathbf{1}\bar{a})/(\sqrt{n-1}Std(\mathbf{a})) \\
&= -2\alpha(\mathbf{w}_c^\top \mathbf{a} - n\bar{w}_c\bar{a})/(\sqrt{n-1}Std(\mathbf{a})) \\
&= -2\alpha\sqrt{n-1}Cov(\mathbf{w}_c, \mathbf{a})/Std(\mathbf{a}) < 0
\end{aligned}
\tag{8.6}
$$

If we think each entry of $\mathbf{a}$ and $\mathbf{w}_c$ as samples, the $Cov(\mathbf{w}_c, \mathbf{a})$ corresponds to the empirical covariance of these samples. This suggests that as long as $\mathbf{w}_c$ and $\mathbf{a}$ are aligned, our attack can always reduce the logit of the correct class. Note that α is approximately the product of the magnitude of the perturbation on $\mathbf{x}$ and the sensitivity of $f(\cdot)$, therefore the reduction of the logit could be large if $f(\cdot)$ is sensitive, which is often the case in practice.

In general, y_c could be any activation that is useful for the task, which may not be classification. As long as y_c is large for natural examples, indicating a certain feature is detected, it is always reduced by our attacks according to the analysis above. Thus, our attack is agnostic to tasks and the choice of loss functions.

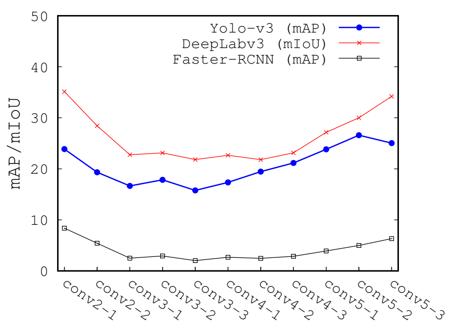
8.2 Experimental Results



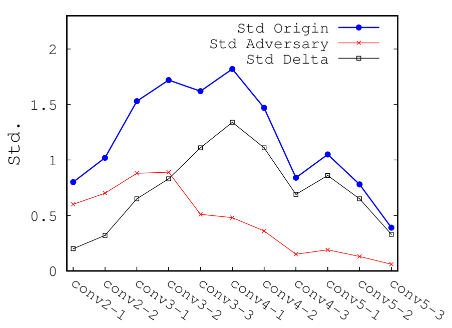
(a) SSD-Res50 (b) RetinaNet-Res50 (c) SSD-MobileNet (d) FasterRCNN-Res50

Figure 8.2: **Results of DR attack with different steps N.** We can see that our DR attack outperforms all baselines even starting from small steps (e.g. $N = 20$).

We compare the DR attack with the state-of-the-art black-box adversarial attacks on object detection and semantic segmentation tasks (using publicly available models), and commercially deployed Google Cloud Vision (GCV) tasks.



(a) mAP/mIoU results.



(b) Std. before and after attack

Figure 8.3: **Results of DR attack with different attack layers of VGG16.** We see that attacking the middle layers results in higher drop in the performance compared to attacking top or bottom layers. At the same time, in the attacking process, the drop in std of middle layers is also larger than the top and bottom layers. This motivates us that we can find a good attack layer by looking at the std drop during the attack.

8.2.1 Experimental Settings

Network Types: We consider Yolov3-DarkNet53 [133], RetinaNet-ResNet50 [134], SSD-MobileNetv2 [135], Faster R-CNN-ResNet50 [136], Mask R-CNN-ResNet50 [137] as the target object detection models and DeepLabv3Plus-ResNet101 [138], DeepLabv3-ResNet101 [139], FCN-ResNet101 [140] as the target semantic segmentation models. All network models are publicly available, and details are provided in the Appendix. The source networks for generating adversarial examples are VGG16, Inception-v3 and Resnet152 with output image sizes of (224×224), (299×299) and (224×224), respectively. For the evaluation on COCO2017 and PASCAL VOC2012 datasets, the mAP and mIoU are calculated as the evaluation metrics for detection and semantic segmentation, respectively. Due to the mismatch of different models being trained with different labeling systems (COCO / VOC), only 20 classes that correspond to VOC labels are chosen from COCO labels if a COCO pretrained model is tested on the PASCAL VOC dataset or a VOC pretrained model is tested on the COCO dataset. For the evaluation on ImageNet, since not all test images have the ground truth bounding boxes and pixelwise labels, the mAP and mIoU are calculated as the difference between the outputs of benign / clean images and adversarial images.

Implementation details: We compare the DR method with projected gradient descent

(PGD) [141], momentum iterative fast gradient sign method (MI-FGSM) [142], diverse inputs method (DIM) [143] and translation-invariant attacks (TI) [144]. As for the hyper-parameters, the maximum perturbation is set to be $\epsilon = 16$ for all the experiments with pixel values in [0, 255]. For the DR attacks, the step size $\alpha = 4$, and the number of training steps $N = 100$. For the baseline methods, we first follow the default settings in [143] and [144] with $\alpha = 1$ and $N = 20$ for PGD, MI-FGSM and DIM, $\alpha = 1.6$ and $N = 20$ for TI-DIM. Then, we apply the same hyper-parameters ($\alpha = 4$, $N = 100$) used with the DR method to all the baseline methods. For MI-FGSM, we adopt the default decay factor $\mu = 1.0$. For DIM and TI-DIM, the transformation probability is set to $p = 0.5$.

8.2.2 Diagnostics

The effect of training steps N

We show the results of attacking SSD-ResNet50, RetinaNet-ResNet50, SSD-MobileNet and Faster RCNN-ResNet50 with different number of training steps ($N = \{20, 100, 500\}$) based on MS COCO2017 validation set. We also compare the DR attack with multiple baselines, namely PGD, MI-FGSM, DIM, TI-DIM. The results are shown in Fig. 8.2. In contrast to the classification-based transfer attacks [88, 89, 130], we do not observe over-fitting in cross-task transfer attacks for all the tested methods. Therefore, instead of using $N = 20$, which is the value used by the baseline attacks we compare with, we can employ larger training steps (N=100), and achieve better attacking performance at the same time. In addition, we can see that our DR attack outperforms all the state-of-the-art baselines for all the step size settings. It should be noticed that DR attack is able to achieve promising results at $N = 20$, and the results from the DR attack, using 20 steps, are better than those of baseline methods using 500 steps. This shows that the DR attack has higher efficiency than the baselines.

		Yolov3 DrkNet mAP	RetinaNet ResNet50 mAP	SSD MobileNet mAP	Faster-RCNN ResNet50 mAP	Mask-RCNN ResNet50 mAP
		COCO/VOC	COCO/VOC	COCO/VOC	COCO/VOC	COCO/VOC
VGG16	PGD (α=1, N=20)	33.5 / 54.8	14.7 / 31.8	16.8 / 35.9	9.7 / 14.2	10.3 / 15.9
	PGD (α=4, N=100)	21.6 / 38.7	7.2 / 14.6	7.9 / 18.2	4.9 / 6.4	5.7 / 9.7
	MI-FGSM (α=1, N=20)	28.4 / 48.9	12.0 / 23.6	13.6 / 29.6	7.8 / 10.9	8.2 / 12.0
	MI-FGSM (α=4, N=100)	**19.0** / 35.0	5.8 / 10.6	7.0 / 19.1	4.4 / 5.0	4.8 / 7.1
	DIM (α=1, N=20)	26.7 / 46.9	11.0 / 21.9	11.0 / 22.9	6.4 / 8.2	7.2 / 11.6
	DIM (α=4, N=100)	20.0 / 37.6	6.2 / 13.0	6.5 / 14.9	4.1 / 5.0	4.6 / 6.7
	TI-DIM (α=1.6, N=20)	25.8 / 41.4	9.6 / 17.4	10.4 / 19.9	6.5 / 7.5	7.4 / 9.2
	TI-DIM (α=4, N=100)	19.5 / **33.4**	7.7 / 13.1	7.5 / 16.7	4.0 / 5.2	4.8 / 6.6
	DR (α=4, N=100)**(ours)**	19.8 / 38.2	**5.3 / 8.7**	**3.9 / 8.2**	**2.5 / 2.8**	**3.2 / 5.1**
Inc3	PGD (α=1, N=20)	46.8 / 67.5	23.9 / 51.8	25.2 / 47.4	27.0 / 45.7	27.5 / 48.7
	PGD (α=4, N=100)	35.3 / 57.1	15.0 / 33.0	14.0 / 31.6	18.2 / 31.7	19.4 / 34.8
	MI-FGSM (α=1, N=20)	42.0 / 63.9	20.0 / 44.3	20.9 / 43.5	22.8 / 39.3	23.7 / 42.9
	MI-FGSM (α=4, N=100)	32.4 / 54.0	12.5 / 27.1	13.1 / 29.2	16.3 / 26.9	17.9 / 30.5
	DIM (α=1, N=20)	32.5 / 54.5	12.9 / 27.5	13.9 / 29.7	14.2 / 24.0	16.3 / 27.7
	DIM (α=4, N=100)	29.1 / 48.3	10.4 / 20.5	10.4 / 22.0	12.2 / 18.2	13.8 / 44.6
	TI-DIM (α=1.6, N=20)	32.1 / 50.2	12.8 / 25.8	13.5 / 28.0	12.5 / 20.4	14.4 / 23.0
	TI-DIM (α=4, N=100)	27.1 / **42.2**	11.0 / 19.8	10.4 / 22.1	9.9 / 14.6	11.1 / 17.5
	DR (α=4, N=100)**(ours)**	**24.2** / 45.1	**8.5 / 18.9**	**9.0 / 19.5**	**8.3 / 14.3**	**9.8 / 17.0**
Res152	PGD (α=1, N=20)	39.4 / 62.0	19.1 / 42.9	19.9 / 41.6	13.8 / 19.4	15.0 / 22.0
	PGD (α=4, N=100)	28.8 / 51.5	12.2 / 25.9	11.2 / 24.4	8.2 / 11.3	8.8 / 13.9
	MI-FGSM (α=1, N=20)	35.1 / 58.1	15.8 / 36.2	16.7 / 35.8	11.1 / 16.3	12.2 / 18.1
	MI-FGSM (α=4, N=100)	26.4 / 48.2	11.2 / 23.5	9.9 / 21.3	7.0 / 9.5	8.2 / 11.4
	DIM (α=1, N=20)	28.1 / 50.3	12.2 / 26.3	11.0 / 23.9	7.0 / 10.6	7.9 / 12.6
	DIM (α=4, N=100)	24.7 / 43.2	8.8 / 19.4	7.8 / 16.1	5.1 / 7.1	6.2 / 10.3
	TI-DIM (α=1.6, N=20)	27.9 / 45.6	11.7 / 21.7	11.3 / 22.5	6.8 / 8.7	7.5 / 9.9
	TI-DIM (α=4, N=100)	**22.3 / 36.7**	9.0 / 15.8	8.7 / 19.1	5.0 / 6.6	5.7 / 8.2
	DR (α=4, N=100)**(ours)**	22.7 / 43.8	**6.8 / 12.4**	**4.7 / 7.6**	**2.3 / 2.8**	**3.0 / 4.5**

Table 8.1: **Detection results using validation images of COCO2017 and VOC2012 datasets.** The DR attack performs best on 25 out of 30 different cases and achieves 12.8 mAP on average over all the experiments. It creates 3.9 more drop in mAP compared to the best of the baselines (TI-DIM: 16.7 mAP).

The effect of attack layer

We show the results of attacking different convolutional layers of the VGG16 network with the DR attack based on PASCAL VOC2012 validation set. Fig. 8.3a, shows the mAP for Yolov3 and faster RCNN, and mIoU for Deeplabv3 and FCN. In Fig. 8.3b, we plot the standard deviation (std) values before and after the DR attack together with the change. As can be seen, attacking the middle layers of VGG16 results in higher drop in the performance compared to attacking top or bottom layers. At the same time, the change in std for middle layers is larger compared to the top and bottom layers. We can infer that for initial layers, the budget ϵ constrains the loss function to reduce the std, while for the layers near the output, the std is already relatively small, and cannot be reduced too much further. Based on this observation, we choose one of the middle layers as the target of the DR attack. More specifically, we attack conv3-3 for VGG16, the last layer of $group - A$ for inception-v3 and the last layer of 2nd group of bottlenecks(conv3-8-3) for ResNet152 in the following experiments.

8.2.3 Open Source Model Experiments

We compare the DR attack with the state-of-the-art adversarial techniques to demonstrate the transferability of our method on public object detection and semantic segmentation models. We use validation sets of ImageNet, VOC2012 and COCO2017 for testing object detection and semantic segmentation tasks. For ImageNet, 5000 correctly classified images from the validation set are chosen. For VOC and COCO, 1000 images from the validation set are chosen. The test images are shared in github repository: dispersion_reduction_test_images [145].

The results for detection and segmentation on COCO and VOC datasets are shown in Table 8.1 and Table 8.2, respectively. The results for detection and segmentation on the ImageNet dataset are provided in the Appendix. We also include the table for average results over all the datasets, including the ImageNet, in the Appendix.

As can be seen from Tables 8.1 and 8.2, (**DR**) achieves the best results on 36 out of 42 set of experiments by degrading the performance of the target model by a larger margin. For detection experiments, the **DR** attack performs best on 25 out of 30 different cases and for semantic segmentation 11 out of 12 different cases. For detection, DR attack achieves 12.8 mAP on average over all the experiments. It creates 3.9 more drop in mAP compared to the best of the baselines (TI-DIM: 16.7 mAP). For semantic segmentation, DR attack achieves 20.0 mIoU on average over all the experiments. It achieves 5.9 more drop in mIoU compared to the best of the baselines (DIM: 25.9 mIoU).

To summarize the results on the ImageNet dataset provided in the Appendix, (**DR**) achieves the best results in 17 out of 21 sets of experiments. For detection, DR attack achieves 7.4 relative-mAP on average over all the experiments. It creates 3.8 more drop in relative-mAP compared to the best of the baselines (TI-DIM: 11.2). For semantic segmentation, DR attack achieves 16.9 relative-mIoU on average over all the experiments. It achieves 4.8 more drop in relative-mIoU compared to the best of the baselines (TI-DIM: 21.7).

Seg. Results Using Val. Images of COCO and VOC Datasets		DeepLabv3 ResNet-101 mIoU	FCN ResNet-101 mIoU
		COCO/VOC	COCO/VOC
VGG16	PGD (α=1, N=20)	37.8 / 42.6	26.7 / 29.1
	PGD (α=4, N=100)	22.3 / 24.0	17.1 / 18.1
	MI-FGSM (α=1, N=20)	32.8 / 36.2	22.7 / 25.0
	MI-FGSM (α=4, N=100)	19.9 / 21.6	22.0 / 16.5
	DIM (α=1, N=20)	30.3 / 33.2	15.5 / 22.4
	DIM (α=4, N=100)	21.2 / 23.7	16.2 / 16.9
	TI-DIM (α=1.6, N=20)	29.9 / 31.1	21.9 / 23.0
	TI-DIM (α=4, N=100)	23.8 / 24.7	18.9 / 19.2
	DR (α=4, N=100)**(ours)**	**17.2 / 21.8**	**12.9 / 14.4**
IncV3	PGD (α=1, N=20)	49.4 / 56.0	36.8 / 40.1
	PGD (α=4, N=100)	37.1 / 41.3	26.1 / 28.3
	MI-FGSM (α=1, N=20)	44.2 / 51.1	32.4 / 35.4
	MI-FGSM (α=4, N=100)	33.7 / 39.1	24.0 / 35.4
	DIM (α=1, N=20)	35.7 / 40.4	24.9 / 27.2
	DIM (α=4, N=100)	30.4 / 33.9	21.3 / 22.3
	TI-DIM (α=1.6, N=20)	35.3 / 37.0	26.4 / 27.7
	TI-DIM (α=4, N=100)	29.0 / 29.8	22.5 / 23.5
	DR (α=4, N=100)**(ours)**	**23.2 / 29.2**	**17.1 / 20.9**
Res152	PGD (α=1, N=20)	45.2 / 50.2	30.7 / 34.6
	PGD (α=4, N=100)	31.5 / 35.1	21.6 / 24.0
	MI-FGSM (α=1, N=20)	39.9 / 43.9	26.4 / 29.9
	MI-FGSM (α=4, N=100)	28.2 / 32.2	19.9 / 22.1
	DIM (α=1, N=20)	31.3 / 35.5	22.3 / 23.9
	DIM (α=4, N=100)	25.9 / 28.8	19.0 / 19.9
	TI-DIM (α=1.6, N=20)	31.8 / 33.9	23.7 / 25.2
	TI-DIM (α=4, N=100)	26.6 / **26.6**	20.3 / 21.4
	DR (α=4, N=100)**(ours)**	**22.7** / 27.0	**16.4 / 17.6**

Table 8.2: **Semantic Segmentation results using validation images of COCO2017 and VOC2012 datasets.** DR attack performs best on 11 out of 12 different cases and achieves 20.0 mIoU on average over all the experiments. It achieves 5.9 more drop in mIoU compared to the best of the baselines (DIM: 25.9 mIoU).

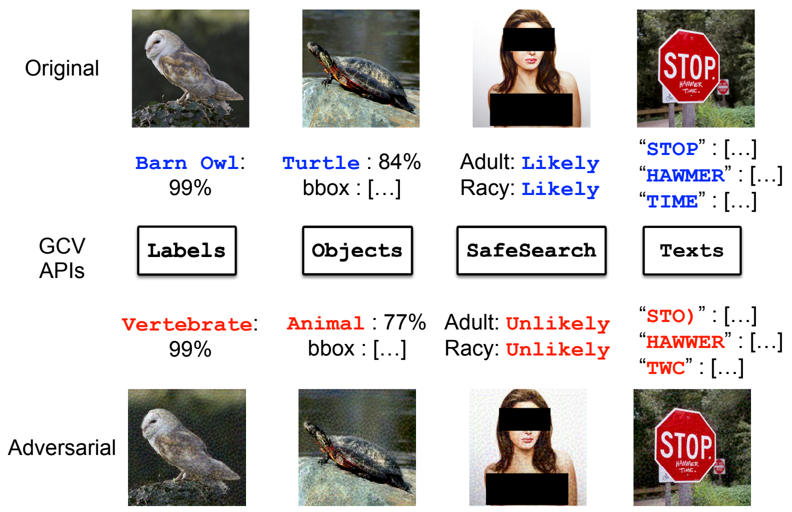

Original
Barn Owl: 99%
Turtle : 84% bbox : […]
Adult: Likely Racy: Likely
"STOP" : […] "HAWMER" : […] "TIME" : […]
GCV APIs
Labels
Objects
SafeSearch
Texts
Vertebrate: 99%
Animal : 77% bbox : […]
Adult: Unlikely Racy: Unlikely
"STO)" : […] "HAWWER" : […] "TWC" : […]
Adversarial

Figure 8.4: Visualization of images chosen from testing set and their corresponding AEs generated by DR. All the AEs are generated on VGG-16 `conv3.3` layer, with perturbations clipped by $l_\infty \leq 16$, and they effectively fool the four GCV APIs as indicated by their outputs.

8.2.4 Cloud API Experiments

We compare DR attack with the state-of-the-art adversarial techniques to enhance transferability on commercially deployed Google Cloud Vision (GCV) tasks [1]:

- Image Label Detection (**`Labels`**) classifies image into broad sets of categories.
- Object Detection (**`Objects`**) detects multiple objects with their labels and bounding boxes in an image.
- Image Texts Recognition (**`Texts`**) detects and recognize text within an image, which returns their bounding boxes and transcript texts.
- Explicit Content Detection (**`SafeSearch`**) detects explicit content such as adult or violent content within an image, and returns the likelihood.

Datasets. We use ImageNet validation set for testing `Labels` and `Objects`, and the NSFW Data Scraper [146] and COCO-Text [147] dataset for evaluating against `SafeSearch` and `Texts`, respectively. We randomly choose 100 images from each dataset for our eval-

[1]https://cloud.google.com/vision/docs

uation, and Fig. 8.4 shows sample images in our test set. Please note that due to the API query fees, larger scale experiments could not be performed for this part.

Model	Attack	`Labels`	`Objects`	`SafeSearch`	`Texts`	
		acc.	mAP (IoU=0.5)	acc.	AP (IoU=0.5)	C.R.W[2]
baseline (SOTA)[1]		82.5%	73.2	100%	69.2	76.1%
VGG-16	MI-FGSM	41%	42.6	62%	38.2	15.9%
	DIM	39%	36.5	57%	29.9	16.1%
	DR (**Ours**)	**23**%	**32.9**	**35**%	**20.9**	**4.1**%
Resnet-152	MI-FGSM	37%	41.0	61%	40.4	17.4%
	DIM	49%	46.7	60%	**34.2**	15.1%
	DR (**Ours**)	**25**%	**33.3**	**31**%	34.6	**9.5**%

[1] The baseline performance of GCV models cannot be measured due to the mismatch between original labels and labels used by Google. We use the GCV prediction results on original images as ground truth, thus the baseline performance should be 100% for all accuracy and 100.0 for mAP and AP. Here we provide state-of-the-art performance [148, 149, 147, 146] for reference.

[2] Correctly recognized words (C.R.W) [147].

Table 8.3: **The degraded performance of four Google Cloud Vision models, where we attack a single model from the left column.** DR attack degrades the accuracy of **`Lables`** and **`SafeSearch`** to 23% and 35%, the mAP of **`Objects`** and **`Texts`** to 32.9 and 20.9, the word recognition accuracy of **`Texts`** to only 4.1%, which outperform existing attacks.

Experiment setup. To generate the AEs, We use normally trained VGG-16 and Resnet-152 as our source models, since Resnet-152 is commonly used by MI-FGSM and DIM for generation [89, 88]. Since DR attack targets a specific layer, we choose `conv3.3` for VGG-16 and `conv3.8.3` for Resnet-152 as per the profiling result in Table 8.3 and discussion in Sec. 8.2.2.

Attack parameters. We follow the default settings in [88] with the momentum decay factor $\mu = 1$ when implementing the MI-FGSM attack. For the DIM attack, we set probability $p = 0.5$ for the stochastic transformation function $T(x; p)$ as in [89], and use the same decay factor $\mu = 1$ and total iteration number $N = 20$ as in the vanilla MI-FGSM. For DR attack, we do not rely on FGSM method, and instead use Adam optimizer ($\beta_1 = 0.98$, $\beta_2 = 0.99$) with learning rate of $5e^{-2}$ to reduce the dispersion of target feature map. The maximum perturbation of all attacks in the experiments are limited by clipping at $l_\infty = 16$, which is still considered less perceptible for human observers [150].

Evaluation metrics. We perform adversarial attacks only on single network and test them on the four black-box GCV models. The effectiveness of attacks is measured by the model performance under attacks. As the labels from original datasets are different from labels used by GCV, we use the prediction results of GCV APIs on the original data as the ground truth, which gives a baseline performance of 100% relative accuracy or 100.0 relative mAP and AP respectively.

Results. We provide the state-of-the-art results on each CV task as reference in Table 8.3. As shown in Table 8.3, DR outperforms other baseline attacks by degrading the target model performance by a larger margin. For example, the adversarial examples crafted by DR on VGG-16 model brings down the accuracy of **`Labels`** to only 23%, and **`SafeSearch`** to 35%. Adversarial examples created with the DR, also degrade mAP of **`Objects`** to 32.9% and AP of text localization to 20.9%, and with barely 4.1% accuracy in recognizing words. Strong baselines like MI-FGSM and DIM, on the other hand, only cause 38% and 43% success rate, respectively, when attacking `SafeSearch`, and are less effective compared with DR when attacking all other GCV models. The results demonstrate the better cross-task transferability of the dispersion reduction attack.

Figure 8.4 shows example of each GCV model's output for original and adversarial examples. The performance of **`Labels`** and **`SafeSearch`** are measured by the accuracy of classification. More specifically, we use top1 accuracy for **`Labels`**, and use the accuracy for detecting the given porn images as `LIKELY` or `VERY_LIKELY` being `adult` for **`SafeSearch`**. The performance of **`Objects`** is given by the mean average precision (mAP) at IoU=0.5. For **`Texts`**, we follow the bi-fold evaluation method of ICDAR 2017 Challenge [147]. We measure text localization accuracy using average precision (AP) of bounding boxes at IoU=0.5, and evaluate the word recognition accuracy with correctly recognized words (C.R.W) that are case insensitive.

When comparing the effectiveness of attacks on different generation models, the results that DR generates adversarial examples that transfer better across these four commercial

APIs still hold. The visualization in Fig. 8.4 shows that the perturbed images with $l_\infty \leq 16$ well maintain their visual similarities with original images, but fool the real-world computer vision systems.

8.3 Discussion and Conclusion

In this chapter, we present a Dispersion Reduction (DR) attack to improve the cross-task transferability of adversarial examples. Specifically, our method reduces the dispersion of intermediate feature maps by iterations. Compared to existing black-box attacks, the results on MS COCO, PASCAL VOC and ImageNet show that DR method performs better on attacking black-box cross-CV-task models. One intuition behind the DR attack is that by minimizing the dispersion of feature maps, images become ”featureless”. This is because few features can be detected if neuron activations are suppressed by perturbing the input (Fig. 8.1). Moreover, with the observation that low-level features bear more similarities across CV models, we hypothesize that the DR attack would produce transferable adversarial examples when one of the middle convolution layers is targeted. Evaluation on different CV tasks shows that this enhanced attack greatly degrades model performance by a large margin compared to the state-of-the-art attacks, and thus would facilitate evasion attacks against a different task model or even an ensemble of CV-based detection mechanisms. We hope that DR attack can serve as benchmark for evaluating robustness of future defense mechanisms.

8.4 Appendix

8.4.1 Target models

The backbones and datasets of pretrained weights for target models are shown in Table 8.4.

Models	Backbone	Pretrained Dataset
Yolov3[133][151]	DarkNet53	COCO
RetineNet[134][152]	ResNet50	COCO
SSD[135][153]	MobileNet	COCO
Faster R-CNN[136][154]	ResNet50	COCO
Mask R-CNN[137][154]	ResNet50	COCO
DeepLabv3[139][154]	ResNet101	sub COCO in VOC labels
FCN [140][154]	ResNet101	sub COCO in VOC labels

Table 8.4: Backbone and pretrained dataset for target models.

8.4.2 Experiments on ImageNet

We have performed adversarial attacks on randomly chosen 5000 correctly classified images from the ImageNet validation set. The accuracies for detection and segmentation are shown in Table 8.6 and Table 8.7, respectively. Since there are no ground truth annotations and masks for the test images, the performance metrics are selected as the relative mAP/mIoU for detection and semantic segmentation respectively. In other words, the predictions from benign samples are regarded as the ground truth and predictions from adversarial examples are regarded as inference results.

DR achieves the best results in 17 out of 21 sets of experiments (81.0%) by degrading the performance of the target model by a larger margin. For detection, DR attack reduces the mAP, on average, to 7.41 over all the experiments. It creates 3.8 more drop in mAP compared to the best of the baselines (TI-DIM: 11.2 mAP). For semantic segmentation, DR attack achieves 16.93 mIoU on average over all the experiments. It achieves 4.76 more drop in mIoU compared to the best of the baselines (DIM: 21.69 mIoU).

Avg. Res.	Det. mAP	Seg. mIoU
	COCO&VOC/ImageNet	
PGD	26.1 / 19.1	33.6 / 28.8
MI-FGSM	22.8 / 15.6	30.6 / 25.2
DIM	18.6 / 11.5	25.9 / 21.8
TI-DIM	16.7 / 11.2	26.4 / 21.7
DR (Ours)	**12.8 / 7.4**	**20.0 / 16.9**

Table 8.5: Average results for detection and segmentation using COCO, VOC and ImageNet validation images.

		Yolov3 DrkNet mAP	RetinaNet ResNet50 mAP	SSD MobileNet mAP	Faster-RCNN ResNet50 mAP	Mask-RCNN ResNet50 mAP
VGG16	PGD(α=1,N=20)	31.6	19.1	19.5	6.4	7.1
	PGD(α=4,N=100)	18.7	7.0	7.7	2.8	3.3
	MI-FGSM(α=1,N=20)	25.9	13.4	15.2	4.7	5.0
	MI-FGSM(α=4,N=100)	16.4	5.0	6.6	1.8	2.2
	DIM(α=1,N=20)	23.4	11.3	11.5	3.7	4.5
	DIM(α=4,N=100)	17.2	5.8	6.3	2.2	2.7
	TI-DIM(α=1.6,N=20)	21.5	10.2	11.6	3.5	4.0
	TI-DIM(α=4,N=100)	**16.3**	7.8	8.6	2.3	2.7
	DR(α=4,N=100)(**ours**)	17.0	**3.6**	**4.1**	**1.2**	**1.5**
InceptionV3	PGD(α=1,N=20)	51.3	36.6	33.9	25.9	25.1
	PGD(α=4,N=100)	33.3	16.4	16.2	14.1	14.7
	MI-FGSM(α=1,N=20)	44.6	27.4	27.5	19.8	20.1
	MI-FGSM(α=4,N=100)	30.3	14.1	15.3	11.9	12.5
	DIM(α=1,N=20)	30.6	15.2	16.4	11.0	11.7
	DIM(α=4,N=100)	25.3	10.2	10.6	6.9	8.2
	TI-DIM(α=1.6,N=20)	30.6	15.4	16.1	9.4	10.3
	TI-DIM(α=4,N=100)	23.7	11.2	12.2	6.8	7.0
	DR(α=4,N=100)(**ours**)	**21.1**	**8.6**	**9.4**	**4.5**	**5.3**
Resnet152	PGD(α=1,N=20)	40.8	27.6	27.0	10.4	10.8
	PGD(α=4,N=100)	27.2	13.4	13.0	5.0	6.1
	MI-FGSM(α=1,N=20)	33.9	20.3	21.2	7.6	8.0
	MI-FGSM(α=4,N=100)	24.6	11.4	11.8	3.9	4.7
	DIM(α=1,N=20)	26.9	13.2	13.0	4.4	5.3
	DIM(α=4,N=100)	22.2	9.3	8.7	2.9	3.7
	TI-DIM(α=1.6,N=20)	25.3	13.0	13.3	4.2	5.0
	TI-DIM(α=4,N=100)	**19.5**	9.4	9.8	2.7	2.9
	DR(α=4,N=100)(**ours**)	21.0	**6.2**	**4.8**	**1.3**	**1.6**

Table 8.6: Detection results for ImageNet.

		DeepLabv3 ResNet101 mIoU	FCN ResNet101 mIoU
VGG16	PGD(α=1,N=20)	30.3	24.6
	PGD(α=4,N=100)	17.5	15.1
	MI-FGSM(α=1,N=20)	25.4	20.8
	MI-FGSM(α=4,N=100)	**15.5**	13.9
	DIM(α=1,N=20)	24.7	19.0
	DIM(α=4,N=100)	17.1	14.5
	TI-DIM(α=1.6,N=20)	23.8	20.0
	TI-DIM(α=4,N=100)	18.3	16.5
	DR(α=4,N=100)(**ours**)	16.5	**12.4**
InceptionV3	PGD(α=1,N=20)	47.3	37.5
	PGD(α=4,N=100)	31.0	24.4
	MI-FGSM(α=1,N=20)	40.5	31.8
	MI-FGSM(α=4,N=100)	28.3	22.8
	DIM(α=1,N=20)	30.4	24.4
	DIM(α=4,N=100)	25.0	20.0
	TI-DIM(α=1.6,N=20)	28.1	24.4
	TI-DIM(α=4,N=100)	22.1	20.6
	DR(α=4,N=100)(**ours**)	**19.7**	**17.2**
Resnet152	PGD(α=1,N=20)	39.5	31.1
	PGD(α=4,N=100)	26.4	20.9
	MI-FGSM(α=1,N=20)	33.5	26.3
	MI-FGSM(α=4,N=100)	24.5	19.3
	DIM(α=1,N=20)	26.8	21.0
	DIM(α=4,N=100)	21.7	17.3
	TI-DIM(α=1.6,N=20)	26.2	21.9
	TI-DIM(α=4,N=100)	**20.1**	18.3
	DR(α=4,N=100)(**ours**)	20.5	**15.3**

Table 8.7: Segmentation Results for ImageNet.

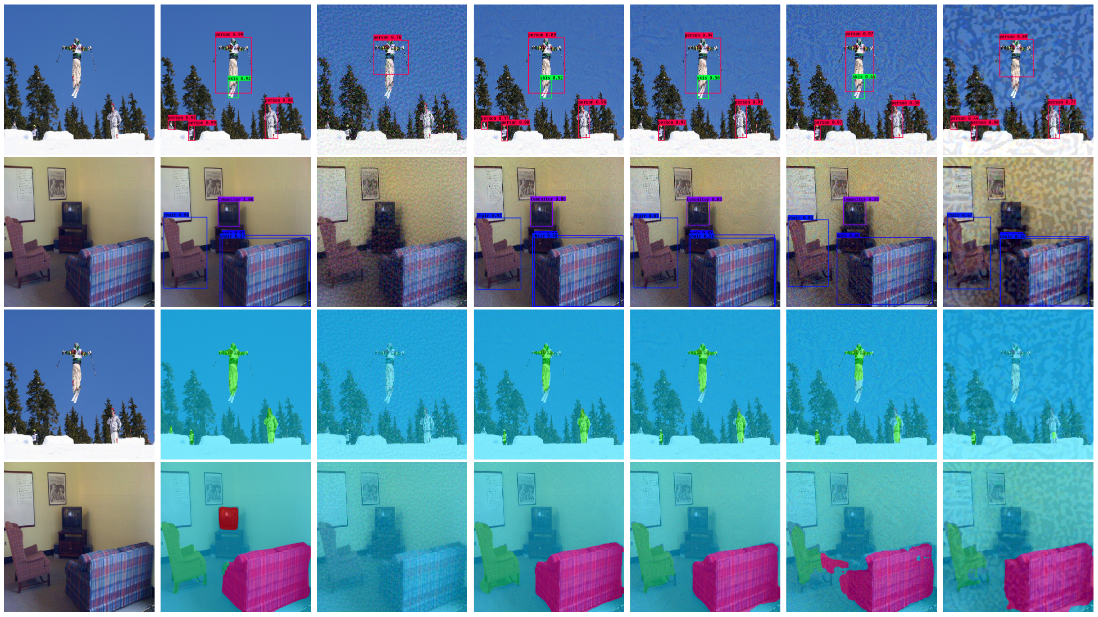

(a) Clean (b) Benign (c) DR (d) PGD (e) MIFGSM (f) DIM (g) TI-DIM

Figure 8.5: Samples of Detection and Segmentation Results

8.4.3 Average Results

We have compared DR attack with the state-of-the-art adversarial techniques to demonstrate the transferability of our method on public object detection and semantic segmentation models. We have used the validation sets of ImageNet, VOC2012 and COCO for testing object detection and semantic segmentation tasks. The average results can be seen in Table 8.5,

For COCO and VOC datasets, DR achieves the best results by degrading the performance of the target model by a larger margin. For detection, DR drops the mAP to 12.8 on average over all the experiments. It creates 3.9 more drop in mAP compared to the best of the baselines (TI-DIM: 16.7 mAP). For semantic segmentation, DR attack causes the mIoU to drop to 20.0 on average over all the experiments. It achieves 5.9 more drop in mIoU compared to the best of the baselines (DIM: 25.9 mIoU).

The diagnostic of average results for ImageNet can be seen in 8.4.2.

8.4.4 Visualization

Figure 8.5 shows the visualization samples for the DR method and baselines attacks. Examples of detection and segmentation results for clean images, results for benign images, DR images, PGD images, MI-FGSM images, DIM images and TI-DIM images are shown in each column (starting from left), respectively. First two rows are the detection results, and the last two rows are the segmentation results. We can see that the DR attack is able to effectively perform vanishing attack to both segmentation and detection tasks. It is also noted that the DR attack is more successful and effective, compared to the baselines, when attacking and degrading the performance for smaller objects.

Chapter 9

Robust Analysis of Multiple Object Tracking for Autonomous Driving

This chapter studies the adversarial machine learning attacks considering the *complete* visual perception pipeline in autonomous driving, i.e., both object detection and object tracking, and discover a novel attack technique, called *tracker hijacking*, that can effectively fool the MOT process using AEs on object detection. Our key insight is that although it is highly difficult to directly create a tracker for fake objects or delete a tracker for existing objects, we can carefully design AEs to attack the tracking error reduction process in MOT to deviate the tracking results of existing objects towards an attacker-desired moving direction. Such process is designed for increasing the robustness and accuracy of the tracking results, but ironically, we find that it can be exploited by attackers to substantially alter the tracking results. Leveraging such attack technique, successful AEs on as few as *one single frame* is enough to move an existing object in to or out of the headway of an autonomous vehicle and thus may cause potential safety hazards.

We select 20 out of 100 randomly sampled video clips from the Berkeley Deep Drive dataset to evaluate our attack technique. Under recommended MOT algorithm configurations in practice [109] and normal measurement noise levels, we find that our attack can

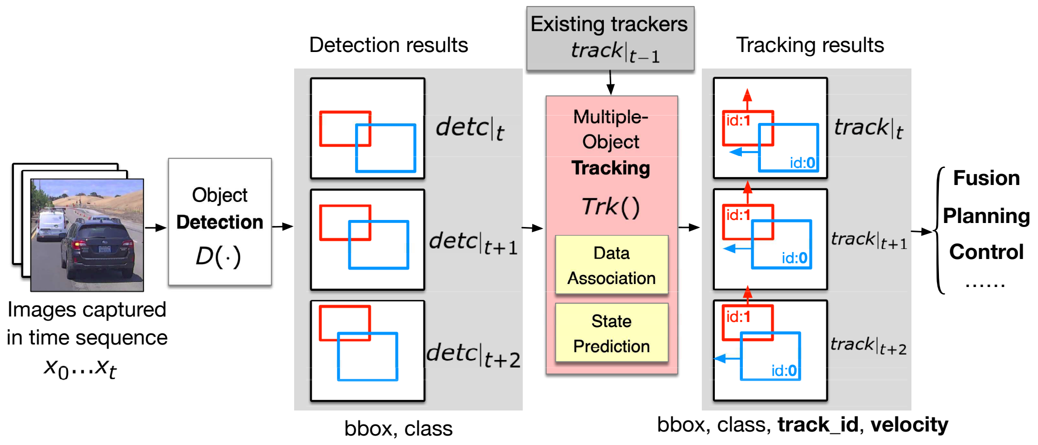
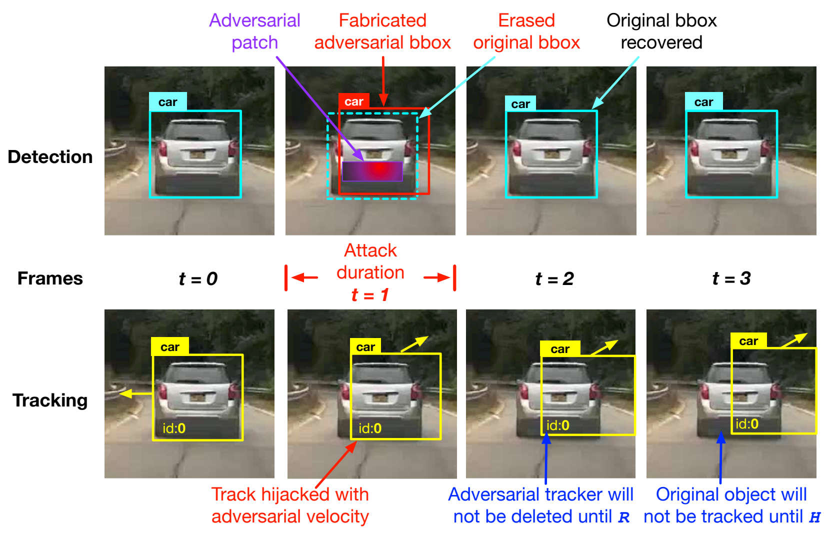


Figure 9.1: The complete visual perception pipeline in autonomous driving, i.e., both object detection and Multiple Object Tracking (MOT) [1, 2, 3, 4, 5, 6, 7].

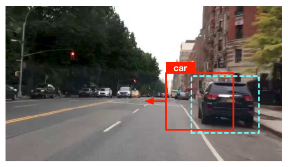
succeed with successful AEs on as few as one frame, and 2 to 3 consecutive frames on average. We also reproduce and compare with previous attacks that blindly target object detection, and find that when attacking 3 consecutive frames, our attack has a nearly 100% success rate while attacks that blindly target object detection only have up to 25%.

9.1 Method

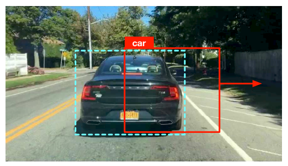
Overview. Fig. 9.2a illustrates the tracker hijacking attack discovered in this chapter, in which an AE for object detection (e.g., in the form of adversarial patches on the front car) that can fool the detection result for as few as one frame can largely deviate the tracker of a target object (e.g., a front car) in MOT. As shown, the target car is originally tracked with a predicted velocity to the left at t_0. The attack starts at time t_1 by applying an adversarial patch onto the back of the car. The patch is carefully generated to fool the object detector with two adversarial goals: (1) erase the bounding box of target object from detection result, and (2) fabricate a bounding box with similar shape that is shifted a little bit towards an attacker-specified direction. The fabricated bounding box (red one in detection result at t_1) will be associated with the original tracker of target object in the tracking result, which we call a *hijacking* of the tracker, and thus would give a fake velocity towards the


Adversarial patch
Fabricated adversarial bbox
Erased original bbox
Original bbox recovered
Detection
car
Frames
t = 0
Attack duration t = 1
t = 2
t = 3
Tracking
id:0
Track hijacked with adversarial velocity
Adversarial tracker will not be deleted until R
Original object will not be tracked until H

(a) Tracker hijacking attack overview



(b) Object move-in



(c) Object move-out

Figure 9.2: Description of the tracker hijacking attack flow (**a**), and two different attack scenarios: object move-in (**b**) and move-out (**c**), where tracker hijacking may lead to severe safety consequences including emergency stop and rear-end crashes.

attacker-desired direction to the tracker. The tracker hijacking shown in Fig. 9.2a lasts for only one frame, but its adversarial effects could last tens of frames, depending on the MOT parameter R and H. For example, at time t_2 after the attack, all detection bounding boxes are back to normal, however, two adversarial effects persist: (1) the tracker that has been hijacked with attacker-induced velocity will not be deleted until a reserved age (R) has passed, and (2) the target object, though is recovered in the detection result, will not be tracked until a hit count (H) has reached, and before that the object remains missing in the tracking result. However, it's important to note that our attack may not always succeed with one frame in practice, as the recovered object may still be associated with its original tracker, if the tracker is not deviated far enough from the object's true position during a short attack duration. Our empirical results show that our attack usually achieves a nearly 100% success rate when 3 consecutive frames are successfully attacked using AE (§9.2).

Such persistent adversarial effects may cause severe safety consequences in self-driving scenarios. We highlight two attack scenarios that can cause emergency stop or even a rear-end crashes:

Attack scenario 1: Target object move-in. Shown in Fig. 9.2b, an adversarial patch can be placed on roadside objects, *e.g.*, a parked vehicle to deceive visual perception of autonomous vehicles passing by. The adversarial patch is generated to cause a translation of the target bounding box towards the center of the road in the detection result, and the hijacked tracker will appear as a moving vehicle cutting in front in the perception of the victim vehicle. This tracker would last for 2 seconds if R is configured as $2\cdot$ fps as suggested in [109], and tracker hijacking in this scenario could cause an emergency stop and potentially a rear-end crash.

Attack scenario 2: Target object move-out. Similarly, tracker hijacking attack can also deviate objects in front of the victim autonomous vehicle away from the road to cause a crash as shown in Fig. 9.2c. Adversarial patch applied on the back of front car could deceive MOT of autonomous vehicle behind into believing that the object is moving out of its way, and the front car will be missing from the tracking result for a duration of $200ms$, if H uses the recommended configuration of $0.2\cdot$ fps [109]. This may cause the victim autonomous vehicle to crash into the front car.

9.1.1 Attack Methodology

Targeted MOT design. Our attack targets the most common MOT pipeline described in background. Specifically, we target first-order Kalman filter which predicts a state vector containing position and velocity of detected objects over time. For the data association, we adopt the mostly widely used Intersection over Union (IoU) as the similarity metric, and the IoU between bounding boxes are calculated by Hungarian matching algorithm [155] to solve the bipartite matching problem that associates bounding boxes detected in consecutive frames with existing trackers. Such combination of algorithms in the MOT is the most common in previous work [107, 108, 106] and real-world systems [1].

We now describe our methodology of generating an adversarial patch that manipulates detection results to hijack a tracker. As detailed in Alg.9.1, given a targeted video image

Algorithm 1 Tracker Hijacking Attack

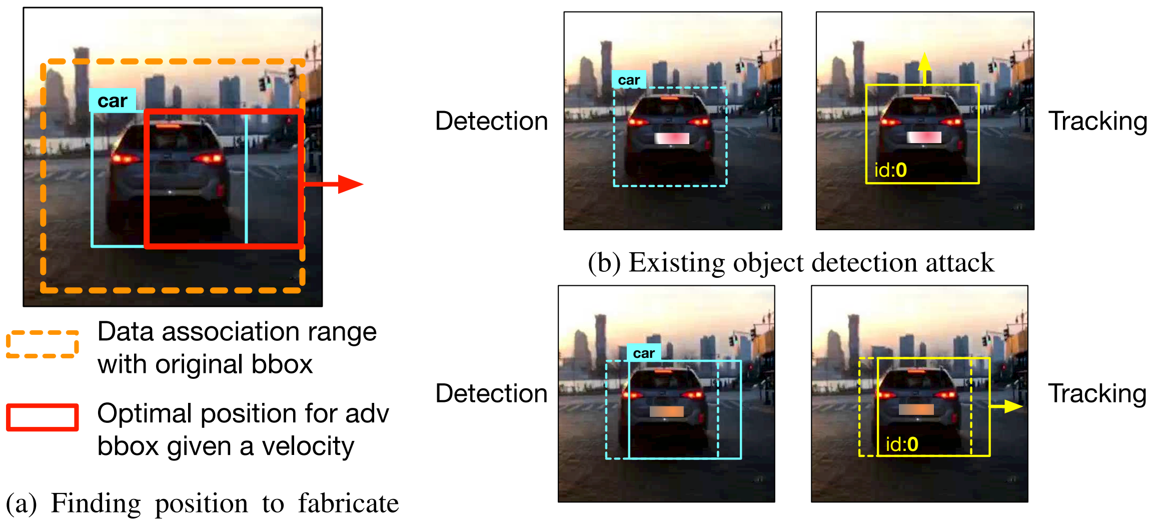
```
Input: Video image sequence $X = [x_0, x_1, ..., x_n]$; object detector $D(\cdot)$; MOT algorithm $Trk(\cdot)$;
Input: Index of target object to be hijacked $K$, attacker-desired directional velocity $\vec{v}$, adversarial
patch area as a mask matrix $patch$.
Output: Sequence of adversarial examples $X' = [x'_1, ..., x'_r]$ required for a successful attack.
Initialization $X' \leftarrow \{\}$, $detc|_0 \leftarrow D(x_0)$, $track|_0 \leftarrow \{current\_tracks\}$
 1: for $t = 1$ to $n$ do
 2:     $detc|_t \leftarrow D(x_t)$
 3:     if $detc|_t[K]$ matches $track|_{t-1}[K]$ then      ▷ target object matches with an existing tracker
 4:         find position $pos$ to place fabricated bbox with Eq. 1
                $pos \leftarrow$ FINDPOS$(Trk(\cdot), track|_{t-1}, K, \vec{v}, patch)$    see SuppAlg.1.1
 5:         generate adversarial frame $x'$ with Eq. 2    ▷ attack object detector with specialized loss
                $x'_t \leftarrow$ GENERATEADV$(x, D(\cdot), pos, K, patch)$    see SuppAlg.1.2
 6:         $X' \xleftarrow{+} x'_t$
 7:     else
 8:         return $X'$    ▷ attack succeeds when target object is not associated with original tracker
 9:     end if
10:     $track|_t \leftarrow Trk(track|_{t-1}, D(x'_t))$          ▷ update current tracker with adversarial frame
11: end for
```

sequence, the attack iteratively finds the minimum required frames to perturb for a successful track hijack, and generates the adversarial patches for these frames. In each attack iteration, an image frame in the original video clip is processed, and given the index of target objects K, the algorithm finds an optimal position to place the adversarial bounding box pos in order to hijack the tracker of target object by solving Eq. 9.1. The attack then constructs adversarial frame against object detection model with an adversarial patch, using Eq. 9.2 as the loss function to erase the original bounding box of target object and fabricate the adversarial bounding box at the given location. The tracker is then updated with the adversarial frame that deviates the tracker from its original direction. If the target object in the next frame is not associate with its original tracker by the MOT algorithm, attack has succeeded; otherwise, this process is repeated for the next frame. We discuss two critical steps in this algorithm below, and please refer to our supplementary material for the complete implementation of the algorithm.

Finding optimal position for adversarial bounding box. To deviate the tracker of a target object K, besides removing its original bounding box $detc|_t[K]$, the attack also needs to fabricate an adversarial box with a shift δ towards a specified direction. This turns into an optimization problem (Eq. 9.1) of finding the translation vector δ that maximizes the cost of Hungarian matching ($\mathcal{M}(\cdot)$) between the detection box and the existing tracker so that the bounding box is still associated with its original tracker ($\mathcal{M} \leq \lambda$), but the shift is large enough to give an adversarial velocity to the tracker. Note that we also limit the shifted bounding box to be overlapped with the $patch$ to facilitate adversarial example generation , as it's often easier for adversarial perturbations to affect prediction results in its proximity, especially in physical settings [99].

$$\begin{aligned} &\max_{\delta} \mathcal{M}(detc|_t[K] + \delta, track|_{t-1}[K]) \\ &s.t.\ \mathcal{M} \leq \lambda, IoU(detc|_t[K] + \delta, patch) \geqslant \gamma \end{aligned} \tag{9.1}$$

Generating adversarial patch against object detection. Similar to the existing adversarial attacks against object detection models [99, 93, 98], we also formulate the adversarial patch generation as an optimization problem shown in Eq. 9.2. Existing attacks without considering MOT directly minimize the probability of target class (e.g., a stop sign) to erase the target from detection result. However, as shown in Fig. 9.3b, such AEs are highly ineffective in fooling MOT as the tracker will still track for R frames even after the detection bounding box is erased. Instead, the loss function of our tracker hijacking attack incorporates two loss terms: $\mathcal{L}_1$ minimizes the target class probability at given location to erase the target bounding box, where $\sum_{i=0}^{B} \mathbb{1}_i^{obj}$ identifies all bounding boxes (B) before non-max suppression [156], who contain the center location (cx_t, cy_t) of pos, while C_i is the confidence score of bounding boxes; $\mathcal{L}_2$ controls the fabrication of adversarial bounding box at given center location (cx_t, cy_t) with given shape (w_t, h_t) to hijack the tracker. In the implementation, we use Adam optimizer to minimize the loss by iteratively perturbing the pixels along the gradient directions within the patch area, and the generation process stops when an adversarial patch that satisfies the requirements is generated. Note that the fabrication loss $\mathcal{L}_2$ needs only to be used when generating the first adversarial frame in a sequence to give the tracker an attacker-desired velocity $\vec{v}$, and then λ can be set to 0


car
Data association range with original bbox
Optimal position for adv bbox given a velocity
Detection
car
id:0
Tracking
Detection
car
id:0
Tracking

(a) Finding position to fabricate adversarial bounding box

(b) Existing object detection attack

(c) Our tracker hijacking attack

Figure 9.3: Comparison between previous object detection attack and our tracker hijacking attack. Previous attack that simply erase the bbox has no impact on the tracking output (**b**), while tracker hijacking attack that fabricates bbox with carefully chosen position successfully redirects the tracker towards attacker-specified direction (**c**).

to only focus on erasing target bounding box similar to previous work. Thus, our attack wouldn't add much difficulty to the optimization. Details of our algorithm can be found in the supplementary material, and the implementation can be found at [157].

$$
\begin{aligned}
&\min_{\Delta \in patch} \mathcal{L}_1(x_t + \Delta) + \lambda \cdot \mathcal{L}_2(x_t + \Delta) \\
&\mathcal{L}_1 = \sum_{i=0}^{B} \mathbb{1}_i^{obj} \cdot [C_i^2 - CrossEntropy(p_i, class_t)] \\
&\mathcal{L}_2 = \sum_{i=0}^{B} \mathbb{1}_i^{obj} \cdot \{[(cx_i - cx_t)^2 + (cy_i - cy_t)^2] + [(\sqrt{w_i} - \sqrt{w_t})^2 + (\sqrt{h_i} - \sqrt{h_t})^2] \\
&\qquad + (1 - C_i)^2 + CrossEntropy(p_i, class_t)\}
\end{aligned}
\tag{9.2}
$$

9.2 Evaluation

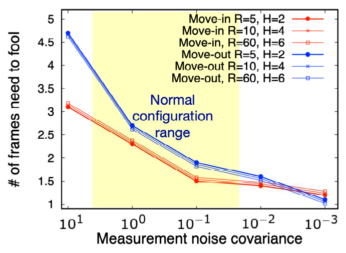
In this section, we describe our experiment settings for evaluating the effectiveness of our tracker hijacking attack, and comparing it with previous attacks that blindly attack object detection in detail.

9.2.1 Experiment Methodology

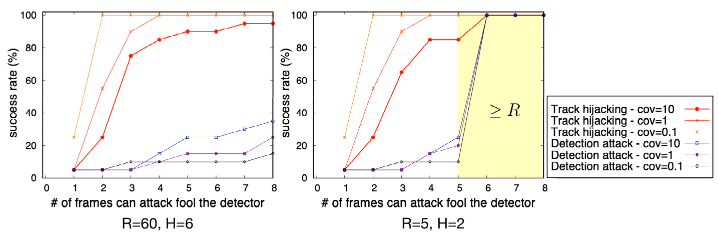
Evaluation metrics. We define a successful attack as that the detected bounding box of target object can no longer be associated with any of the existing trackers when attack has stopped. We measure the effectiveness of our track hijacking attack using the minimum number of frames that the AEs on object detection need to succeed. The attack effectiveness highly depends on the difference between the direction vector of the original tracker and adversary's objective. For example, attacker can cause a large shift on tracker with only one frame if choosing the adversarial direction to be opposite to its original direction, while it would be much harder to deviate the tracker from its established track, if the adversarial direction happens to be the same as the target's original direction. To control the variable, we measure the number of frames required for our attack in two previous defined attack scenarios: target object move-in and move-out. Specifically, in all move-in scenarios, we choose the vehicle parked along the road as target, and the attack objective is to move the tracker to the center, while in all move-out scenarios, we choose vehicles that are moving forward, and the attack objective is to move the target tracker off the road.

Dataset selection. We randomly sampled 100 video clips from Berkeley Deep Drive dataset [158], and then manually selected 10 suitable for the object move-in scenario, and another 10 for the object move-out scenario. For each clip, we manually label a target vehicle and annotate the patch region to be a small area at the back of it as shown in Fig. 9.3c. All videos have the same frame rate of 30 fps.

Implementation details. We implement our targeted visual perception pipeline using Python, with YOLOv3 [159] as the object detection model since it is among the most popular detectors used by real-time systems. For the MOT implementation, we use the Hungarian matching implementation called `linear_assignment` in the `sklearn` package for the data association, and we provide a reference implementation of Kalman filter based



(a) Frames required to be fooled for a successful tracker hijack



(b) Attack success rate at $R = 60$ $H = 6$, and $R = 5$, $H = 2$

Figure 9.4: In normal measurement noise covariance range (**a**), our tracker hijacking attack would require the adversarial example to fool only 2˜3 consecutive frames on average to successfully deviate the target tracker despite the (R, H) settings. Moreover we compare the success rate of tracker hijacking with previous adversarial attack against object detectors only under different attacker capabilities, *i.e.*, the number of consecutive frames the adversarial example can reliably fool the object detector (**b**). Tracker hijacking achieves superior attack success rate (100%) even by fooling as few as 3 frames, while previous attack is only effective when the adversarial example can reliably fools at least R consecutive frames.

on the one used in OpenCV [160].

The effectiveness of attack depends on a configuration parameter of Kalman filter, called measurement noise covariance (cov). cov is an estimation about how much noise is in the system, a low cov value would give Kalman filter more confidence on the detection result at time t when updating the tracker, while a high cov value would make Kalman filter to place trust more on its own previous prediction at time $t - 1$ than that at time t. We give a detailed introduction of configurable parameters in Kalman filter in §2 of our supplementary material. This measurement noise covariance is often tuned based on the performance of detection models in practice. We evaluate our approach under different cov configurations ranging from very small (10^{-3}) to very large (10) as shown in Fig. 9.4a, while cov is usually set between 0.01 and 10 in practice [1, 2].

9.2.2 Evaluation Results

Effectiveness of tracker hijacking attack. Fig. 9.4a shows the average number of frames that the AEs on object detection need to fool for a successful track hijacking over the

20 video clips in the evaluation. Although a configuration with $R = 60$ and $H = 6$ is recommended when fps is 30 [109], we still test different reserved age (R) and hit count (H) combinations as real-world deployment are usually more conservative and use smaller R and H [1, 2]. The results show that tracker hijacking attack only requires successful AEs on object detection in 2 to 3 consecutive frames on average to succeed despite the (R, H) configurations. We also find that even with a successful AE on only one frame, our attack still has 50% and 30% success rates when cov is 0.1 and 0.01 respectively.

Interestingly, we find that object move-in generally requires less frames compared with object move-out. The reason is that the parked vehicles in move-in scenarios (Fig. 9.2b) naturally have a moving-away velocity relative to the autonomous vehicle. Thus, compared to move-out attack, move-in attack triggers a larger difference between the attacker-desired velocity and the original velocity. This makes the original object, once recovered, harder to associate correctly, making hijacking easier.

Comparison with attacks that blindly target object detection. Fig. 9.4b shows the success rate of our attack and previous attacks that blindly target object detection, which we denote as *detection attack*. We reproduced the recent adversarial patch attack on object detection from Jia *et al.* [161] in 2018, which targets the autonomous driving context and has validated attack effectiveness using real-world car testing. In this attack, the objective is to erase the target class from the detection result of each frame. Evaluated under two (R, H) settings, we find that tracker hijacking attack achieves superior attack success rate (100%) even by attacking as few as 3 frames, while the detection attack needs to reliably fool at least R consecutive frames to guarantee success. When R is set to 60 according to the frame rate of 30 fps, the detection attack needs to have an adversarial patch that can constantly succeed at least 60 frames while the victim autonomous vehicle is driving. It translates to an over 98.3% ($\frac{59}{60}$) AE success rate, which has never been achieved or even got close to in previous work [98, 94, 99, 96]. Note that the detection attack still can have up to ˜25% success rate before R. This is because the detection attack causes the object to disappear for some frames, and when the vehicle heading changes during such disappearing period, it is still possible to cause the original object, when recovered, to misalign with the tracker predication in the original tracker. However, since our attack is designed to intentionally mislead the tracker predication in MOT, our success rate is substantially higher (3-4$\times$) and can reach 100% with as few as 3 frames attacked.

9.3 Discussion

Implications for future research in this area. Today, adversarial machine learning research targeting the visual perception in autonomous driving, no matter on attack or defense, uses the accuracy of objection detection as the *de facto* evaluation metric [102]. However, as concretely shown in our work, without considering MOT, successful attacks on the detection results alone do not have direct implication on equally or even closely successful attacks on the MOT results, the final output of the visual perception task in real-world autonomous driving [1, 2]. Thus, we argue that future research in this area should consider: (1) using the MOT accuracy as the evaluation metric, and (2) instead of solely focusing on object detection, also studying weaknesses specific to MOT or interactions between MOT and object detection, which is a highly under-explored research space today. This chapter marks the first research effort towards both directions.

Practicality improvement. Our evaluation currently are all conducted digitally with captured video frames, while our method should still be effective when applied to generate physical patches. For example, the adversarial patch generation method can be naturally combined with different techniques proposed by previous work to enhance reliability of AEs in the physical world (*e.g.*, non-printable loss [91] and expectation-over-transformation [162]). We leave this as future work.

Generality improvement. Though in this work we focused on MOT algorithm that uses IoU based data association, our approach of finding location to place adversarial bounding box is generally applicable to other association mechanisms (e.g., appearance-based matching). Our AE generation algorithm against YOLOv3 should also be applicable to other object detection models with modest adaptations. We plan to provide reference implementations of more real-world end-to-end visual perception pipelines to pave the way for future adversarial learning research in self-driving scenarios.

9.4 Conclusion

In this work, We are the first to study adversarial machine learning attacks against the complete visual perception pipeline in autonomous driving, i.e., both object detection and MOT. We discover a novel attack technique, tracker hijacking, that exploits the tracking error reduction process in MOT and can enable successful AEs on as few as one frame to move an existing object in to or out of the headway of an autonomous vehicle to cause potential safety hazards. The evaluation results show that on average when 3 frames are attacked, our attack can have a nearly 100% success rate while attacks that blindly target object detection only have up to 25%. The source code and data is all available at [157].

Our discovery and results strongly suggest that MOT should be systematically considered and incorporated into future adversarial machine learning research targeting the visual perception in autonomous driving. Our work initiates the first research effort along this direction, and we hope that it can inspire more future research into this largely overlooked research perspective.

Chapter 10

Boosting Ticket: Towards Practical Pruning for Adversarial Training with Lottery Ticket Hypothesis

We observe the standard technique introduced in [118] for identifying winning tickets does not always find boosting tickets. In fact, the requirements are more restrictive. We extensively investigate underlining factors that affect such boosting effect, considering three state-of-the-art large model architectures: VGG-16 [163], ResNet-18 [164], and WideResNet [165]. We conclude that the boosting effect depends principally on three factors: (i) learning rate, (ii) pruning ratio, and (iii) network capacity; we also demonstrate how these factors affect the boosting effect. By controlling these factors, after only one training epoch on CIFAR-10, we are able to obtain 90.88%/90.28% validation/test accuracy (regularly requires $>$30 training epochs) on WideResNet-34-10 when 80% parameters are pruned.

We further show that the boosting tickets have a practical application in accelerating adversarial training, an effective but expensive defensive training method for obtaining robust models against adversarial examples. Adversarial examples are carefully perturbed inputs that are indistinguishable from natural inputs but can easily fool a classifier [86, 132].

We first show our observations on winning and boosting tickets extend to the adversarial training scheme. Furthermore, we observe that the boosting tickets pruned from a weakly robust model can be used to accelerate the adversarial training process for obtaining a strongly robust model. On CIFAR-10 trained with WideResNet-34-10, we manage to save up to 49% of the total training time (including both pruning and training) compared to the regular adversarial training process.

10.1 Empirical Study of Boosting tickets

We first investigate boosting tickets on the standard setting without considering adversarial robustness. In this section, we show that with properly chosen hyperparameters, we are managed to find boosting tickets on VGG-16 and ResNet that can be trained much faster than the original dense network. Detailed model architectures and the setup can be found in Supplementary Section A.

10.1.1 Existence of Boosting Tickets

To find the boosting tickets, we use a similar algorithm for finding winning tickets, which is briefly described in the previous section and will be detailed here. First, a neural network is randomly initialized and saved in advance. Then the network is trained until convergence, and a given proportion of weights with the smallest magnitudes are pruned, resulting in a mask where the pruned weights indicate 0 and remained weights indicate 1. We call this train-and-prune step *pruning*. This mask is then applied to the saved initialization to obtain a sub-network, which are the boosting tickets. All of the weights that are pruned (where zeros in the mask) will remain to be 0 during the whole training process. Finally, we can retrain the sub-networks.

The key differences between our algorithm and the one proposed in [118] to find winning tickets are (i) we use a small learning rate for pruning and retrain the sub-network (tickets) with learning rate warm-up from this small learning rate. In particular, for VGG-16 we choose 0.01 for pruning and warmup from 0.01 to 0.1 for retraining; for ResNet-18 we choose 0.05 for pruning and warmup from 0.05 to 0.1 for retraining; (ii) we find it is sufficient to prune and retrain the model only once instead of iterative pruning for multiple times. In Supplementary Section B, we show the difference of boosting effects brought from the tickets found by iterative pruning and one-shot pruning is negligible. Note warmup is also used in [118]. However, they propose to use warmup from small learning rate to a large one during pruning as well, which hinders the boosting effect as shown in the following experiments.

First, we show the existence of boosting tickets for VGG-16 and ResNet-18 on CIFAR-10 in Figure 10.1 and compare to the winning tickets. In particular, we show the boosting tickets are winning tickets, in the sense that they outperform the randomly initialized models. When compared to the winning tickets, boosting tickets demonstrate equally good performance with a higher convergence rate. Similar results on MNIST can be found in Supplementary Section C.

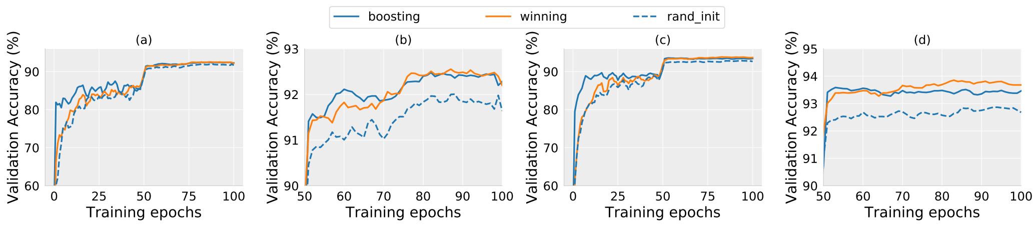


Figure 10.1: Validation accuracy during the training process on VGG-16 (a, b) and ResNet-18 (c, d) for winning tickets, boosting tickets, and randomly initialized weights. In both models, the boosting tickets show faster convergence rate and equally good performance as the winning tickets.

To measure the overall convergence rate, early stopping seems to be a good fit in the literature. It is commonly used to prevent overfitting and the final number of steps are used to measure convergence rates. However, early stopping is not compatible with learning rate scheduling we used in our case where the total number of steps is determined before training.

This causes two issues in our evaluation in Figure 10.1: (i) Although the boosting tickets reach a relatively high validation accuracy much earlier than the winning ticket, the training procedure is then hindered by the large learning rate. After the learning rate drops, the performance gap between boosting tickets and winning tickets becomes negligible. As a result, the learning rate scheduling obscures the improvement on convergence rates of boosting tickets; (ii) Due to fast convergence, boosting tickets tend to overfit, as observed in ResNet-18 after 50 epochs.

To mitigate these two issues without excluding learning rate scheduling, we conduct another experiment where we mimic the early stopping procedure by gradually increasing the total number of epochs from 20 to 100. The learning rate is still dropped at the 50% and 75% stage. In this way, we can better understand the speed of convergence without worrying about overfitting even with learning rate scheduling involved. In figure 10.2, we compare the boosting tickets and winning tickets in this manner on VGG-16.

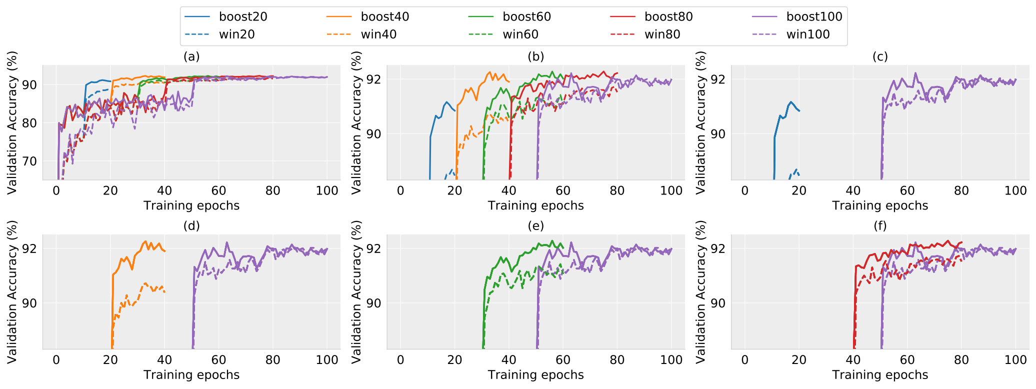


Figure 10.2: Validation accuracy when the total number of epochs are 20, 40, 60, 80, 100 for both the boosting tickets (straight lines) and winning tickets (dash lines) on VGG-16. Plot (a) and (b) contains the validation accuracy for all the training epochs in different scales. Plot (c,d,e,f) compare the validation accuracy between models trained for fewer epochs and the one for 100 epochs.

While the first two plots in Figure 10.2 show the general trend of convergence, the improvement of convergence rates is much clearer in the last four plots. In particular, the validation accuracy of boosting tickets after 40 epochs is already on pair with the one trained for 100 epochs. Meanwhile, the winning tickets fall much behind the boosting

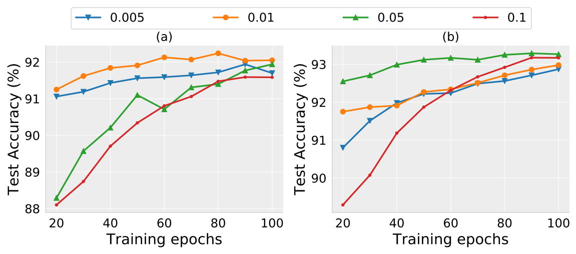


Figure 10.3: The final test accuracy achieved when total number of epochs vary from 20 to 100 on four different tickets. Each line denotes one winning ticket found by learning rate 0.005, 0.01, 0.05, and 0.1 for VGG-16 (a) and ResNet-18 (b).

tickets until 100 epochs where two finally match.

We further investigate the test accuracy at the end of training for boosting and winning tickets in Table 10.1. We find the test accuracy of winning tickets gradually increase as we allow for more training steps, while the boosting tickets achieve the highest test accuracy after 60 epochs and start to overfit at 100 epochs.

Table 10.1: Final test accuracy of winning tickets and boosting tickets trained in various numbers of epochs on VGG-16.

# of Epochs	20	40	60	80	100
Test Accuracy on Winning Tickets (%)	88.10	90.03	90.96	91.79	92.00
Test Accuracy on Boosting Tickets (%)	91.25	91.84	92.13	92.14	92.05

Summarizing the observations above, we confirm the existence of boosting tickets and state the boosting ticket hypothesis: *A randomly initialized dense neural network contains a sub-network that is initialized such that, when trained in isolation, converges faster than the original network and other winning tickets while matches their performance.*

In the following sections, we investigate three major components that affect the boosting effects.

10.1.2 Learning Rate

As finding boosting tickets requires alternating learning rates, it is natural to assume the performance of boosting tickets relies on the choice of learning rate. Thus, we extensively investigate the influence of various learning rates.

We use similar experimental settings in the previous section, where we increase the total number of epochs gradually and use the test accuracy as a measure of convergence rates. We choose four different learning rates 0.005, 0.01, 0.05 and 0.1 for pruning to get the tickets. All of the tickets found by those learning rates obtain the accuracy improvement over randomly reinitialized sub-model and thus satisfy the definition of winning tickets (i.e., they are all winning tickets).

As shown in the first two plots of Figure 10.3, tickets found by smaller learning rates tend to have stronger boosting effects. For both VGG-16 and ResNet-18, the models trained with learning rate 0.1 show the least boosting effects, measured by the test accuracy after 20 epochs of training. On the other hand, training with too small learning rate will compromise the eventual test accuracy at a certain extent. Therefore, we treat the tickets found by learning rate 0.01 as our boosting tickets for VGG-16, and the one found by learning rate 0.05 as for ResNet-18, which converge much faster than all of the rest while achieving the highest final test accuracy.

10.1.3 Pruning Ratio

Pruning ratio has been an important component for winning tickets [118], and thus we investigate its effect on boosting tickets. Since we are only interested in the boosting effect, we use the validation accuracy at early stages as a measure of the strength of boosting to avoid drawing too many lines in the plots. In Figure 10.4, we show the validation accuracy after the first and fifth epochs of models for different pruning ratios for VGG-16 and ResNet-18.

For both VGG-16 and ResNet-18, boosting tickets always reach much higher accu-

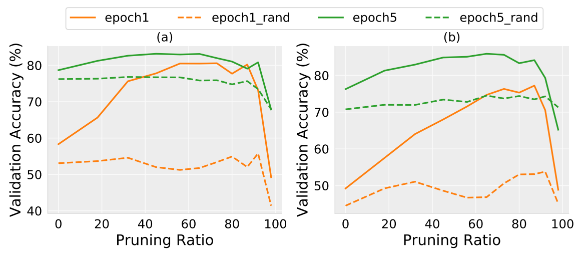


Figure 10.4: Under various pruning ratios, the changes of validation accuracy after the first and fifth training epoch, trained from the original initialized weights of boosting tickets and randomly reinitialized ones for VGG-16 (a) and ResNet-18 (b).

racy than randomly reinitialized sub-models, demonstrating their boosting effects. When the pruning ratio falls into the range from 60% to 90%, boosting tickets can provide the strongest boosting effects which obtain around 80% and 83% validation accuracy after the first and the fifth training epochs for VGG-16 and obtain 76% and 85% validation accuracy for ResNet-18. On the other hand, the increase of validation accuracy between the first training epoch and the fifth training epoch become smaller when boosting effects appear. It indicates their convergence starts to saturate due to the large learning rate at the initial stage and is ready for dropping the learning rate.

10.1.4 Model Capacity

We finally investigate how model capacity, including the depth and width of models, affects the boosting tickets. We use WideResNet [165] either with its depth or width fixed and vary the other factor. In particular, we keep the depth as 34 and increases the width from 1 to 10, comparing their boosting effect. Then we keep the width as 10 and increase the depth from 10 to 34. The changes of validation accuracy of the models are shown in Figure 10.5.

Overall, Figure 10.5 shows models with larger capacity have a more significant boosting effect, though the boosting effects keep the same when the depth is larger than 22. Notably, we find the largest model WideResNet-34-10 achieves 90.88% validation accuracy after only one training epoch.

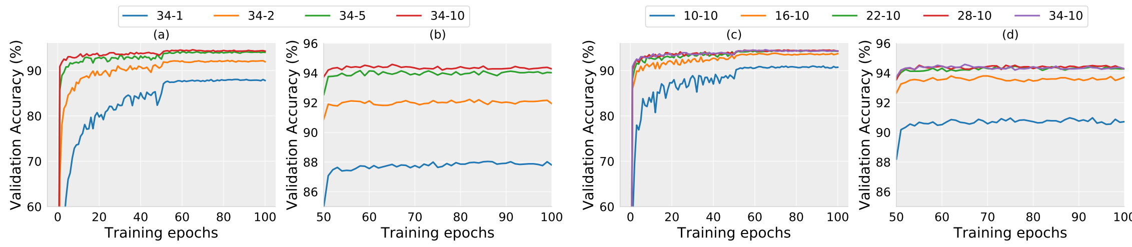


Figure 10.5: Plot (a) and (b) correspond to boosting tickets for various of model widths. Plot (c) and (d) correspond to boosting tickets for various of model depths. While a wider model always boosts faster, deep models have similar boosting effect when the depth is large enough.

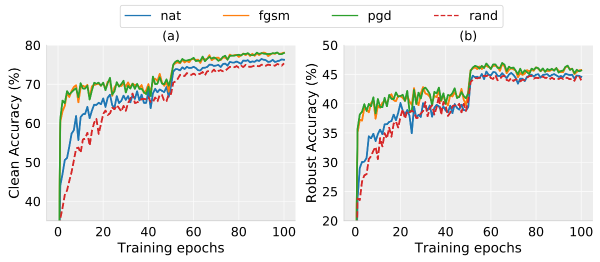


Figure 10.6: The clean accuracy (a) and robust accuracy (b) of pruned models on the validation set. The models are pruned based on different training methods (natural training, FGSM-based adversarial training, and PGD-based adversarial training). For each obtained boosting ticket, it is retrained with PGD-based adversarial training with 100 training epochs.

10.2 Boosting Tickets in Adversarial Settings

Although the lottery ticket hypothesis is extensively studied in [118] and [166], the same phenomenon in adversarial training setting lacks thorough understanding.

In this section, we show two important facts that make boosting tickets suitable for the adversarial scheme: (1) the lottery ticket hypothesis and boosting ticket hypothesis are applicable to the adversarial training scheme; (2) pruning on a weakly robust model allows to find the boosting ticket for a strongly robust model and save training cost.

10.2.1 Applicability for Adversarial Training

In the following experiment, we use a naturally trained model, that is trained in the standard manner, and two adversarially trained models using FGSM and PGD respectively to obtain the tickets by pruning these models. Then we retrain these pruned models with the same PGD-based adversarial training from the same initialization. In Figure 10.6, we report the corresponding accuracy on the original validation sets and on the adversarially perturbed validation examples, noted as clean accuracy and robust accuracy. We further train the pruned model from random reinitialization to validate lottery ticket hypothesis.

Unless otherwise stated, in all the PGD-based adversarial training, we keep the same setting as [167]. The PGD attacks are performed in 10 steps with step size $2/255$ (PGD-10). The PGD attacks are bounded by $8/255$ in its ℓ_∞ norm. For the FGSM-based adversarial training, the FGSM attacks are bounded by $8/255$.

Both models trained from the boosting tickets obtained with FGSM- and PGD-based adversarial training demonstrate superior performance and faster convergence than the model trained from random reinitialization. This confirms the lottery ticket hypothesis and boosting ticket hypothesis are applicable to adversarial training scheme on both clean accuracy and robust accuracy. More interestingly, the performance of the models pruned with FGSM- and PGD-based adversarial training are almost the same. This observation suggests it is sufficient to train a weakly robust model with FGSM-based adversarial training for obtaining the boosting tickets and retrain it with stronger attacks such as PGD.

This finding is interesting because FGSM-based adversarial trained models will suffer from label leaking problems as learning weak robustness [168]. In fact, the FGSM-based adversarially trained model from which we obtain our boosting tickets has 89% robust accuracy against FGSM but with only 0.4% robust accuracy against PGD performed in 20 steps (PGD-20). However, Figure 10.6 shows the following PGD-based adversarial retraining on the boosting tickets obtained from that FGSM-based trained model is indeed robust.

In [169], the authors argued that the lottery ticket hypothesis fails to hold in adversarial training via experiments on MNIST. We show they fail to observe winning tickets because the models they used have limited capacity. In the adversarial setting bounded by $L_\infty \leq$

0.3, small models such as a CNN with two convolutional layers used in [169] can not yield even winning tickets when pruning ratio is large. In Figure 10.7, plot (a) and (b) are the clean and robust accuracy of the pruned models when the pruning ratio is 80%. The pruned model degrades into a trivial classifier where all example are classified into the same class with 11.42%/11.42% valid/test accuracy. However, when we use VGG-16, as shown in plot (c) and (d), the winning tickets are found again. This can be explained as adversarial training requires much larger model capacity than standard training [167], thus pruning small models could undermine their performance. Since MNIST is a simple dataset, adversarial training converges quickly at the first few epochs for both the tickets and randomly initialized models. Therefore, there is no winning tickets performing obvious boosting effect which we can identify as a boosting ticket on MNIST.

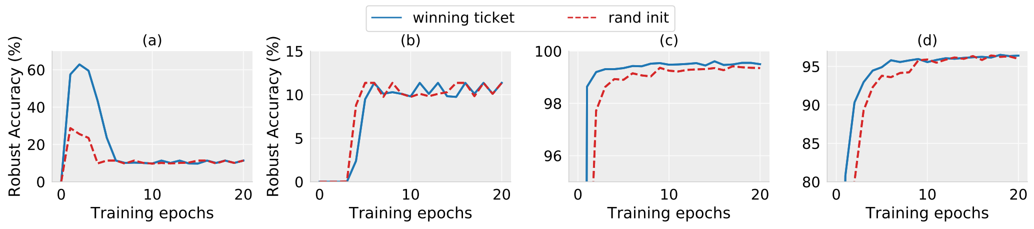


Figure 10.7: We show clean (a,c) and robust accuracy (b,d) for both winning tickets and randomly initialized weights on LeNet (a,b) and Vgg-16 (c,d) on MNIST with adversarial training.

10.2.2 Convergence Speedup

We then conduct the same experiment as in Figure 10.2 but in the adversarial training setting to better show the improved convergence rates. The results for validation accuracy and test accuracy are presented in Figure 10.8 and Table 10.2 respectively. It suggests it is sufficient to train 60 epochs to achieve similar robust accuracy as the full model trained for 100 epochs.

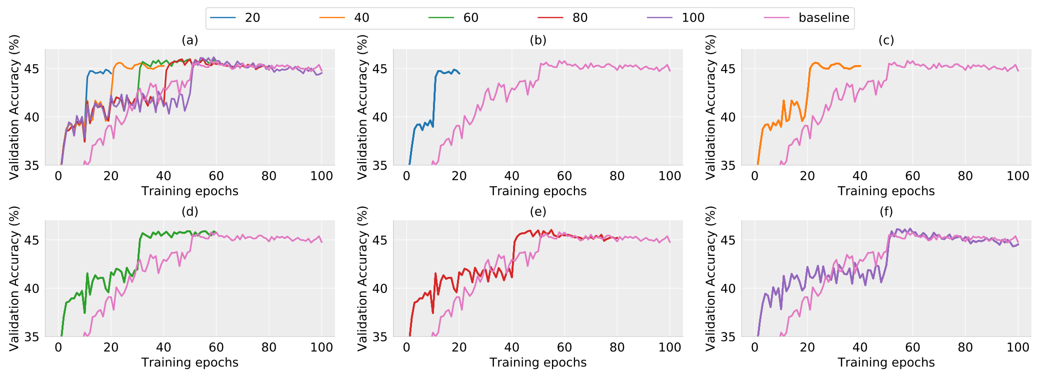


Figure 10.8: Validation robust accuracy of pruned models with PGD-based adversarial training on VGG-16 where the total number of epochs are 20, 40, 60, 80, 100 respectively. Plot (a) and (b) show all the results while plot (c), (d), (e), (f) compare each model with the baseline model. The baseline model is obtained by 100-epoch PGD-based adversarial training on the original full model.

Table 10.2: Best test clean and robust accuracy for PGD-based adversarial training on boosting tickets obtained by FGSM-based adversarial training in various numbers of epochs on VGG-16. Baseline model is obtained by 100-epoch PGD-based adversarial training on original full model.

# of Epochs	20	40	60	80	100	Baseline
Robust Test Accuracy	44.49	45.27	**45.73**	45.20	44.53	44.78
Clean Test Accuracy	75.15	76.28	76.48	77.60	**78.07**	77.21

10.2.3 Boosting Ticket Applications on adversarially trained WideResNet-34-10

Until now, we have confirmed that boosting tickets exist consistently across different models and training schemes and convey important insights on the behavior of pruned models. However, in the natural training setting, although boosting tickets provide faster convergence, it is not suitable for accelerating the standard training procedure as pruning to find the boosting tickets requires training full models beforehand. On the other hand, the two observations mentioned in Section 10.2 enable boosting tickets to accelerate adversarial training. In particular, we can find boosting tickets with FGSM-based adversarial training that and they can significantly accelerate the PGD-based adversarial training. Note that the cost of FGSM-based training is only 1/10 times of the standard 10-step PGD-based one and thus is almost negligible compared to the time saved due to the boosting effect.

In Table 10.3, we apply adversarial training to WideResNet-34-10, which has the same structure used in [167], with the presented approach for 40, 70 and 100 epochs and report the best accuracy/robust accuracy under various attacks among the whole training process. In particular, we perform 20-step PGD, 100-step PGD as white-box attacks where the attackers have the access to the model parameters. More experimental results are included in the Appendix.

Table 10.3: Best test clean accuracy, robust accuracy, and training time for PGD-based adversarial training on boosting tickets obtained by FGSM-based one in various numbers of epochs on WideResNet-34-10. Overall, our training strategy based on boosting tickets can save up to 49% of the total training time while performing better compared to regular adversarial training on the full model.

	Test Accuracy(%)			Consumed Time(s)			
Models	Clean	PGD-20	PGD-100	Pruning	Training	Total	Ratio
Madry's	86.21	50.07	49.32	-	134,764	134,764	-
Ours-40	87.72	50.37	49.28	15,462	54,090	**69,552**	**0.51**
Ours-70	**87.85**	**50.48**	**49.58**	15,462	94,796	110,258	0.82
Ours-100	87.35	49.92	49.11	15,462	137,105	152,567	1.13

We report the time consumption for training each model to measure how much time is saved by boosting tickets. We run all the experiments on a workstation with 2 V100 GPUs in parallel. From Table 10.3 we observe that while our approach requires pruning before training, it is overall faster as it uses FGSM-based adversarial training. In particular, to achieve its best robust accuracy, original Madry et al.'s training method [167] requires 134,764 seconds on WideResNet-34-10. To achieve that, our boosting ticket only requires 69,552 seconds, including 15,462 seconds to find the boosting ticket and 54,090 seconds to retrain the ticket, saving 49% of the total training time.

10.3 Conclusion

In this chapter, we investigate boosting tickets, sub-networks coupled with certain initialization that can be trained with significantly faster convergence rate. As a practical application, in the adversarial training scheme, we show pruning a weakly robust model allows to find boosting tickets that can save up to 49% of the total training time to obtain a strongly robust model that matches the state-of-the-art robustness. Finally, it is an interesting direction to investigate whether there is a way to find boosting tickets without training the full model beforehand, as it is technically not necessary.

Chapter 11

Conclusion

Most of the available devices and approaches for step counting rely only on accelerometer data, and thus are prone to over-counting. We have presented an autonomous and robust method for counting footsteps, and tracking and calculating stride length by using both accelerometer and camera data from smart phones or a Google$^{\text{TM}}$glass. To provide higher precision, instead of using a preset step and/or stride length, the presented method calculates the distance traveled with each step by using the camera data. The presented method has been compared with the commonly-used accelerometer-based step counter applications (apps). The results show that the presented method provides a significant increase in accuracy, and has the lowest average error rate both in number of steps taken and the distance traveled compared to commercially available, accelerometer-based step counters and apps.

Then, we have presented a robust and autonomous method to perform fine-grain activity classification by leveraging data from multiple sensor modalities, more specifically egocentric video and IMU sensor data from wearable devices. In contrast to many CNN-based approaches, we have proposed to use a capsule network to obtain features from egocentric video data. Instead of using a single CapsNet, multiple CapsNets are employed for consecutive images, and then a convolutional LSTM is used to build a recurrent CapsNet. The LSTM framework is employed both on IMU data and egocentric camera data to capture the temporal aspect of actions, which span a time window. Moreover, we proposed a GA-based approach to autonomously and systematically set the various parameters of our network architecture.

In order to optimize the performance of the ego-centric neural networks. We have presented a method to efficiently perform human activity classification from egocentric videos by incorporating actor-critic model of reinforcement learning. Actor-critic reinforcement learning allows placing a bounding box on a region of interest, and clipping that region. Then, only the clipped region is processed through a deeper network, while the entire image is processed by a shallow one. This strategically reduced complexity of the network structure provides significant increase in the processing speed, while maintaining the same level of accuracy.

Moreover, we have presented a novel and systematic method that autonomously and simultaneously optimizes multiple parameters of any given deep neural network by using a genetic algorithm (GA) aided by a novel Bi-Generative Adversarial Network (GAN) with two generators, which is referred to as Bi-GAN. The proposed Bi-GAN allows the autonomous exploitation and choice of the number of neurons, for the fully-connected layers, and number of filters for the convolutional layers, from a large range of values. The approach can be used to autonomously refine the number of convolutional layers and dense layers, number and size of kernels, and the number of neurons; choose the type of the activation function; and decide whether to use dropout and batch normalization or not, to improve the accuracy of different deep neural network architectures.

In order to study robustness of neural networks, we present a Dispersion Reduction (DR) attack to improve the cross-task transferability of adversarial examples. Specifically, our method reduces the dispersion of intermediate feature maps by iterations. Compared to existing black-box attacks, the results show that DR method performs better on attacking black-box cross-CV-task models. One intuition behind the DR attack is that by minimizing the dispersion of feature maps, images become ”featureless”. This is because few features can be detected if neuron activations are suppressed by perturbing the input (Fig. 8.1). Moreover, with the observation that low-level features bear more similarities across CV models, we hypothesize that the DR attack would produce transferable adversarial examples when one of the middle convolution layers is targeted. Evaluation on different CV tasks shows that this enhanced attack greatly degrades model performance by a large margin compared to the state-of-the-art attacks, and thus would facilitate evasion attacks against a different task model or even an ensemble of CV-based detection mechanisms. We hope that DR attack can serve as benchmark for evaluating robustness of future defense mechanisms.

For the aspect of tracking part in perception system, we are the first to study adversarial machine learning attacks against the complete visual perception pipeline in autonomous driving, i.e., both object detection and MOT. We discover a novel attack technique, tracker hijacking, that exploits the tracking error reduction process in MOT and can enable successful AEs on as few as one frame to move an existing object in to or out of the headway of an autonomous vehicle to cause potential safety hazards. Our discovery and results strongly suggest that MOT should be systematically considered and incorporated into future adversarial machine learning research targeting the visual perception in autonomous driving. Our work initiates the first research effort along this direction, and we hope that it can inspire more future research into this largely overlooked research perspective.

At last, towards higher neural network efficiency, we investigate boosting tickets, subnetworks coupled with certain initialization that can be trained with significantly faster convergence rate. As a practical application, in the adversarial training scheme, we show pruning a weakly robust model allows to find boosting tickets that can save training time to obtain a strongly robust model that matches the state-of-the-art robustness.

Bibliography

[1] Baidu, “Baidu Apollo,” https://github.com/ApolloAuto/apollo.

[2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, “Autoware on board: enabling autonomous vehicles with embedded systems,” in ICCPS’18. IEEE Press, 2018, pp. 287–296.

[3] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, “An open approach to autonomous vehicles,” IEEE Micro, vol. 35, no. 6, pp. 60–68, 2015.

[4] D. Zhao, H. Fu, L. Xiao, T. Wu, and B. Dai, “Multi-object tracking with correlation filter for autonomous vehicle,” Sensors, vol. 18, no. 7, p. 2004, 2018.

[5] A. Ess, K. Schindler, B. Leibe, and L. Van Gool, “Object detection and tracking for autonomous navigation in dynamic environments,” The International Journal of Robotics Research, vol. 29, no. 14, pp. 1707–1725, 2010.

[6] MathWorks, “Automated driving toolbox,” https://www.mathworks.com/products/automated-driving.html.

[7] Udacity, “Self-driving car engineer nanodegree program,” https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013.

[8] S. Zhang, P. McCullagh, C. Nugent, and H. Zheng, “Activity monitoring using a smart phone’s accelerometer with hierarchical classification,” in Intelligent

Environments (IE), 2010 Sixth International Conference on, July 2010, pp. 158–163.

[9] K. Ozcan and S. Velipasalar, "Wearable camera- and accelerometer-based fall detection on portable devices," in IEEE Embedded Systems Letters, 2015.

[10] L. Song, Y. Wang, J.-J. Yang, and J. Li, "Health sensing by wearable sensors and mobile phones: A survey," in e-Health Networking, Applications and Services (Healthcom), 2014 IEEE Int'l Conf. on, Oct 2014, pp. 453–459.

[11] F. Guo, Y. Li, M. S. Kankanhalli, and M. S. Brown, "An evaluation of wearable activity monitoring devices," in Proceedings of the 1st ACM International Workshop on Personal Data Meets Distributed Multimedia, ser. PDM '13. New York, NY, USA: ACM, 2013, pp. 31–34. [Online]. Available: http://doi.acm.org/10.1145/2509352.2512882

[12] K. Park, H. Shin, and H. Cha, "Smartphone-based pedestrian tracking in indoor corridor environments," Personal and Ubiquitous Computing, vol. 17, no. 2, pp. 359–370, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00779-011-0499-5

[13] N. Capela, E. Lemaire, and N. Baddour, "A smartphone approach for the 2 and 6-minute walk test," in Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE, Aug 2014, pp. 958–961.

[14] H.-J. Jang, J. Kim, and D. Hwang, "Robust step detection method for pedestrian navigation systems," Electronics Letters, vol. 43, no. 14, July 2007.

[15] D. Alvarez, R. Gonzalez, A. Lopez, and J. Alvarez, "Comparison of step length estimators from weareable accelerometer devices," in Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE, Aug 2006, pp. 5964–5967.

[16] M.-S. Pan and H.-W. Lin, "A step counting algorithm for smartphone users: Design and implementation," Sensors Journal, IEEE, vol. 15, no. 4, pp. 2296–2305, April 2015.

[17] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones," in Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ser. UbiComp '13. New York, NY, USA: ACM, 2013, pp. 225–234. [Online]. Available: http://doi.acm.org/10.1145/2493432.2493449

[18] M. Marschollek, M. Goevercin, K.-H. Wolf, B. Song, M. Gietzelt, R. Haux, and E. Steinhagen-Thiessen, "A performance comparison of accelerometry-based step detection algorithms on a large, non-laboratory sample of healthy and mobility-impaired persons," in Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE, Aug 2008, pp. 1319–1322.

[19] F. Aubeck, C. Isert, and D. Gusenbauer, "Camera based step detection on mobile phones," in Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on, Sept 2011, pp. 1–7.

[20] K. Ozcan and S. Velipasalar, "Robust and reliable step counting by mobile phone cameras," in Proceedings of the 9th International Conference on Distributed Smart Cameras, ser. ICDSC '15. New York, NY, USA: ACM, 2015, pp. 164–169. [Online]. Available: http://doi.acm.org/10.1145/2789116.2789120

[21] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam&#x2013;based navigation for autonomous micro helicopters in gps-denied environments," J. Field Robot., vol. 28, no. 6, pp. 854–874, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1002/rob.20412

[22] A. Bachrach, A. de Winter, R. He, G. Hemann, S. Prentice, and N. Roy, “Range - robust autonomous navigation in gps-denied environments,” in Robotics and Automation (ICRA), 2010 IEEE International Conference on, May 2010, pp. 1096–1097.

[23] L. Ojeda and J. Borenstein, “Personal dead-reckoning system for gps-denied environments,” in Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on, Sept 2007, pp. 1–6.

[24] A. Mannini and A. M. Sabatini, “Machine learning methods for classifying human physical activity from on-body accelerometers,” Sensors, vol. 10, no. 2, pp. 1154–1175, 2010. [Online]. Available: http://www.mdpi.com/1424-8220/10/2/1154

[25] ——, “Accelerometry-based classification of human activities using markov modeling,” Intell. Neuroscience, vol. 2011, pp. 4:1–4:10, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1155/2011/647858

[26] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, “Physical human activity recognition using wearable sensors,” Sensors, vol. 15, no. 12, pp. 31 314–31 338, 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/12/29858

[27] N. Abhayasinghe and I. Murray, “Human activity recognition using thigh angle derived from single thigh mounted imu data,” in 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Oct 2014, pp. 111–115.

[28] A. Bulling, U. Blanke, and B. Schiele, “A tutorial on human activity recognition using body-worn inertial sensors,” ACM Comput. Surv., vol. 46, no. 3, pp. 33:1–33:33, Jan. 2014. [Online]. Available: http://doi.acm.org/10.1145/2499621

[29] M. Zhang and A. A. Sawchuk, “Motion primitive-based human activity recognition using a bag-of-features approach,” in Proceedings of the 2Nd

ACM SIGHIT International Health Informatics Symposium, ser. IHI '12. New York, NY, USA: ACM, 2012, pp. 631–640. [Online]. Available: http://doi.acm.org/10.1145/2110363.2110433

[30] A. Bayat, M. Pomplun, and D. A. Tran, "A study on human activity recognition using accelerometer data from smartphones," Procedia Computer Science, vol. 34, pp. 450 – 457, 2014, the 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops. [Online]. Available: //www.sciencedirect.com/science/article/pii/S1877050914008643

[31] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," Sensors, vol. 16, no. 1, 2016. [Online]. Available: http://www.mdpi.com/1424-8220/16/1/115

[32] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li, "Large-scale video classification with convolutional neural networks," in Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[33] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in Comp. Vision and Pattern Recogn., 2015, pp. 677–691.

[34] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 4489–4497.

[35] A. Montes, A. Salvador, S. Pascual, and X. Giroinieto, "Temporal activity detection in untrimmed videos with recurrent neural networks," arXiv:1608.08128, 2017.

[36] S. Buch, V. Escorcia, C. Shen, B. Ghanem, and J. C. Niebles, "Sst: Single-stream temporal action proposals," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017, pp. 6373–6382.

[37] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," 12 2012.

[38] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, "Activitynet: A large-scale video benchmark for human activity understanding," in Computer Vision and Pattern Recognition, 2015, pp. 961–970.

[39] M. Cornacchia, K. Ozcan, Y. Zheng, and S. Velipasalar, "A survey on activity detection and classification using wearable sensors," IEEE Sensors Journal, vol. 17, no. 2, pp. 386–403, Jan 2017.

[40] C. Li and K. M. Kitani, "Pixel-level hand detection in ego-centric videos," in IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 3570–3577.

[41] Y. J. Lee, J. Ghosh, and K. Grauman, "Discovering important people and objects for egocentric video summarization," in Computer Vision and Pattern Recognition, 2012, pp. 1346–1353.

[42] M. S. Ryoo and L. Matthies, "First-person activity recognition: What are they doing to me?" in IEEE Conf. on CVPR, 2013, pp. 2730–2737.

[43] A. Fathi, X. Ren, and J. M. Rehg, "Learning to recognize objects in egocentric activities," in IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 3281–3288.

[44] K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto, "Fast unsupervised ego-action learning for first-person sports videos," in Computer Vision and Pattern Recognition, 2011, pp. 3241–3248.

[45] H. Pirsiavash and D. Ramanan, "Detecting activities of daily living in first-person camera views," in 2012 IEEE Conference on Computer Vision and Pattern Recognition, June 2012, pp. 2847–2854.

[46] T. McCandless and K. Grauman, "Object-centric spatio-temporal pyramids for egocentric activity recognition," in BMVC, 2013, pp. 30.1–30.11.

[47] Y. Li, A. Fathi, and J. M. Rehg, "Learning to predict gaze in egocentric video," in IEEE International Conference on Computer Vision, 2014, pp. 3216–3223.

[48] M. Moghimi, P. Azagra, L. Montesano, and A. C. Murillo, "Experiments on an rgb-d wearable vision system for egocentric activity recognition," in Computer Vision and Pattern Recognition Workshops, 2014, pp. 611–617.

[49] Z. Lu and K. Grauman, "Story-driven summarization for egocentric video," in Computer Vision and Pattern Recognition, 2013, pp. 2714–2721.

[50] T. H. Nguyen, J. C. Nebel, and F. Florezrevuelta, "Recognition of activities of daily living with egocentric vision: A review," Sensors, vol. 16, no. 1, p. 72, 2016.

[51] F. Conti, D. Palossi, R. Andri, M. Magno, and L. Benini, "Accelerated visual context classification on a low-power smartwatch," IEEE Transactions on Human-Machine Systems, vol. 48, no. 1, pp. 19–30, 2017.

[52] K. Zhan, S. Faux, and F. Ramos, "Multi-scale conditional random fields for first-person activity recognition on elders and disabled patients," Pervasive and Mobile Computing, vol. 16, Part B, pp. 251–267, 2015, selected Papers from the 12th Annual {IEEE} Int'l Conf. on Pervasive Computing and Communications (PerCom 2014).

[53] J. Windau and L. Itti, "Situation awareness via sensor-equipped eyeglasses," in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov 2013, pp. 5674–5679.

[54] E. H. Spriggs, F. D. L. Torre, and M. Hebert, "Temporal segmentation and activity classification from first-person sensing," in Computer Vision and Pattern Recogn. Workshops. IEEE Computer Society Conf. on, 2009, pp. 17–24.

[55] Y. Lu and S. Velipasalar, "Human activity classification from wearable devices with cameras," in Signals, Systems, and Computers, Asilomar Conf. on, Oct 2017.

[56] A. B. X. M. J. M. A. C. F. Torre, J. Hodgins and P. Beltran, "Guide to the carnegie mellon university multimodal activity (cmu-mmac) database," in Tech. report CMU-RI-TR-08-22, Robotics Institute, Carnegie Mellon University, April 2008.

[57] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, April 2002.

[58] P. Benardos and G.-C. Vosniakos, "Optimizing feedforward artificial neural network architecture," Engineering Applications of Artificial Intelligence, vol. 20, no. 3, pp. 365–382, 2007.

[59] L. Magnier and F. Haghighat, "Multiobjective optimization of building design using trnsys simulations, genetic algorithm, and artificial neural network," vol. 45, pp. 739–746, 03 2010.

[60] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," IEEE Transactions on Neural Networks, vol. 14, no. 1, pp. 79–88, Jan 2003.

[61] M. D. Ritchie, B. C. White, J. S. Parker, L. W. Hahn, and J. H. Moore, "Optimizationof neural network architecture using genetic programming improvesdetection and modeling of gene-gene interactions in studies of humandiseases," BMC bioinformatics, vol. 4, no. 1, p. 28, 2003.

[62] B. U. Islam, Z. Baharudin, M. Q. Raza, and P. Nallagownden, "Optimization of neural network architecture using genetic algorithm for load forecasting," in Intelligent and Advanced Systems (ICIAS), 2014 5th International Conference on. IEEE, 2014, pp. 1–6.

[63] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evolutionary computation, vol. 10, no. 2, pp. 99–127, 2002.

[64] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," arXiv preprint arXiv:1703.00548, 2017.

[65] B. I. Rylander, "Computational complexity and the genetic algorithm," Ph.D. dissertation, Moscow, ID, USA, 2001, aAI3022336.

[66] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol. 13, no. Feb, pp. 281–305, 2012.

[67] J. Jin, Z. Yan, K. Fu, N. Jiang, and C. Zhang, "Neural network architecture optimization through submodularity and supermodularity," arXiv preprint arXiv:1609.00074, 2016.

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

[69] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” arXiv preprint arXiv:1508.06576, 2015.

[70] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” arXiv preprint arXiv:1511.06434, 2015.

[71] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” in European Conference on Computer Vision, 2016.

[72] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” arXiv preprint, 2017.

[73] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” arXiv preprint arXiv:1703.10593, 2017.

[74] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ser. ICCV ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 4489–4497. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2015.510

[75] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli, “Perforatedcnns: Acceleration through elimination of redundant convolutions,” in NIPS, 2016.

[76] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, “Efficient and accurate approximations of nonlinear convolutional networks,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1984–1992, 2015.

[77] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in Proceedings

of the 27th International Conference on Neural Information Processing Systems - Volume 1, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 1269–1277. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2968968

[78] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," CoRR, vol. abs/1511.06530, 2015.

[79] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 2204–2212. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969073

[80] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[81] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, 2015.

[82] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," nature, 2016.

[83] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[84] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[85] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 39–57.

[86] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," arXiv preprint arXiv:1312.6199, 2013.

[87] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," arXiv preprint arXiv:1611.02770, 2016.

[88] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9185–9193.

[89] C. Xie, Z. Zhang, J. Wang, Y. Zhou, Z. Ren, and A. Yuille, "Improving transferability of adversarial examples with input diversity," arXiv preprint arXiv:1803.06978, 2018.

[90] "Google Cloud Vision," Link.

[91] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 1528–1540.

[92] W. Zhou, X. Hou, Y. Chen, M. Tang, X. Huang, X. Gan, and Y. Yang, "Transferable adversarial perturbations," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 452–467.

[93] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual clas-

sification," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1625–1634.

[94] ——, "Robust physical-world attacks on deep learning models," arXiv preprint arXiv:1707.08945, 2017.

[95] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, "Adversarial examples for semantic segmentation and object detection," in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1369–1378.

[96] J. Lu, H. Sibai, and E. Fabry, "Adversarial examples that fool detectors," arXiv preprint arXiv:1712.02494, 2017.

[97] J. Lu, H. Sibai, E. Fabry, and D. Forsyth, "Standard detectors aren't (currently) fooled by physical adversarial stop signs," arXiv preprint arXiv:1710.03337, 2017.

[98] Y. Zhao, H. Zhu, Q. Shen, R. Liang, K. Chen, and S. Zhang, "Practical adversarial attack against object detector," arXiv preprint arXiv:1812.10217, 2018.

[99] S.-T. Chen, C. Cornelius, J. Martin, and D. H. P. Chau, "Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector," in Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2018, pp. 52–68.

[100] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.

[101] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7263–7271.

[102] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T.-K. Kim, "Multiple object tracking: A literature review," arXiv preprint arXiv:1409.7618, 2014.

[103] W. Feng, Z. Hu, W. Wu, J. Yan, and W. Ouyang, “Multi-object tracking with multiple cues and switcher-aware classification,” arXiv preprint arXiv:1901.06129, 2019.

[104] S. Murray, “Real-time multiple object tracking-a study on the importance of speed,” arXiv preprint arXiv:1709.03572, 2017.

[105] J. H. Yoon, C. Lee, M. Yang, and K. Yoon, “Online multi-object tracking via structural constraint event aggregation,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[106] S. Sharma, J. A. Ansari, J. K. Murthy, and K. M. Krishna, “Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 3508–3515.

[107] C. Long, A. Haizhou, Z. Zijie, and S. Chong, “Real-time multiple people tracking with deeply learned candidate selection and person re-identification,” in ICME, 2018.

[108] Y. Xiang, A. Alahi, and S. Savarese, “Learning to track: Online multi-object tracking by decision making,” in 2015 IEEE International Conference on Computer Vision (ICCV), Dec 2015, pp. 4705–4713.

[109] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang, “Online multi-object tracking with dual matching attention networks,” in Computer Vision – ECCV 2018. Cham: Springer International Publishing, 2018, pp. 379–396.

[110] P. Bergmann, T. Meinhardt, and L. Leal-Taixé, “Tracking without bells and whistles,” CoRR, vol. abs/1903.05625, 2019. [Online]. Available: http://arxiv.org/abs/1903.05625

[111] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and real-time tracking," in 2016 IEEE International Conference on Image Processing (ICIP). IEEE, 2016, pp. 3464–3468.

[112] W. Choi, "Near-online multi-target tracking with aggregated local flow descriptor," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 3029–3037.

[113] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," Acm computing surveys (CSUR), vol. 38, no. 4, p. 13, 2006.

[114] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in Advances in neural information processing systems, 2015, pp. 1135–1143.

[115] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[116] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," arXiv preprint arXiv:1608.08710, 2016.

[117] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[118] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in International Conference on Learning Representations, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[119] N. Yoder, “Peakfinder: Quickly finds local maxima (peaks) or minima (valleys) in a noisy signal,” 2014. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/25500peakfinder

[120] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 4, pp. 509–522, April 2002.

[121] P. Dollár, R. Appel, S. Belongie, and P. Perona, “Fast feature pyramids for object detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 8, pp. 1532–1545, Aug 2014.

[122] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in Advances in Neural Information Processing Systems 30: Annual Conf. on NIPS, 4-9 Dec. 2017,, 2017, pp. 3859–3869. [Online]. Available: http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules

[123] X. Shi, Z. Gao, L. Lausen, H. Wang, D. Yeung, W. Wong, and W. Woo, “Deep learning for precipitation nowcasting: A benchmark and A new model,” in Advances in Neural Information Processing Systems 30: Annual Conf. on NIPS, 2017, pp. 5622–5632. [Online]. Available: http://papers.nips.cc/paper/7145-deep-learning-for-precipitation-nowcasting-a-benchmark-and-a-new-model

[124] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” CoRR, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[125] S. L. Soran B., Farhadi A., “Action recognition in the presence of one egocentric and multiple static cameras,” in ACCV 2014. Lecture Notes in Computer Science,, 2014, pp. 17–24.

[126] Y. Lu and S. Velipasalar, “Human activity classification incorporating egocentric video and inertial measurement unit data,” in Signal and Information Processing (GlobalSIP), IEEE Global Conf. on, Nov. 2018.

[127] M. M. Silva, W. L. S. Ramos, J. P. K. Ferreira, F. C. Chamone, M. F. M. Campos, and E. R. Nascimento, “A weighted sparse sampling and smoothing frame transition approach for semantic fast-forward first-person videos,” in 2018 IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 2383–2392.

[128] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1912–1920.

[129] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in Proceedings of the 26th annual international conference on machine learning. ACM, 2009, pp. 609–616.

[130] Y. Dong, T. Pang, H. Su, and J. Zhu, “Evading defenses to transferable adversarial examples by translation-invariant attacks,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019.

[131] C. A. Mack, NIST,SEMATECH e-Handbook of Statistical Methods, 2007.

[132] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” arXiv preprint arXiv:1412.6572, 2014.

[133] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” arXiv, 2018.

[134] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in 2017 IEEE International Conference on Computer Vision (ICCV), Oct 2017, pp. 2999–3007.

[135] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," CoRR, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[136] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[137] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask r-cnn," 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2980–2988, 2017.

[138] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," arXiv:1802.02611, 2018.

[139] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," CoRR, vol. abs/1706.05587, 2017. [Online]. Available: http://arxiv.org/abs/1706.05587

[140] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," CoRR, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038

[141] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” 2017.

[142] Y. Dong, F. Liao, T. Pang, X. Hu, and J. Zhu, “Discovering adversarial examples with momentum,” CoRR, vol. abs/1710.06081, 2017. [Online]. Available: http://arxiv.org/abs/1710.06081

[143] C. Xie, Z. Zhang, J. Wang, Y. Zhou, Z. Ren, and A. L. Yuille, “Improving transferability of adversarial examples with input diversity,” CoRR, vol. abs/1803.06978, 2018. [Online]. Available: http://arxiv.org/abs/1803.06978

[144] Y. Dong, T. Pang, H. Su, and J. Zhu, “Evading defenses to transferable adversarial examples by translation-invariant attacks,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019.

[145] Anonymized, “Anonymized github repository for our evaluation data,” https://github.com/anonymous0120/dr_images.

[146] “NSFW Data Scraper,” Link.

[147] “ICDAR2017 Robust reading challenge on COCO-Text,” Link.

[148] “ImageNet Challenge 2017,” Link.

[149] “Keras Applications,” Link.

[150] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao, “Foveation-based mechanisms alleviate adversarial examples,” arXiv preprint arXiv:1511.06292, 2015.

[151] qqwweee, “keras yolo3,” https://github.com/qqwweee/keras-yolo3.

[152] fizyr, “keras retinanet,” https://github.com/fizyr/keras-retinanet.

[153] pierluigiferrari, “ssd keras,” https://github.com/pierluigiferrari/ssd_keras.

[154] Pytorch, “Torchvision models,” https://pytorch.org/docs/master/torchvision/models.html.

[155] F. Luetteke, X. Zhang, and J. Franke, “Implementation of the hungarian method for object tracking on a camera monitored transportation system,” in ROBOTIK 2012; 7th German Conference on Robotics, May 2012, pp. 1–6.

[156] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in 18th International Conference on Pattern Recognition (ICPR’06), vol. 3, Aug 2006, pp. 850–855.

[157] Anonymized, “Anonymized github repository for the source code of our attack and evaluation data,” https://github.com/anonymousjack/hijacking.

[158] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, “Bdd100k: A diverse driving video database with scalable annotation tooling,” arXiv preprint arXiv:1805.04687, 2018.

[159] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” arXiv preprint arXiv:1804.02767, 2018.

[160] OpenCV, “Kalman Filter Class Reference,” https://docs.opencv.org/3.4.1/dd/d6a/classcv_1_1KalmanFilter.html.

[161] Z. Zhong, W. Xu, Y. Jia, and T. Wei, “Perception Deception: Physical Adversarial Attack Challenges and Tactics for DNN-Based Object Detection,” in Black Hat Europe, 2018.

[162] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, “Synthesizing robust adversarial examples,” arXiv preprint arXiv:1707.07397, 2017.

[163] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

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

[165] S. Zagoruyko and N. Komodakis, "Wide residual networks," arXiv preprint arXiv:1605.07146, 2016.

[166] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "The lottery ticket hypothesis at scale," arXiv preprint arXiv:1903.01611, 2019.

[167] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," arXiv preprint arXiv:1706.06083, 2017.

[168] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," arXiv preprint arXiv:1611.01236, 2016.

[169] S. Ye, K. Xu, S. Liu, H. Cheng, J.-H. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin, "Second rethinking of network pruning in the adversarial setting," arXiv preprint arXiv:1903.12561, 2019.

Yantao Lu received his B.S. degree in Electrical Engineering in 2013 from Xian Jiaotong University, Xi'an, China and M.S. degree in Electrical Engineering in 2013 from Syracuse University, Syracuse, US. He has completed the Ph.D. degree in the Department of Electrical Engineering and Computer Science, Syracuse University, in 2020. During his Ph.D. study, he has authored two journal publications and seven conference publications. His research interests are in the automatic driving perception, egocentric video processing, and adversarial examples for neural networks.