*ABSTRACT*

# The Intersection of Machine Learning and Cybersecurity

Kevin Kulda

Director: Matthew Pirko, SSCP

This paper will examine the intersection of cybersecurity and machine learning. Use cases integrating machine learning for both defensive and offensive cybersecurity will be surveyed. Within defensive cybersecurity, this paper will investigate how machine learning is being used to protect against external threats and internal threats. To show an interesting way machine learning may be used in a cyber attack, this paper will look at a Prime+Probe cache side-channel attack that aims to learn which machine learning transfer model a program is running. From an external perspective, the analysis will show how the side-channel attack may be implemented, and how it can be defended against. Finally, we propose an additional method to detect and prevent this attack on an internal network.

APPROVED BY DIRECTOR OF HONORS THESIS:

____________________________________________

Professor Matthew Pirko, Department of Information Systems

APPROVED BY THE HONORS PROGRAM:

____________________________________________

Dr. Elizabeth Corey, Director

DATE:________________________

# The Intersection of Machine Learning and Cybersecurity

A Thesis Submitted to the Faculty of

Baylor University

In Partial Fulfillment of the Requirements for the

Honors Program

By

Kevin Kulda

Waco, Texas

May 2020

TABLE OF CONTENTS

# CHAPTER ONE

## Introduction and Motivation

Cyberspace is now the fifth domain of war.[1] Everyday new cybersecurity breaches, exploits, and attacks are observed, mitigated, and studied. Cyber security professionals constantly improve their capabilities to thwart these evolving attacks. Cyber attacks come from unsophisticated sources: scriptkiddies and lone wolf hackers, but they are increasingly coming from very sophisticated sources: nation-states, criminal hacking rings, and tech-savy insiders. These malicious actors are motivated to commit cyber crimes because this attack vector avoids direct conflict and has a low risk-reward ratio compared to more base ways of committing crimes. Additionally, as our world becomes more technologically advanced, more data is put on computers that may be susceptible to cyber attacks, especially from sophisticated sources. Malicious actors know how to monetize or create value from stolen data or compromised computational resources. A hacker may use ransomware to get rich or sell personal information on the dark web to further identity fraud, and a nation-state may profit politically from national secrets or may inflict harm by manipulating power grids or elections, or may improve military technology by stealing intellectual property.

As more and more value is realized from cyber attacks the intent to make these attacks more effective increases. Malicious actors creatively use new technology to develop attacks and develop new attacks to exploit new technology. Machine learning is one such technology that is positioned at the intersection of helping defend against new attacks and helping deploy new attacks. For example, machine learning

1. Charles H Hall, *Operational Art in the Fifth Domain*, technical report (NAVAL WAR COLL NEWPORT RI JOINT MILITARY OPERATIONS DEPT, 2011).

may be used defensively to make better tools that scan one's network looking for vulnerabilities.[2] However, this same use for machine learning may aid malicious actors in finding vulnerabilities in one's network to exploit. Additionally, machine learning technology itself may be attacked to steal intellectual technology or to reverse engineer the technology to learn how it can be tricked.[3] Being able to trick a classifier that depends on machine learning gives an attacker the ability to bypass the virtual security check the program is designed to implement. For instance, if an attacker can reverse engineer the machine learning model used to classify an executable as a virus or not, the attacker is much closer to tricking the model into classifying a program that is a virus as a benign program when deploying infectious code on a victim system.

The attack this paper will ultimately look at in depth is the Prime+Probe cache side-channel timing attack.[4] It is proposed that this attack can be used to predict which transfer machine learning model a program is using and that machine learning may be used to help execute this attack. The Prime+Probe attack monitors the last level cache on a processor shared by more than one user. The attack gleans information by detecting patterns in which cache sets/lines are used by a victim process. If a machine learning neural network is used to process an item for classification the Prime+Probe attack may be used to monitor this process to learn which neural

2. Fabian Yamaguchi, Felix Lindner, and Konrad Rieck, "Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning," in *Proceedings of the 5th USENIX conference on Offensive technologies* (USENIX Association, 2011), 13–13.

3. Sanghyun Hong et al., "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," *arXiv preprint arXiv:1810.03487*, 2018,

4. Fangfei Liu et al., "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy* (IEEE, 2015), 605–622.

network model is being used. The two methods proposed to predict ML models is: 1) take a snapshot of the launch or other common point during the neural network classification process in testing and use this snapshot to predict a model based on a snapshot from a black box, and 2) use the Prime+Probe attack to find the time it takes a machine learning model to classify an input and use this timing information to predict models based on run-time information gleaned from white-box test programs.

Machine learning technology may be used to help implement these attacks by training a model to predict which snapshots or timing information correspond to which transfer model. In the wild it appears it is highly unlikely snapshots or timing information will match testing data exactly because several programs will be using the cache at one time. It is important to account for error and use a system for predicting models based on a data range rather than perfect match.

As machine learning programs become widely deployed it may be useful for malicious actors to learn which transfer model a program uses as a first step to compromise the model. Knowing the teacher model a program uses limits the attackers search space of neural networks so they can focus their energies on one to a few network architectures. With a reduced list of models, an attacker can use other methods to reverse engineer the model or learn how to trick the model using a Generative Adversarial Network (GAN).[5] Knowing which model a program uses to then attack it is similar to how attackers learn what operating system a server runs to learn form there what exploits it may be vulnerable to and how to compromise it. Without reducing the number of possibilities in the search space, the potential time to generate a successful attack increases significantly and there is a reduced chance of success. The ability to predict neural network models using the Prime+Probe attack could

5. Ian Goodfellow et al., "Generative adversarial nets," in *Advances in neural information processing systems* (2014), 2672–2680.

be a tool that malicious actors widely share to be used in attack suites such as Kali Linux.[6] Therefore, if an attacker wanted to target a machine learning program or wanted to compromise one found in reconnaissance, this tool could be used without having to spend the time to craft it themselves. The widespread use of a tool like this would make exploiting machine learning models easier, thus compromising their general integrity.

6. Kali, "Kali Linux," 2020, accessed March 29, 2020, https://www.kali.org/.

# CHAPTER TWO

## Background

While machine learning was once just an idea, today it is a sophisticated science. Machine learning is used as a tool to perform tasks faster and potentially more accurately than humans. Most tools used in the cybersecurity space are classification engines. They act as a black box that receives inputs, analyzes the data, and then reports a predicted classification of the input. For example, a machine learning model may be used to classify program files as malicious or not malicious.[7] Or this tool could be used to classify images as adversarial or benign.[8] For malicious actors, a machine learning based tool could be used to generate adversarial images by altering the image data until the image is misclassified.

Machine learning is an exceptional classification tool for several reasons. Machine learning can accurately classify data it has never seen before, but is similar to what it was trained on. Machine learning can also continuously improve to better classify inputs. Finally, machine learning is widely used because it can process data at a blinding fast rate. As this technology developed, cybersecurity professionals and researchers began adapting it to the security domain for defense, and hackers began adapting it for cyber offense. It is evident that this trend is entrenched in cyber space and will only grow in the future.

7. Dragoş Gavriluţ et al., "Malware detection using machine learning," in *2009 International Multiconference on Computer Science and Information Technology* (IEEE, 2009), 735–741.

8. Warren He et al., "Adversarial example defense: Ensembles of weak defenses are not strong," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)* (2017).

While the study of machine learning is integral to fully understanding the intersection of machine learning and cybersecurity, for purposes of this paper we do not analyze machine learning in depth. If the reader would like more information on machine learning we encourage them to see Appendix A where we discuss many aspects of machine learning programs and neural networks. Additionally, in this appendix the reader will find a survey of five transfer learning models. This survey may be useful to gain an understanding of what the Prime+Probe attack aims to attack.

# CHAPTER THREE

# The Intersection of ML and Cybersecurity

This section will be an analysis of where machine learning (ML) and cybersecurity are intersecting today, and where they are projected to intersect in the future. We will look at where machine learning is being used in offensive cyber as well as defensive cybersecurity, and withing defensive cybersecurity we will look at how machine learning is being used to defend against external and internal threats.

## *Offensive Cyber Attacks*

With the advent of machine learning the question is not whether ML is being used by cyber criminals, but how it is being used by cyber criminals. While there is documentation proving how some cyber criminals have or can use ML to their advantage, some use cases can be inferred by aligning the advantages of ML with the problems cyber criminals try to solve.

### *Machine Learning Used in Cyber Attacks*

Machine learning may be used in cyber attacks where detailed classification or the automation of analyzing data is needed. This tool may allow an attacker to expand their capabilities and range. For purposes of this paper we will look at how machine learning is being used in the domains of social engineering, broken authentication, attack performance, and tool automation.

Consider that in a social engineering attack the cyber criminal tries to use false enticement or an impersonation to get victims to do something that will be harmful. The success of an attacker may depend on how well they convince victims to believe a lie. ML may help attackers sell a lie in two ways. First, ML may be used in the reconnaissance phase of the attack. A cyber criminal may use ML to scan social media and the web to learn about the behaviour of a potential victim. With more information gathered an attacker will be more informed on how to manipulate the victim. For example, if a scan on a victim shows that the individual shops excessively at store X, the attacker will know to package an attack in a fake advertisement from X. Second, ML may be used by an attacker when preparing a social engineering attack. Consider an attacker wanting to write a message to user A that looks like it was from user B. If the attacker has a sample of user B's writing, they can train a ML program to output sentences in the style of B. An example of this is how researchers trained a ML program to write in the style of Shakespeare.[9] Additionally, cyber criminals can now use ML to generate videos of real people saying things they never had said. These deceptive videos are commonly called deep fakes.[10] The fraudulent qualities of these videos are almost imperceptible to the human eye and can be used in social engineering to try and manipulate victims into believing something that is false.

Malicious actors can also use ML to compromise accounts and passwords. In recent years there have been large breaches that have exposed billions of users passwords.[11] These passwords reveal patterns that can be helpful in predicting new user

9. Rosaria Silipo, "Can AI write like Shakespeare?," 2020, accessed March 28, 2020, https://towardsdatascience.com/can-ai-write-like-shakespeare-de710befbfee.

10. David Güera and Edward J Delp, "Deepfake video detection using recurrent neural networks," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (IEEE, 2018), 1–6.

11. Brian Barrett, "Hack Brief: An Astonishing 773 Million Records Exposed in Monster Breach," 2019, accessed January 16, 2019, https://www.wired.com/story/

passwords. ML is used to extract these patterns and create more accurate password tables to use in future attacks.[12] With more accurate password tables, attackers increase their chance of gaining access to accounts especially when they already can guess valid usernames such as email addresses or simple name concatenation.

ML can also be used in the CAPTCHA step of the authentication process.[13] After entering a valid username and password some applications will require you to pass a CAPTCHA to verify that you are a user and not a bot. The CAPTCHA is designed to filter out bots because the user must select images containing objects not easily recognizable by an image classification engine or letters from a highly altered character set. Researchers have developed ways, however, of using ML programs to successfully get through the CAPTCHA and accurately identify vague characters in an image.[14] This allows the cyber criminal to better automate cyber attacks and expand the number of accounts they can try to breach at any given time.

## *Defensive Cybersecurity*

Defensive cybersecurity may be divided into 8 domains: Security Management, Identity and Access Management, Security Engineering, Business Continuity, Compliance, Cryptography, Physical Security, Software Development Security, and

collection-one-breach-email-accounts-passwords/.

12. Christoffer Olsen, "A Machine Learning Approach to Predicting Passwords," 2018,

13. Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis, "I am robot:(deep) learning to break semantic image captchas," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (IEEE, 2016), 388–403.

14. Fabian Stark et al., "CAPTCHA Recognition with Active Deep Learning" (September 2015).

Security Operations.[15] While Machine Learning may be deployed in certain domains more than others, there is likely at least one use case for ML technology in each of these domains. However, for purposes of this paper we will consider the following domains: Identity and Access Management, Security Engineering, and Security Operations.

*Identity and Access Management*

This domain covers the necessary tools and techniques to identify and authenticate users requesting access to a computer system. This could be logging into a terminal at one's place of work, logging in remotely using a VPN, or swiping a key card to be let into a server room. In each situation proper access management is required to prevent the opportunity for unauthorized use to occur and to correctly attribute actions on the system to users. Machine learning technology is commonly deployed today to assist with access management and new techniques are being developed.

Machine learning is primarily used in access management through biometric authentication. Biometric authentication has been developed to identify an individual based on their face, hand, fingerprint, void, ear, vein orientation, and other attributes.[16] In many cases the user must have their physical characteristic input to a ML model to have it trained on.[17] Once the model is able to classify the person's

15. Kenneth Magee, "The CISSP Domains - An Overview," 2020, accessed March 28, 2020, https://resources.infosecinstitute.com/the-cissp-domains-an-overview/#gref.

16. Debnath Bhattacharyya et al., "Biometric authentication: A review," *International Journal of u-and e-Service, Science and Technology* 2, no. 3 (2009): 13–28.

17. K. Sadeghi et al., "Performance and Security Strength Trade-Off in Machine

characteristic it will uniquely identify that person in future situations when they go through the identification process for device or location access.

An area of research related to identity and access management is continuous behavioral analysis. This analysis would be used to monitor when a user is on a system or at a workstation to grant access as long as the correct user remains in control of the system.[18] [19] If for example, another person comes to the user's workstation to access their terminal, the system would identify the anomaly and lock the user's account. Semantics such as keyboard typing patterns, voice recognition, mouse movements may be used in addition to facial recognition.[20] This technology would insure that even if a malicious actor gained access to a user's device or workstation, they would not continually gain access to the resource.

*Security Engineering*

Security engineering refers to building the technology used in cyber defense. This technology may be firewalls, routers, intrusion detection and prevention systems, antivirus tools, email filtering, and vulnerability scanning tools. Because machine learning can be used to process vast amounts of data and identify anomalies or inputs with specific characteristics, there are many uses for ML in this domain. For

Learning Based Biometric Authentication Systems," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2017), 1045–1048.

18. Kyle O Bailey, James S Okolica, and Gilbert L Peterson, "User identification and authentication using multi-modal behavioral biometrics," *Computers & Security* 43 (2014): 77–89.

19. Respondus Monitor, "Respondus Monitor," 2020, accessed April 17, 2020, https://web.respondus.com/he/monitor/.

20. Bhattacharyya et al., "Biometric authentication: A review."

example, ML may be used in intrusion detection and prevention systems to identify traffic that is suspicious or traffic that has been flagged in the past.[21] The type of request or IP address of the traffic may be the specific data points the ML model looks for. Additionally, vulnerability scanning tools may use ML to learn how to better search for weaknesses in a network.[22] A ML classifier may also be a better filter for emails because it can be constantly trained on emerging phishing email attacks to shut down a malicious email broadcast within an organization before users have the chance to view the malicious content.

### *Security Operations*

Security operations entails using existing technology, techniques, and methods to analyze a system for security vulnerabilities, incidents, and threats. In this domain many of the tools that may use ML are deployed to assist security analysts in performing their tasks. Tasks may be finding vulnerabilities in a newly deployed network, discovering how an attack broke through network defenses, and finding which user's behavior is suspicious to attribute data exfiltration to an account.

A specific point of interest in this domain is how to make better use of a security analyst's time. Research is currently being done to determine how to achieve this through the use of machine learning.[23] For example, intrusion detection and

21. Anna L Buczak and Erhan Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials* 18, no. 2 (2015): 1153–1176.

22. Patrice Godefroid, Hila Peleg, and Rishabh Singh, "Learn&fuzz: Machine learning for input fuzzing," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (IEEE, 2017), 50–59.

23. James B Fraley and James Cannady, "The promise of machine learning in cybersecurity," in *SoutheastCon 2017* (IEEE, 2017), 1–6.

prevention systems can generate more data than a team of analysts have time to search through. A ML classification system can be used to analyze the data to find the reports most likely to have critical information that deserves the analysts attention.[24] In this case the analysts are directed to the incidents that are most sensitive and critical to the network.

### *Defense Against External Threats*

While the majority of research into using machine learning for cybersecurity focus on defense against external threats, some applications and research naturally overlap with defense against internal threats. Applications unique to external threats include, network protection, intrusion detection and prevention, phishing email scans, and virus scans. While an insider may execute one of these attacks, they are uniquely viewed as coming from external sources.

Applications of machine learning in cyber security that overlap with defense against external and internal threats include authentication and access management,[25] incident analysis,[26] and network anomaly detection.[27] The ML classifiers used in access management may be intended to prevent an external actor from being let into proprietary domains, but they also prevent internal actors from gaining access to

24. Buczak and Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection."

25. Bhattacharyya et al., "Biometric authentication: A review."

26. Buczak and Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection."

27. Taeshik Shon and Jongsub Moon, "A hybrid machine learning approach to network anomaly detection," *Information Sciences* 177, no. 18 (2007): 3799–3821.

resources they are not cleared to access. In incident analysis the source of an incident a ML tool identifies may lead to an internal actor as well as an external actor. A network anomaly that ML scans find interesting may be a firewall rule change or excessive traffic to a network resource. The tool will alert security analysts to this anomaly regardless if the actor is internal or external even though it is considered unlikely an internal actor would exhibit the kind of risky behavior detected my the ML tool.

## *Defense Against Internal Threats*

Internal threats in cybersecurity are generally considered to be the threats that come from people within an organization. Insiders may include a consulting web developer, a disgruntled administrator, or a CEO. Insiders posing a risk to a system are commonly categorized as ignorant, abusive, or malicious. A threat from ignorance is observed when a user clicks a malicious link, or downloads a malicious file when they were not aware there was any risk in their action. A threat of abusiveness is observed when a user clicks a malicious link, or downloads a malicious file despite knowing that there are guidelines and policies to follow to make sure their action is safe. A threat from a malicious insider is set apart from the other threats because here the user intends harm. This is when an insider, for example, creates a malicious link, or distributes a malicious file to steal, damage, or disrupt the organization.

It is difficult to discern what machine learning technology has been developed to protect uniquely against internal threats. Typically, threats introduced by ignorant or careless insiders are mitigated by virus scanners, network scanners, and incident response tools developed to defend against external threats. Because ignorant or careless users let in threats, rather than generate them, the system will typically need to defend against the threats let in rather than the users who unknowingly let them in.

An area where machine learning may be used to help defend against the ignorant or careless insider threats is in static code analysis tools.[28] Today, when an application is built and deployed on the web it is usually highly complex. It is difficult to analyze all of the code in the application to make sure it was programmed with security in mind. To help developers secure applications, static code analysis tools have been developed to identify security flaws before code is put into production. It is reasonable to assume some developers are ignorant of secure programming, and some developers are careless of security measures when developing an application. Therefore, it would be beneficial to make a static code analysis tool to continually perform better and identify as many security flaws as possible. While it is not clear whether machine learning has been deployed to this end yet, it may be a suitable tool to improve code analysis tools currently used.[29]

The unique internal threats an organization may encounter will typically be introduced by the malicious insider and it is not clear how organizations are currently or planning to mitigate this risk. Reasons this is not clear may be that organizations are keeping their technology private or that insider threats are so unique to an organization that they do not consider widely disseminating their work. Additionally, some organizations may deem insider threats to be a low enough risk that they are not taking measures to defend against them using machine learning.

Protecting against data exfiltration is one way machine learning could be used

28. Ulas Yüksel and Hasan Sözer, "Automated classification of static code analysis alerts: a case study," in *2013 IEEE International Conference on Software Maintenance* (IEEE, 2013), 532–535.

29. Ibid.

to defend against a malicious insider threat. Data exfiltration by an insider is when an employee exports an organization's intellectual property to a location it is unauthorized to be. Exfiltration could be removing documents from the office, allowing an outsider to view proprietary materials, or uploading data to an offsite server. Machine learning could be used to detect the presence of data exfiltration by detecting patterns or anomalies that suggest this is happening. For example, a ML program could alert a cyber security analyst if a user has a pattern of emailing large attachments to an email address outside the company. Additionally, a ML program could use a workstation webcam to detect if an external person is present when sensitive information is being displayed on the user's device. While the ML technology used in these tools is similar to existing ML technology used to for access management,[30] systems engineering, and security operations,[31] it would be directed at a use case specific to defending against internal threats.

30. Bailey, Okolica, and Peterson, "User identification and authentication using multi-modal behavioral biometrics."

31. Buczak and Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection."

# CHAPTER FOUR

# Prime+Probe Cache Attack

The Prime+Probe[32] cache side-channel attack exploits natural behavior of microchip hardware to steal privileged information. The vulnerability this attack exploits is the situation when the last-level cache stores user data that other users may observe. Users may not explicitly observe data in the last-level cache if the data was loaded by another user, but by using the Prime+Probe attack they may indirectly glean information about the data. This attack is possible because the last-level cache is shared between all processes using a cpu, and because there are assembly instructions to manipulate the cache.

## *Why the Prime+Probe Attack*

The Prime+Probe attack emerged from the interest in hardware side-channel attacks that resulted from the Meltdown[33] and Spectre side-channel attack.[34] The attack is interesting because it exploits a vulnerability inherent to most processors, namely the shared cache. This cache was built to improve the performance of CPU's, and the naive defense to the Prime+Probe attack of eliminating the last-level cache would reduce performance. Because of the attrition between performance and secu-

32. Liu et al., "Last-level cache side-channel attacks are practical."

33. Moritz Lipp et al., "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)* (2018).

34. Paul Kocher et al., "Spectre Attacks: Exploiting Speculative Execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)* (2019).

rity, it is likely this kind of vulnerability will remain in place for at least the near future and will be of interest to malicious actors and researchers alike.

### *Setting*

The Prime+Probe attack must be performed in a shared environment where multiple users may use the same computer resources at one time. Obvious settings where this occurs is in cloud computing or shared corporate environments. While the shared system gives the appearance that each user has dedicated computer resources, many users may be allocated to one cpu or server and will share the resource with many other users. This environment is good for efficiency and resource allocation, but through hardware attacks, this environment makes the data loaded onto the cpu visible to the other user processes through side-channel attacks.

### *Implementation*

The attack functions through a malicious process priming the cache with known dummy data and then probing the cache at intervals for the dummy data to determine which data has been evicted by the other users' processes. If the malicious actor has run this attack on known processes, the attacker may compare the live attack prime and probe data to the trial data to know more specifically what the other user is doing. This process has no interaction with the user process other than impacting the user data loaded in the cache.

*Proposed Prime+Probe Attack Implementation*

We propose that the Prime+Probe attack may be used to learn what machine learning transfer model another user or process may be running on a shared resource by detecting the run time of a single classification or by viewing a snapshot of the cache during a classification. For example, consider a deep learning model being used to classify malware binaries as benign or malicious. Note that we are interested in detecting a transfer model and not custom models. To learn precisely what model a program is running would require learning much about the architecture. We would need to learn the number of layers, number of nodes per layer, activation functions, and more to reverse engineer the model. If you consider sophisticated models such as transfer models, more information would be needed such as how many residual layers a model has, whether it has max pooling layers, and where the frozen layers begin. This information is very granular would be very difficult to glean from a side-channel attack. The Prime+Probe attack is useful for gleaning big picture information from the cache. For this reason we chose to be interested in detecting one of a few transfer models from high level details. For this implementation to work, the attacker must already know or must assume the victim program is using a transfer model.

In a corporate environment a machine learning classification tool such as a virus detection engine may be running on the same central computing resources that other users are allowed to access under the assumption the users' processes will not interfere with the machine learning processes. Since transfer models are becoming increasingly common, a malicious insider may assume the virus binary classifier is a transfer model trained on binaries. By choosing one of the common malicious binary data sets,[35] the insider may build a suite of transfer models to classify malware binaries

35. VIRUSTOTAL, "VIRUSTOTAL," 2020, accessed March 28, 2020, https://

hoping that at least one model is likely used by the organization. After running the attack on this suite of models the insider would be prepared to run the live attack. The insider would have data to show what the run-time of each model is and what the cache looks like at specific points during the classification process. This data can be used by the attacker to compare against the results of the live attack to determine which model the antivirus tool is using.

Unless there are detection measures in place, an insider may run the Prime+Probe attack unfettered to try and learn what deep learning transfer model the organization has deployed. If the insider gleans data that closely matches one of the suite models, they are much closer to compromising the deployed model. With the correct model in place and a relatively similar binary data set, the malicious insider can craft malware and test it on the model to determine what will be misclassified as benign and be allowed to be executed on the system.

The advent of transfer learning models in particular makes this attack possible. The Prime+Probe attack is used to learn general information and cannot extract the data in the cache to look at it directly. If a malicious actor tried to learn all the characteristics of a ML model by attacking the cache to reverse engineer the model they would likely fail. Other research demonstrates the limits of the Prime+Probe to this end and propose that the Flush+Reload cache side-channel attack[36] is more relevant to learning granular information.[37]

---

www.virustotal.com/gui/intelligence-overview.

36. Yuval Yarom and Katrina Falkner, "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)* (2014), 719–732.

37. Hong et al., "Security analysis of deep neural networks operating in the presence of cache side-channel attacks."

# CHAPTER FIVE

# Proposed Defenses to the Attack

To defend against this implementation of the Prime+Probe attack one must either be able to detect and shut down the attack, or one must create an environment where the attack does not have one or more necessary conditions to execute. Currently, the only proposed methods to protect against the attack are to eliminate a condition necessary to execute the attack. Otherwise, it is recommended to alter the program being spied on to protect it from the attack. The idea behind this method is that even if the ML model is identified, the attacker will not be able to trick it because it has been modified or designed with this possible situation in mind. In addition to discussing these methods of defense we will propose one new method to detect and shut down the attack.

## *Harden the Victim Program*

Researchers have already proposed ways and tested methods to defend transfer models against attacks similar to the Prime+Probe attack. The existing methods focus on hardening transfer learning models from being able to be tricked.[38] Hardening the models have the potential to do two things: 1) they alter the architecture of the model so it is not easily guessed, and 2) they alter the layers in the model so it will not be tricked.[39] This method attempts to make sure that even if a model is predicted

38. Liu et al., "Last-level cache side-channel attacks are practical."

39. Alexey Kurakin, Ian Goodfellow, and Samy Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016,

and inputs are crafted to try and trick it, the victim model will be sophisticated enough that it will not misclassify the malicious inputs. Specific implementations of this method include mandating certain layers be retrained by the user, allowing users to remove layers to change the structure of the model, and training the parent on altered inputs to make it as difficult as possible for someone to craft a successful malicious input.[40]

The method of altering the architecture of the model is particularly useful to defend against this implementation of the Prime+Probe attack. Altering the architecture has the potential to change the run-time of a classification and therefore the cache set use because it will make the model unique to the user. If the run-time of a classification is unique and the cache set use is unique the attack will have reduced ability to predict which transfer model is running based on samples taken from testing on the generic transfer model. The run-time and cache set use may be similar enough to a model that the attack prediction will be accurate, but this is uncertain without testing.

### *Change the Hardware*

Because the Prime+Probe attack depends on multiple users sharing a system, and multiple processes sharing a CPU, one may consider how to defend against the attack by limiting how systems allocate users to machines and how processes are allocated to CPUs. Methods to defend against the attack within this category include: isolating virtual machines in multi-user systems, allowing one process on a CPU at a time, partitioning the last level cache for each process, making virtual memory

40. Kurakin, Goodfellow, and Bengio, “Adversarial machine learning at scale.”

randomly assigned to cache sets, or eliminating the last level cache on the chip.[41] While each of these methods would be effective at blocking the attack, they all reduce the efficiency of the system. Resources are allocated to multiple users and processes to maximize resource occupation and use. In the general scenario, systems maximize the number of users that may be sharing a system, they maximize the number of processes that may run, and they maximize the how fast processes are run. The system can no longer reach the same peak performance if these defense measures are in place. For this reason these defense measures are not desirable, especially if the risk of a model being compromised is low.

### *Attack Instruction Detection*

At this time, it appears no detection measures have been proposed for the Prime+Probe attack. An insider may execute the attack as they wish on a system and no malicious activity will be attributed to the insider. To detect the Prime+Probe execution, we propose that an operation signature be created for the attack so network scanners may identify it. We define an operation signature as an operation or set of operations that is unique to the attack. For example, the Prime+Probe attack must populate the last-level cache with data it can probe for, and then it must probe the entire cache for the data it placed there to learn what was evicted. This process must be executed constantly to glean useful information from the cache. If the Prime+Probe instruction execution pattern is unique enough, this operation signature may be used to identify when a process is running the attack on a network.

The drawback to the instruction detection method is that monitoring pro-

41. Liu et al., "Last-level cache side-channel attacks are practical."

cesses for malicious execution patterns will add overhead to the system which will reduce efficiency. Additionally, a sophisticated malicious actor may find a way to Prime+Probe the cache using novel techniques that will not be detected using a tool that looks for operation signatures unique to the existing Prime+Probe attack. Ideally, an operation signature would be identified that the Prime+Probe attack is dependent upon and that is also unique enough to minimize the number of false positives this detection method may generate.

# CHAPTER SIX

## Conclusion

Recent advances in machine learning have made it an attractive tool to use in many areas of our digital world. These powerful classification engines can help us drive our cars, secure access to our devices, and help detect when people are lying. For cyber criminals, machine learning also presents both a new target for attack, and a weapon to add to their arsenal. Evidence suggests that cyber security and machine learning are intersecting today, and we project they will continue to intersect in the future.

Because malicious insiders have the potential to execute powerful attacks, it is important to direct focus to this area of cyber defense. To illuminate where machine learning and cybersecurity are intersecting, and where internal threats are the primary concern, we propose a new implementation of the Prime+Probe attack[42] that aims to learn which machine learning transfer model a program is using. Our implementation of the Prime+Probe attack tries to guess which transfer model a program is using to learn how to trick the model for malicious purposes. For example, an insider may want to know which transfer model their organization is using to scan program binaries for viruses so the insider can learn how to craft a malicious binary that will be classified as benign by the scanner. The Prime+Probe attack is particularly relevant to insiders because the attack depends on two users or processes being co-hosted on the same system hardware. Furthermore, the malicious insider may use machine learning to improve the attack effectiveness by training a model that classifies data

42. Liu et al., "Last-level cache side-channel attacks are practical."

from the attack execution as indication one of several potential transfer models.

We believe machine learning will continue to be integrated into cybersecurity and it is important for organization leaders to be aware of what ML technology exists at present. Machine learning has the power to transform cyber defense and offense because this technology can perform tasks that previous computer systems have not been able to perform. Security professionals must be vigilant against this new threat and not overlook the lone malicious insider for the multitude of known external attackers.

Appendices

# APPENDIX A

## A Machine Learning

Machine learning is the use of algorithms, mathematical models, and computational tools to automate the improvement of a program. Use cases for machine learning are limitless. The purpose of a machine learning program may be to detect cats in an image, play a game of AlphaGo, or discover vulnerabilities in a network. Developing programs often requires expert oversight, but once they are deployed they may function autonomously. One example of this is when researchers set up a machine learning program and instructed it to learn how to understand a language by viewing captioned videos.[43] Initially the bot was illiterate. After training though, this bot became proficient at recognizing the English language and could predict what a spoken sentence meant.

If a machine learning program were a body, the neural network would be the brain. These networks are mathematical models trained on labeled data so they accurately classify new but similar data. These models are made up of algorithms, data structures, and data points that are manipulated to increase the classification accuracy. Much of a machine learning program is refining and improving the neural network to improve its decision making ability. Therefore, accuracy of a machine learning program is dependent on the neural network. This is why much research and effort is put into building improved neural networks designs and algorithms.

Transfer learning models are highly sophisticated neural networks that have

43. Candace Ross et al., "Grounding language acquisition by training semantic parsers using captioned videos," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), 2647–2656.

been developed in the last decade. Neural networks emerged as a simple idea in the 1950's, first became feasible in the late 1990's, and are now so sophisticated that it is non-trivial to construct a neural network that outperforms a transfer learning model. With some study one may easily grasp how to build a simple neural network to classify inputs. There are now several open source machine learning platforms and an equal number of tutorials. The simplest design of a model that one could build is the perceptron, a basic three layer neural network. However, it is typically in the interest of users to adapt a transfer model to their own needs rather than construct and train a neural network independently. By adapting a transfer model users take advantage of all the expertise and time it took to build a model that is highly accurate. For example, transfer models have reached accuracy above 95% when classifying handwritten digits from the MNIST data set.

In the remainder of this section we will discuss universal and specific elements of neural networks. The reader may notice that much of our discussion of neural networks will be related to models designed to classify images. We chose to focus on image classification models because it appears these models have the most documentation which makes it easier to understand their architecture than less documented models. Additionally, we believe image classification models represent the essential elements of a neural network well, and the sophisticated elements of these models are representative of what other models have or strive to have. Other models may be designed for source file classification, natural language processing, voice recognition, and more.

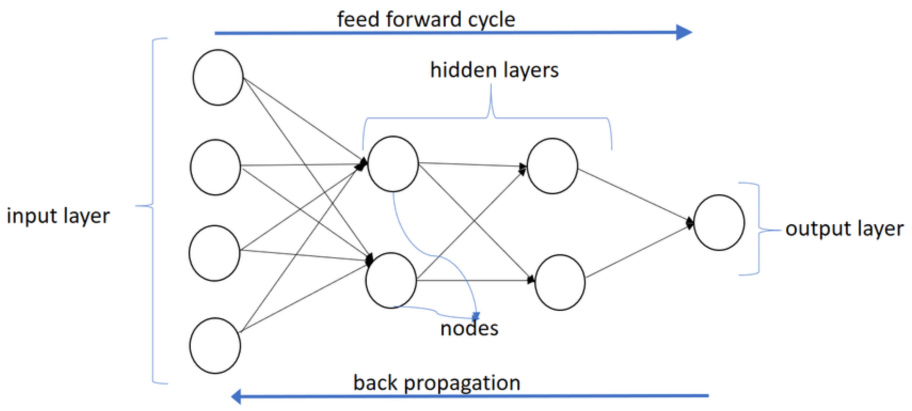


Figure 1.1: A Simple Neural NetworkReal caption[44]

Neural networks, as their name indicates, are modeled after neural pathways in the human brain. The idea of a brain's neural pathways behaving as a system of connected nodes that self-strengthened as the pathways were used was proposed in the late 1940's. In the 1950's this idea was transferred to computer science and the first machine neural network was designed. This first artificial neural network was called a perceptron. Similarly to how the human brain contains a web of neural pathways, these primitive neural network models had a network of their own. However, instead of cell based neurons, the networks were made up of an input layer of nodes, one inner (hidden) layer of nodes to operate on the input data, and an output layer that classified the input as something the network was told to predict.

44. "simple-neural-network," 2019, accessed December 9, 2019, https://www.researchgate.net/figure/A-simple-neural-network-with-two-hidden-layers-of-two-nodes-each-four-inputs-and-a_fig1_327637282

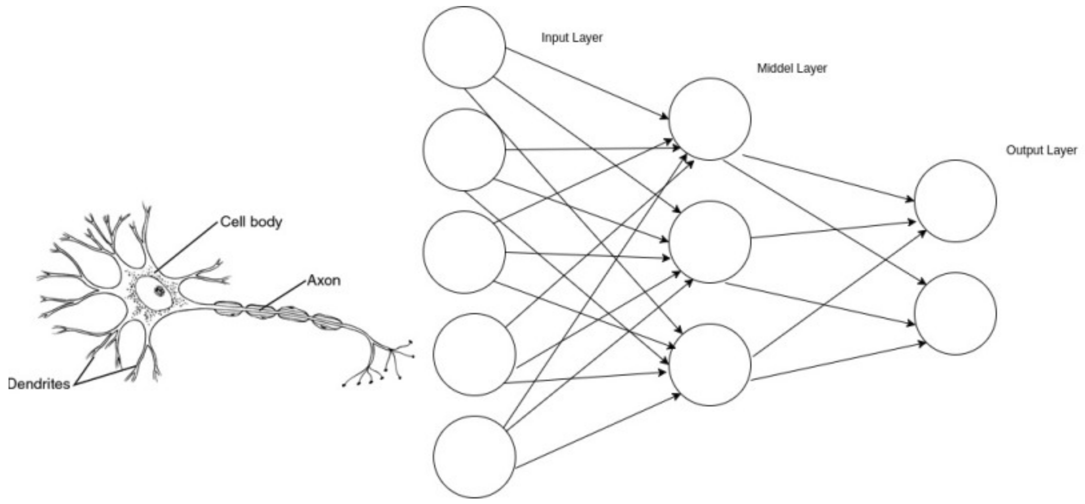


Figure 1.2: Artificial NN vs Biological NN[45]

Today, the architecture of neural networks is much more complex than the primitive perceptrons. Every architecture, however, is essentially a directed acyclic graph that shares a basic structure containing an input layer, multiple inner or hidden layers, and an output layer, each containing nodes, weights, biases, and activation functions.

45. Gogoi, "ann-vs-bnn," 2019, accessed December 9, 2019, https://blog.knoldus.com/first-interaction-artificial-neural-network/

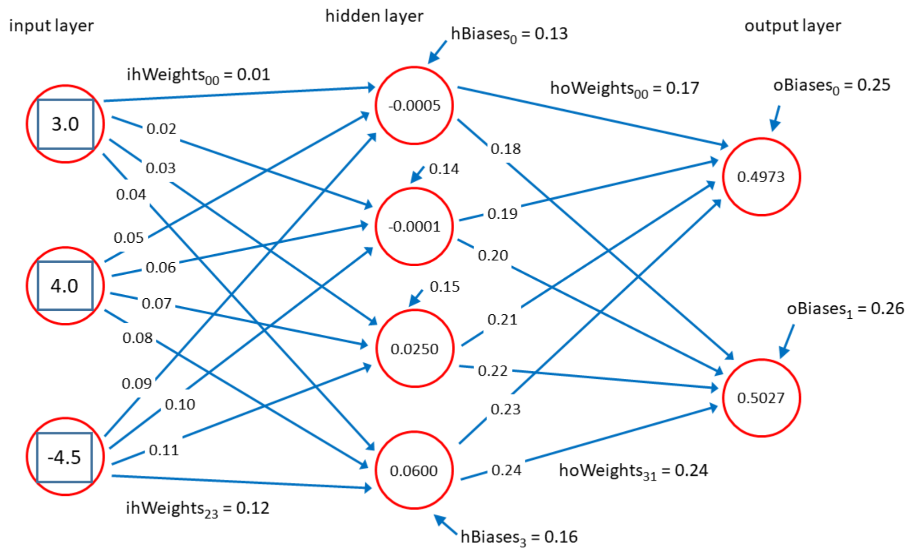


Figure 1.3: Inception-v1 Layers[46]

*Input Layer:*

An input layer is a collection of passive neurons that receive the data being input into the neural network. These neurons typically do not alter the data. They only prepare the data to be passed to the subsequent layers of the network.

*Inner/Hidden Layers:*

A neural network will have at least one inner/hidden layer. Inner layers are responsible for manipulating the data or directing it through the network to optimize classification. The inner layers are where the decisions of the network are formed. The parameters within the inner layers are what is trained to improve the classification accuracy of the model.

46. Christian Szegedy et al., "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 1–9

*Output Layer:*

The output layer is the final layer in a neural network. At this layer the input is finished being processed and is prepared for classification. Typically the output layer will contain a number of nodes equal to the number of classification options. For example, if a network is designed to classify an animal as a cat or a dog, the output layer will have two nodes.

*Nodes:*

All neural network layers are composed of an arrangement of nodes. Nodes operate on the data being processed by the neural network in several ways. The nodes are typically associated with an activation function, discussed below, to normalize the data and to keep the associated value, generally a real number, within an acceptable range. The number of nodes per layer can vary, ranging from 10 for a very simple network to hundreds for advanced deep networks. Nodes are essentially structures that know how to methodically process the data passing through it.

*Edges & Weights:*

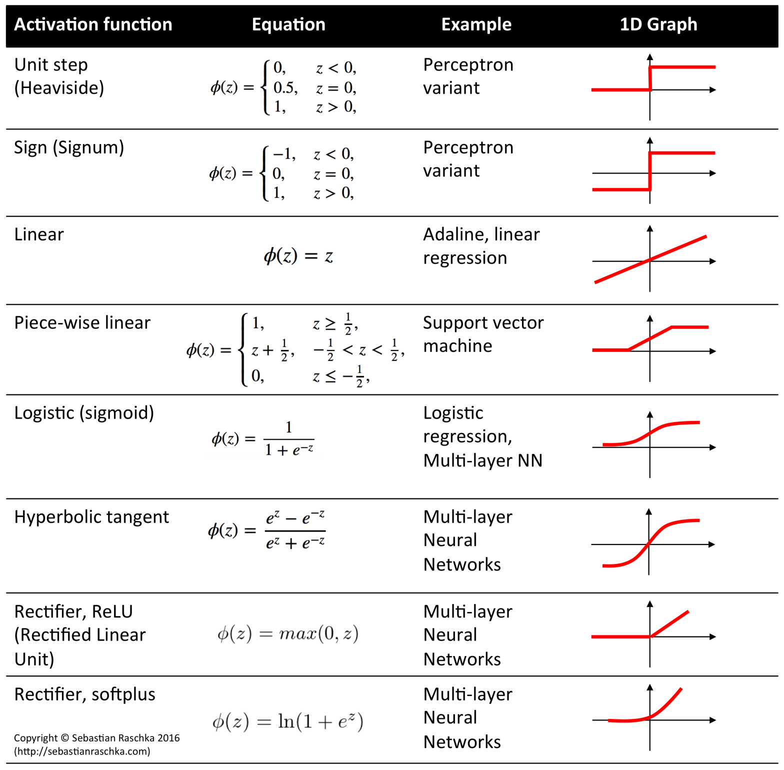
Like any directed graph, nodes between layers are connected by edges. The number of edges can be maximized as in fully connected layers or minimized as in sparsely connected layers. Typically, edges have an associated weight that alters the data being transferred through the edge. These weights are key to training neural networks to classify data accurately. In general, the higher the weight the more effective an input is in classification. The weights are constantly adjusted to manipulate the data so the correct classification score is output after all the processing is complete.

*Biases:*

Biases are constant values added to the value computed at each node. A bias is optional but very useful at making the model best fit the data. Biases are tuned during training just as weights are. For example, in a simple neural network a bias is equivalent to a $y$ intercept being added to a linear equation. It will shift the value of the data along an axis of a two dimensional plane. Some neural networks are more than two dimensions and therefore do not simply ”shift” along any one axis.

*Activation Functions:*

Activation functions are the operations responsible for determining the output of a neuron. Each neuron has an associated activation function, and often all nodes of a layer will share an activation function. The activation can act as a switch, turning a node on or off depending on a threshold, or as a transformer that alters the data so it is received with new information by the next neuron. See figure 3.4 for a list of common activation functions.



| Activation function | Equation | Example | 1D Graph |
|---|---|---|---|
| Unit step (Heaviside) | $\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$ | Perceptron variant | |
| Sign (Signum) | $\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$ | Perceptron variant | |
| Linear | $\phi(z) = z$ | Adaline, linear regression | |
| Piece-wise linear | $\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$ | Support vector machine | |
| Logistic (sigmoid) | $\phi(z) = \frac{1}{1 + e^{-z}}$ | Logistic regression, Multi-layer NN | |
| Hyperbolic tangent | $\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$ | Multi-layer Neural Networks | |
| Rectifier, ReLU (Rectified Linear Unit) | $\phi(z) = max(0, z)$ | Multi-layer Neural Networks | |
| Rectifier, softplus<br>Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com) | $\phi(z) = \ln(1 + e^{z})$ | Multi-layer Neural Networks | |

Figure 1.4: Activation Functions [47]

With these base elements, a neural network can be built that classifies input data. Depending on the kind of input, the input layer will be configured to receive all data points. The data will then be processed by one to hundreds of inner layers. At each layer the data will be transformed by the activation function, which will take in the predetermined edge weight and node bias as arguments. The data will finally reach the output layer and will be given a classification score to identify what type of input it likely is.

47. Sebastian Raschka, "Activation Functions," 2019, accessed December 9, 2019, http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/

## *Training Neural Networks*

Every neural network must be trained on test data to optimize the classification accuracy. Training involves running test data through the neural network, observing which inputs are misclassified, and then going back into the neural network to alter the weights of the edges to fix the misclassifications when the test data is run through again. Modern networks typically use a method known a Stochastic Gradient Descent to minimize a loss function to measure classification optimization. This is achieved over numerous training iterations called epochs. When the output layer receives a value produced by the network, the results reverse through the network through a process called back propagation. This process alters the biases and edge weights in a way that is likely to fix the misclassification. The loss associated with a neural network is the error in classification predictions. By repeatedly processing labeled test data the neural network can learn how to best classify the input.

A neural network must have lots of *good* test data to properly learn how to classify inputs over multiple epochs. *Good* data implies data that is varied, numerous, and labeled correctly. For example, if you train a neural network to classify an animal as a wolf on pictures that all display wolves in snow. Then the network may classify a wolf not pictured in snow as not a wolf because the network was trained to associate snow with being a wolf. In addition to images that are varied you want lots of pictures of wolves in different settings to optimally prepare the network for any input, and you naturally want each picture of a wolf correctly labeled as a wolf, not a dog, which would lead the network to misclassify the image.

Figure 1.5: Wolf in Denali National Park[48]

Figure 1.6: Wolf pictured with snow in background[49]

48. Nathan Kostegian, “Wolf in Denali National Park,” 2019, accessed December 9, 2019, https://www.nps.gov/dena/learn/nature/wolves.htm

49. MacNeil Lyons, “Wolf with Snow in Background,” 2019, accessed December 10, 2019, https://www.flickr.com/photos/usfwsmidwest/6545954933

## *Validating Neural Networks*

A validation data set is used to monitor how well the neural network is classifying inputs. This data is separate from test data and the neural network does not learn from these inputs. Processing the validation data set through the network helps insure the network is not overfitting. For example, if the accuracy on the test data is very good, but very poor on the validation data, overfitting is likely occurring. Overfitting is when the neural network makes strong predictions based on very specific details. An example would be a network classifying a wolf based on snow in the image rather than the specific details of the animal. Processing validation data is an objective measure of accuracy because the network has not learned on these inputs. Therefore, it is essential that validation data is varied, numerous and labeled correctly, just like training data.

## *Features of state of the art neural networks*

The architectural components of neural networks discussed thus far have been the essential elements of a neural network. These elements are required to construct a state-of-the art neural network, but they are no longer sufficient to build a top performing neural network. There have been several advances in network architecture to improve classification accuracy. Modern transfer learning models all use a combination of these features to maximize performance. Some of the common features are discussed below.

*Convolutional layers:*

Nodes in a common neural network layer are connected to several nodes in the following layer. In convolutional layers, the nodes are not typically connected to many nodes in the next layer. Several nodes in the convolutional layer are referred to as receptor fields and the convolutional layer attempts to build a feature map from these fields. A feature map is essentially a 2 dimensional grid of values that highlight where a feature may exist. Each value represents the likelihood of a feature existing at that position in the data. Imagine taking all the pixels of an image as the input and the first convolutional layer attempting to find all straight edges as the features. If the pixel value data is displayed as a 2 dimensional grid, the n x n filter would slide over the entire grid to filter out values that correspond to an edge. Convolutions are used to build a feature map, which is why a layer implementing convolutions is called a convolutional layer. Convolutions are the process of convolving a two dimensional kernel/filter with predefined weights. Each convolution reduces the filter to a single output to identify a feature.

Convolutional layers are also different because all nodes share the same weight and bias. This allows features to be detected anywhere on an image, and at different scales. This works because convolutional layers typically build a feature map by looking for one feature at a time. Directly following pooling layers can further help convolutional layers identify features in an input.

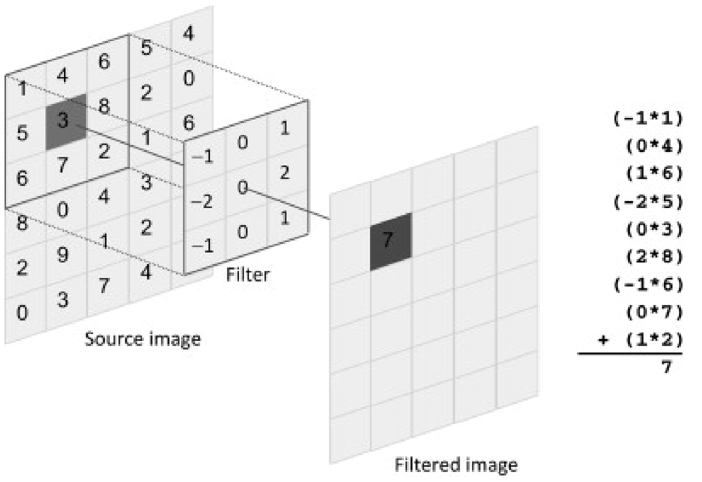


Figure 1.7: Convolution Filter[50]

*Max pooling layers:*

In general, max pooling layers reduce the size of the input. If you imagine a two dimensional data set like an image, consider max pooling shrinking the input to half its size, but keeping the same scale. Max pooling acts upon data according to an n x n filter and a stride. The stride is the distance the map moves over the data. A pooling layer is called a max pooling layer when the max value is computed within the filter each time it strides over the data.

50. Dong Ping Zhang David Kaeli, "Convolution Filter," 2019, accessed December 10, 2019, https://www.sciencedirect.com/topics/computer-science/convolution-filter

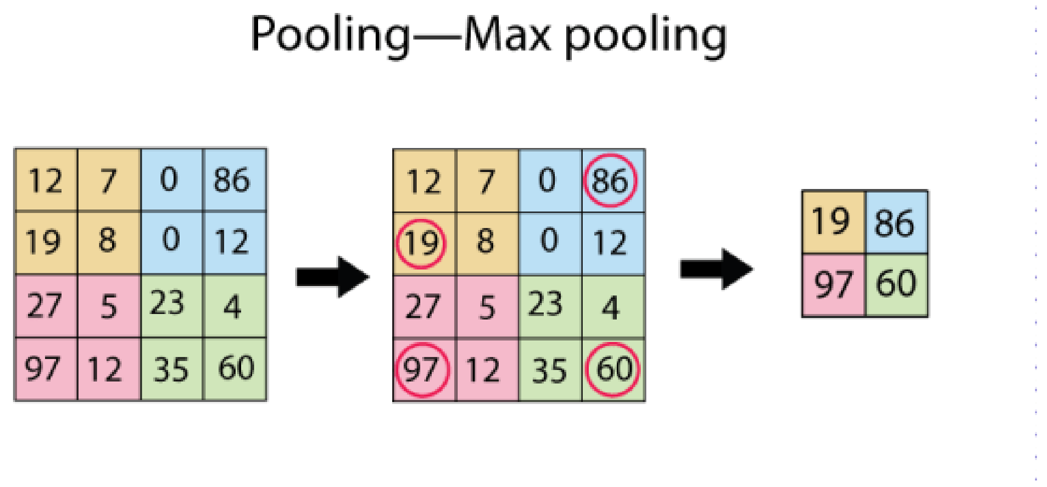


Figure 1.8: Max Pooling Example[51]

*Many hidden layers:*

Transfer learning models may have hundreds of hidden layers, whereas earlier neural networks may have typically had fewer than 100. The number of layers is a result of adding different types of layers to operate on the data, processing larger sizes of input, and having the computational capacity to process data through hundreds of layers and back propagating to train the model.

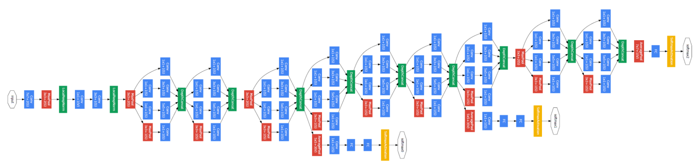


Figure 1.9: Inception-v1 Architecture[52]

51. Craig Will, "Max Pooling Filter," 2019, accessed December 10, 2019, https://principlesofdeeplearning.com/index.php/2018/08/27/is-pooling-dead-in-convolutional-networks/

52. Szegedy et al., "Going deeper with convolutions"

*Frozen layers:*

Layers are frozen to speed up the time it takes to train a neural network. When a layer is frozen the weights associated with that layer no longer can be altered. Researchers have found that some layers stop improving accuracy in training before other layers and it is effective to freeze them and focus processing power on the untrained layers to speed up the process. Transfer learning models commonly have many of the first layers frozen to speed up training by the user. This allows the user to train more cost effectively because only some of the layers are being processed while not sacrificing any significant amount of accuracy.

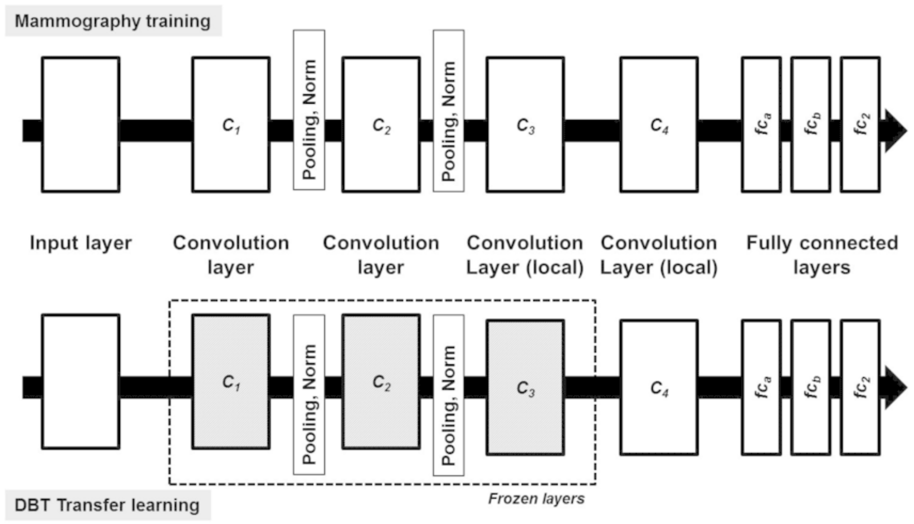


Figure 1.10: Frozen Layers of Model Example[53]

53. Ravi Samala et al., "Mass detection in digital breast tomosynthesis: Deep convolutional neural network with transfer learning from mammography," *Medical Physics* 43 (December 2016): 6654, doi:10.1118/1.4967345]

## *Machine Learning Transfer Models*

Transfer learning models are highly accurate classification tools that are offered for reuse. These models have been built and trained by experts from cutting edge organizations, such as Google, Microsoft, and the University of Oxford. Transfer models have also had the benefit of being trained on the most expansive data sets available. Transfer models have become so sophisticated that it is now expensive and difficult to build superior models. The architecture of these models has been finely tuned by machine learning and data science experts to reach peak classification accuracy. Furthermore, these models are being used more and more as machine learning becomes more prevalent. One may deploy this model in their own program, train it on their specific data set, modify the architecture, and then use it as they please. By doing this one takes advantage of all the work put into these models that would be difficult to emulate in a scaled down setting.

In the next section we will look at an overview of five of the top transfer learning neural networks: Inception-V1 (GoogLeNet),[54] Inception-V3,[55] Inception-V4,[56] Resnet,[57] and VGGnet.[58] These models are all state-of-the art and highly praised

54. Szegedy et al., "Going deeper with convolutions."

55. Christian Szegedy et al., "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 2818–2826.

56. Christian Szegedy et al., "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence* (2017).

57. Kaiming He et al., "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 770–778.

58. Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014,

in the machine learning community. We will analyze these models to understand what the Prime+Probe attack will aim to learn. Additionally, by understanding the complexity of these models, we hope to provide background for why we chose the methods of attack proposed in this paper to learn which model a program is running.

*Inception-V1*

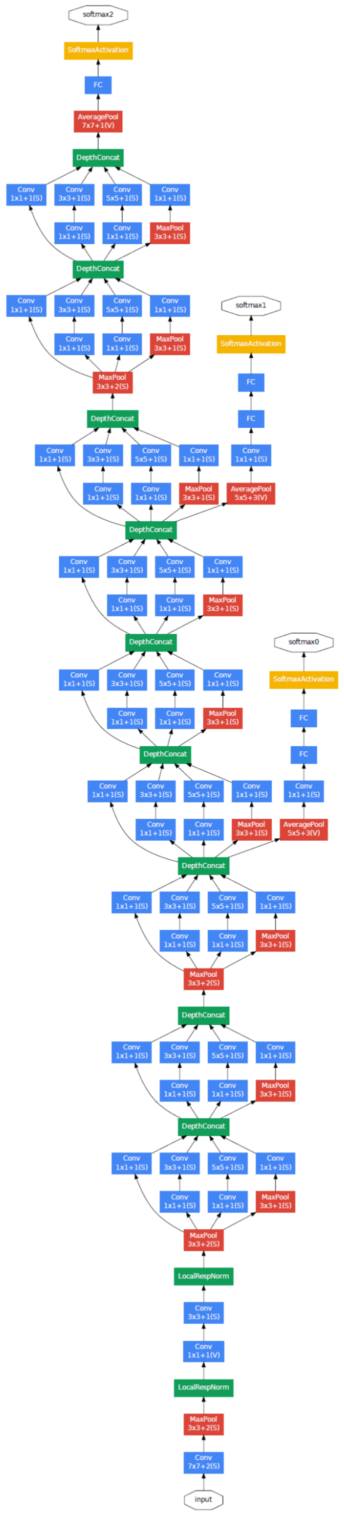


Figure 1.11: Inception-v1 Neural Network[59]

Inception-v1, also known as GoogLeNet,[60] emerged from Google in 2014 to compete in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC).[61] This neural network was designed to classify images as the name of the competition indicates it would need to do. The designers of Inception-v1 state that advances in this network are more a result of ”new ideas, algorithms and improved network architecture” than ”more powerful hardware, larger data sets and bigger models.”[62]

Inception-v1 is a 22 layer deep convolutional neural network. This network uses 12 times fewer parameters (weights, biases, etc.) than the leading image classification neural network of 2012. This network is also deeper than most earlier models, such as AlexNet[63] which had eight layers and VGGNet[64] which had nineteen layers.

59. Szegedy et al., “Going deeper with convolutions”

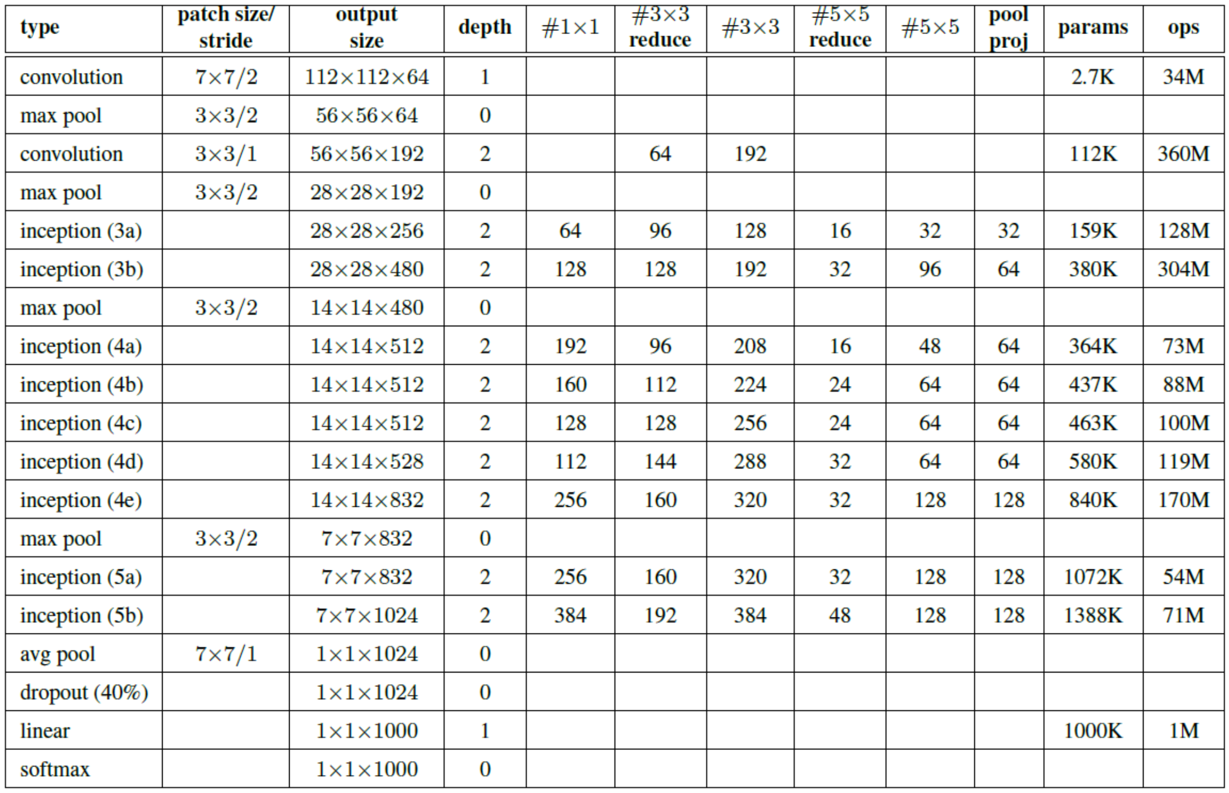
60. Ibid.

61. Olga Russakovsky et al., “Imagenet large scale visual recognition challenge,” *International journal of computer vision* 115, no. 3 (2015): 211–252.

62. Szegedy et al., “Going deeper with convolutions.”

63. Ilya Sutskever, Geoffrey E Hinton, and A Krizhevsky, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, 2012, 1097–1105.

64. Simonyan and Zisserman, “Very deep convolutional networks for large-scale image recognition.”



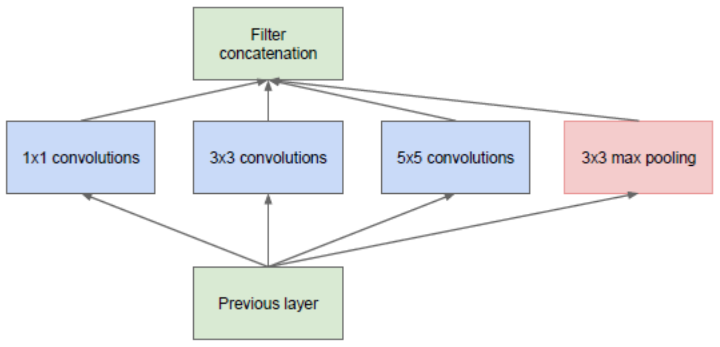
| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Figure 1.12: Inception-v1 Layers[65]

The following features make this network unique: 1 X 1 convolution filters, the inception module, and global average pooling.[66] Like other convolutional filters, the 1 x 1 filter is used to to reduce the size of the data being processed. This addition reduced the number of operations that the network must perform and therefore made it possible to add additional layers without making the model exceptionally inefficient. Specifically, this allowed the designers to include the inception module.

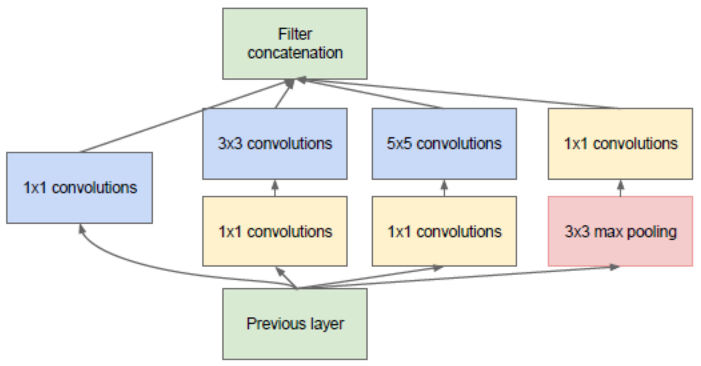
65. Szegedy et al., "Going deeper with convolutions"

66. Ibid.



(a) Inception module, naïve version

Figure 1.13: Inception-v1 Module Naive[67]



(b) Inception module with dimensionality reduction

Figure 1.14: Inception-v1 Module[68]

The inception module is made possible by the addition of 1 x 1 filters. By carefully adding 1 x 1 filters total operations are reduced and more layers may be inserted. As you can see by comparing figure 4.13 and 4.14, the inception module includes two additional 1 x 1 filters. The result of the inception module is dimension reduction of the data, less computation, and reduced overfitting.

Global average pooling rather than fully connected layers is used near the end

67. Szegedy et al., “Going deeper with convolutions”

68. ibid.

of the network. Global average pooling averages each 7 x 7 feature map to a 1 x 1 feature map and does not have weights. This reduction in data greatly lessens the computation requirements while maintaining accuracy. By using global average pooling accuracy went up by roughly 0.6% and overfitting was reduced.

*Inception-V3*

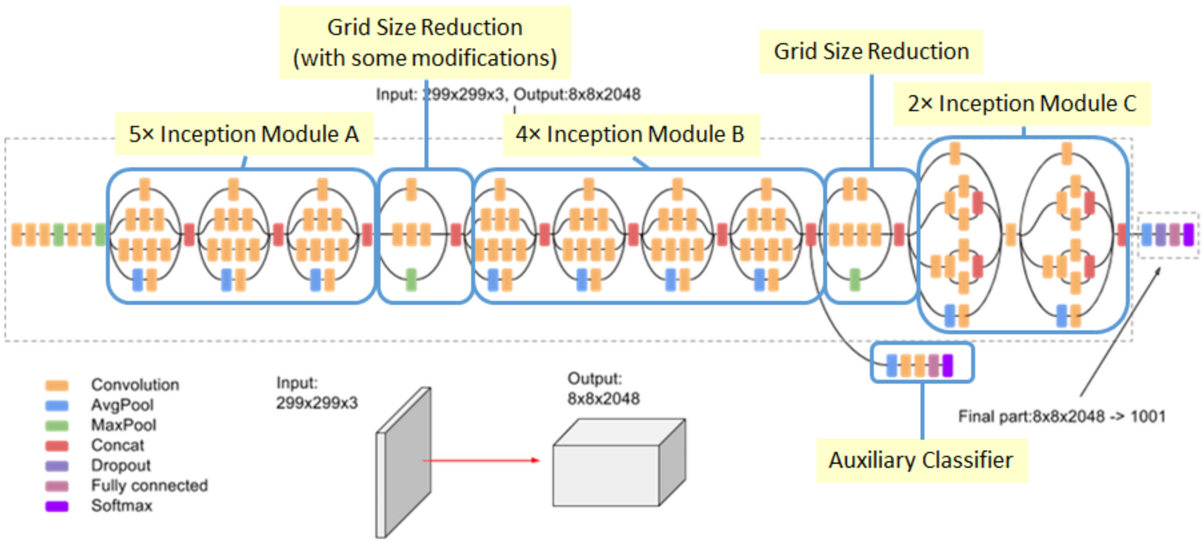


Figure 1.15: Inception-v3[69]

Inception-v3 like the other versions of the Inception models is designed and built by researchers at Google. Inception-v3 was built in 2015 and was used to compete in the ILSVRC.[70] This neural network relied heavily on Inception-v1 as a foundation to improve upon.

The following features make this network unique: factorizing convolutions,

69. Szegedy et al., "Rethinking the inception architecture for computer vision"

70. Russakovsky et al., "Imagenet large scale visual recognition challenge."

auxiliary classifiers, and batch normalization.[71] Factorizing convolutions was introduced to maintain efficiency while introducing data reduction. One way this is achieved is by replacing a 5 x 5 convolution filter with two 3 x 3 filters. By 3 x 3 filters "the number of parameters is reduced by 28%".[72] Additionally, this improvement can be achieved by replacing a 3 x 3 filter with two other filters, a 3 x 1 filter and a 1 x 3 filter.

Auxiliary classifiers are essentially mini neural networks inserted into the overall architecture. In this network only one auxiliary classifier is inserted. The purpose of these classifiers is to compute a loss that is added to the overall network loss during training time. Additionally, in Inception-v3 the auxiliary classifier is used as a "regularizer".[73] Weight regularization is primarily used to reduce overfitting in the model.

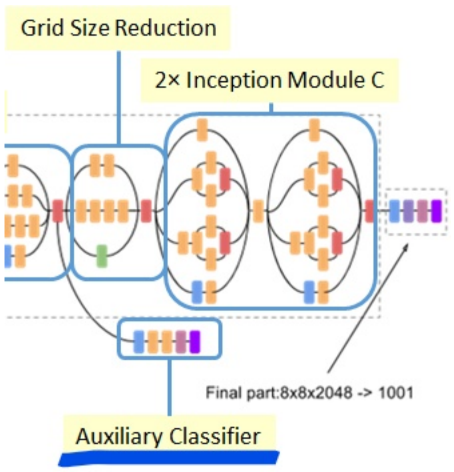


Figure 1.16: Inception-v3 Auxiliary Classifier [74]

71. Szegedy et al., "Rethinking the inception architecture for computer vision."

72. Ibid.

73. Ibid.

Batch normalization reduces overfitting, lessens dropout, and increases accuracy in the face of covariance shift.[75] It does this by normalizing the resulting calculation from nodes so they are never abnormally high or low. Two trainable parameters are inserted specific layers so the network can alter the values with each epoch to find the optimum normalization. Dropout is the process of randomly nullifying data points in input to force the network to account for atypical inputs. Covariance shifts is the idea of introducing a type of input that is similar in most regards, but very different in at least one regard, which often leads to misclassification. For example, consider neural network trained to detect stop signs. We know stop sign are typically hexagonal. However, what if there were also square stop signs and the network needed to be trained on them? Everything about the signs may be identical, but because the shape is different, the network will likely have a high misclassification rate on these inputs.

### *Inception-V4*

Inception-v4 is the most recent generation of the Inception neural networks built by teams at Google. This version of the Inception neural networks was designed in 2016 and it also was used to compete in the ImageNet classification challenge. This generation heavily relies on the architecture of its predecessors as did Inception-v3. However, residual connections, as introduced by the ResNet neural network built by

74. Szegedy et al., "Rethinking the inception architecture for computer vision"

75. Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015,

Microsoft,[76] were added to speed up the training process.

The creators of Inception-v4 observe that this network has a simpler architecture than its Inception-v3.[77] In addition to adding residual connections, it is different because it has more inception modules than its predecessor. However, the marginal gains are slimmer than the gains observed in the earlier Inception networks.[78]

Residual blocks or connections occur when each layer feeds into the immediately following layer and layers not immediately following the layer. These layers are called residual layers, rather than skip layers which may be more intuitive, because the layer receiving input from the residual layer is actually seeing the residual output from that layer, not the true output as sequential layers see. This idea emerged from the challenge researchers faced of how to improve accuracy without simply adding more layers to the network. At this time adding more layers began producing diminishing marginal returns in improved accuracy and new ideas were needed to realize better performance.

Inception-v4 specifically has three primary inception modules. The design of these modules follows the design originally introduced in Inception-v1. Additional modules and the ability to handle the additional overload allows this network to compound the benefits of this feature. There is better data reduction, fewer computations required, and lessened overfitting.

76. He et al., "Deep residual learning for image recognition."

77. Szegedy et al., "Inception-v4, inception-resnet and the impact of residual connections on learning."

78. Ibid.

*Resnet*

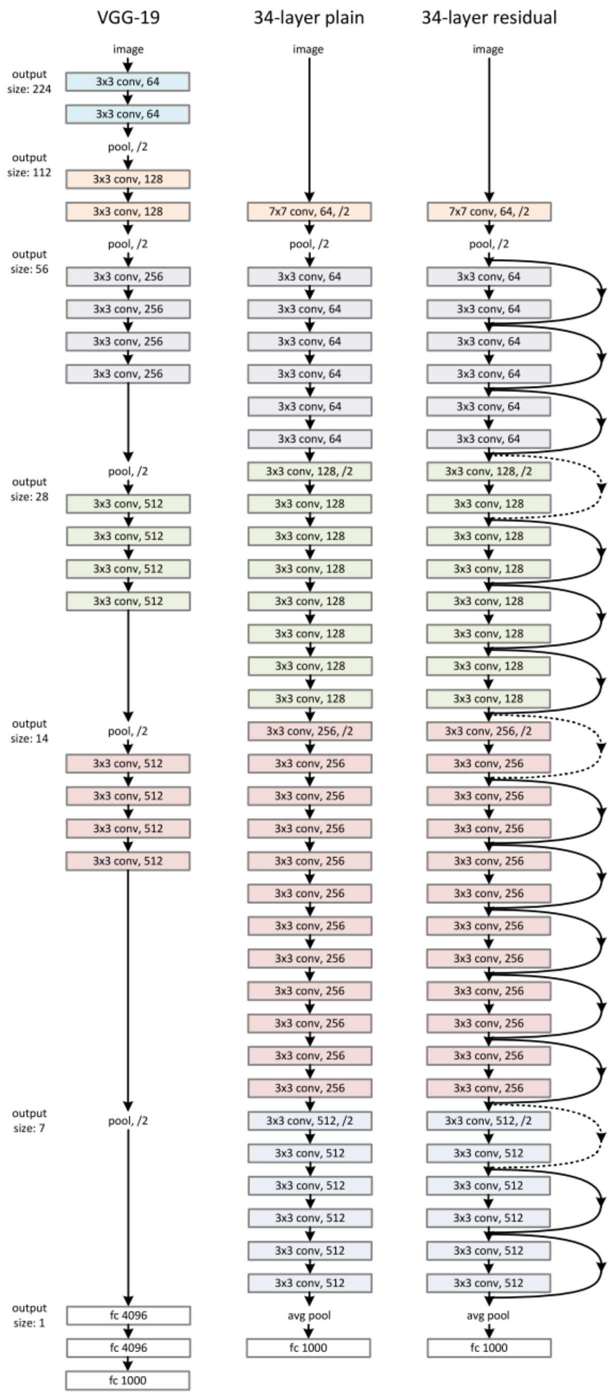
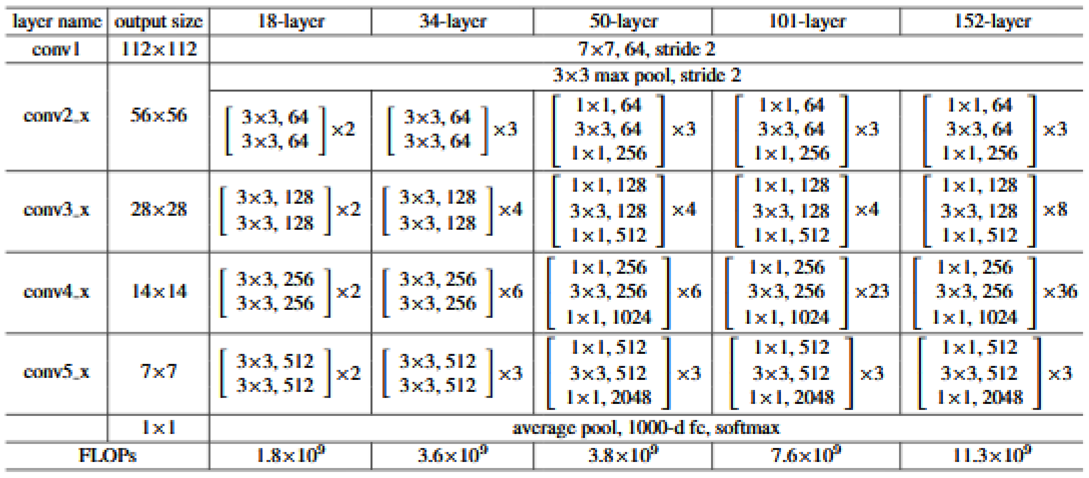


Figure 1.17: Residual Architecture[79]



| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Figure 1.18: Residual Architecture Breakdown[80]

Resnet defines a family of neural networks that utilize residual network architecture. This innovative architecture feature emerged in 2015 from teams at Microsoft Research.[81] Since being introduced to the machine learning community, numerous transfer models have included residual layers in their architecture. We will consider here the ResNet neural network as it was introduced in 2015. This is the network that won the ILSVRC competition in 2015 with an error race of 3.57% on the ImageNet data set.[82]

Teams at Microsoft built residual networks with up to 152 layers to prove these networks could be extended greatly without significantly increasing complexity. In 2015, a 152 layer network was 8 times deeper than VGG networks, another state-of-the-art transfer model. Like with Inception-v4, residual networks add residual layers

79. He et al., "Deep residual learning for image recognition"

80. ibid.

81. Ibid.

82. Ibid.

so the model can have better accuracy, but will not be inefficient and suffer from diminishing marginal returns of added complexity and overfitting.

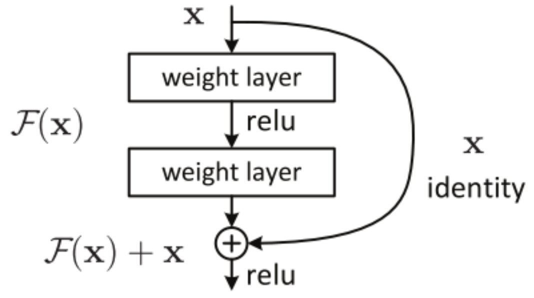


Figure 1.19: Residual Block[83]

The Resnet model originally had 34 layers. The majority of these layers were convolutional. the convolutional layers used a 3 x 3 filter and a stride of 2. The last two layers are a global average pooling layer and a 1000-way fully connected layer using the softmax activation function to prepare the data for classification. The designers of ResNet note that this network has "fewer filters and lower complexity than VGG nets."[84] It has 3.6 billion multiply-adds vs 19.6 billion multiply-adds in VGG.[85]

In addition to having 13 residual layers, Resnet has three shortcut layers to adjust identity mapping for inputs with increased dimensions. The shortcut layers pad zeros to the data rather than introduce additional parameters, or they use an equation to match the dimension to what the network expects. This helps the neural network classify varying sizes of input because it may happen that not all images in

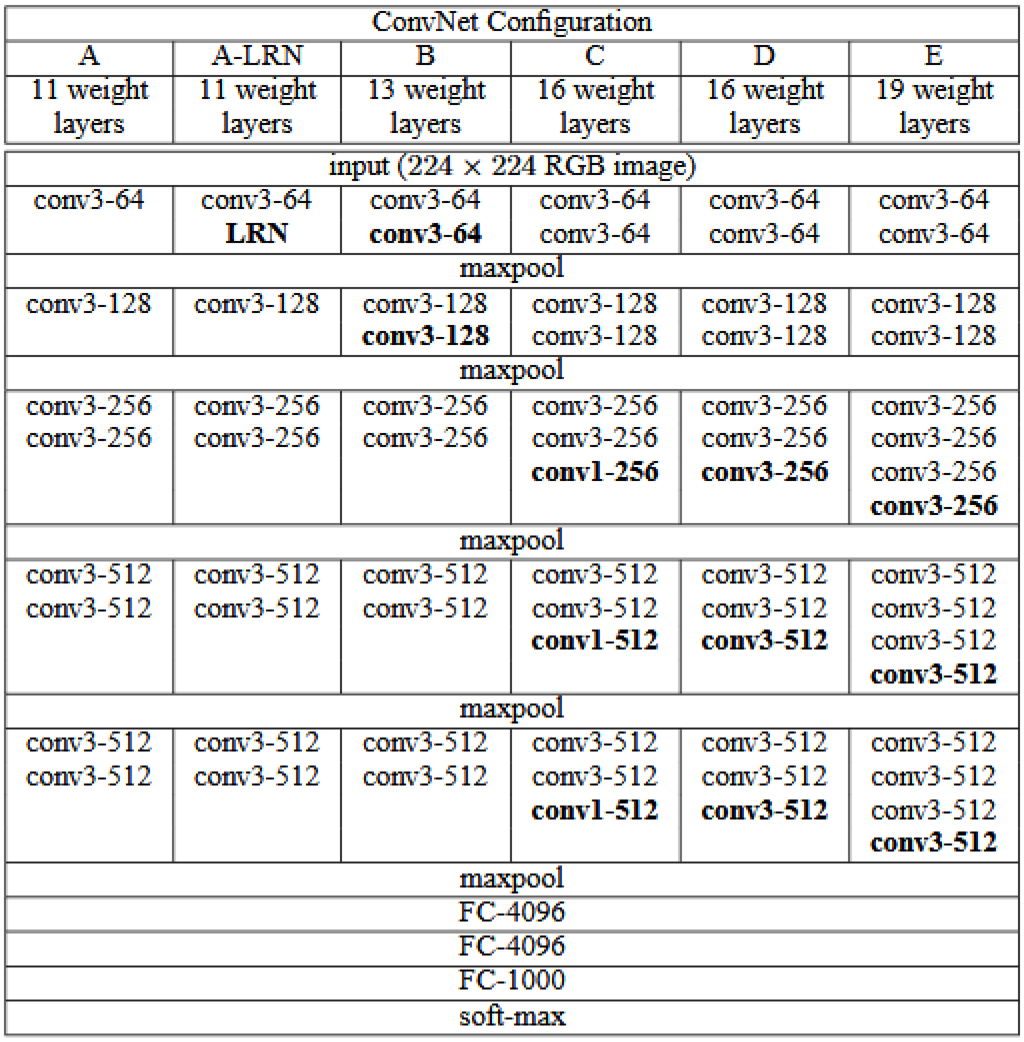
83. He et al., "Deep residual learning for image recognition"

84. Ibid.

85. Ibid.

a data set are uniform in size.

*VGGNnet*



| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64 | conv3-64 | conv3-64 | conv3-64 | conv3-64 |
| | **LRN** | **conv3-64** | conv3-64 | conv3-64 | conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 | conv3-128 | conv3-128 | conv3-128 |
| | | **conv3-128** | conv3-128 | conv3-128 | conv3-128 |
| maxpool | | | | | |
| conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 |
| conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 |
| | | | **conv1-256** | **conv3-256** | conv3-256 |
| | | | | | **conv3-256** |
| maxpool | | | | | |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 |
| | | | **conv1-512** | **conv3-512** | conv3-512 |
| | | | | | **conv3-512** |
| maxpool | | | | | |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 |
| | | | **conv1-512** | **conv3-512** | conv3-512 |
| | | | | | **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Figure 1.20: VGG Model Layer Breakdown[86]

86. Simonyan and Zisserman, “Very deep convolutional networks for large-scale image recognition”

VGG was built by the Visual Geometry Group based at the University of Oxford.[87] This model was 1st runner-up in the 2014 ILSVRC ImageNet classification competition, but won the localization task. The localization task is where the model must bound objects it identifies in an image. VGG was praised at this time for being a significant improvement over ZFNet and AlexNet, the two models that won the ILSVRC competition in 2013 and 2012 respectively. This was the first year where models achieved an error rate of less than 10%. Additionally, models today are still being designed using VGG net as a baseline.

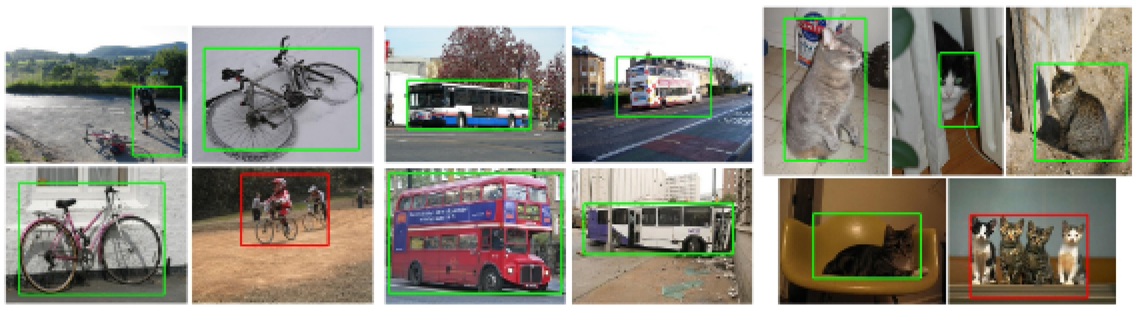


Figure 1.21: Localization Task[88]

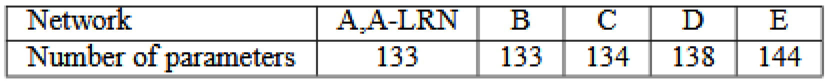
While there were six VGG networks (VGG-11, VGG-11 (LRN), VGG-13, VGG-16 (Conv1), VGG-16, and VGG-19),[89] VGG-16 and VGG-19 are the most commonly discussed and used. The primary reason is that these two versions of VGG

87. Simonyan and Zisserman, "Very deep convolutional networks for large-scale image recognition."

88. Matthew B Blaschko and Christoph H Lampert, "Learning to localize objects with structured output regression," in *European conference on computer vision* (Springer, 2008), 2–15

89. Simonyan and Zisserman, "Very deep convolutional networks for large-scale image recognition."

achieved the lowest error rates. It is interesting to note that these two models have the most parameters as well. However, VGG-16 achieved the best accuracy using fewer layers and parameters than VGG-19. VGG-16 acheived an 8.8% error rate while VG-19 achieved a 9.0% error rate.[90]



| Network | A,A-LRN | B | C | D | E |
|---|---|---|---|---|---|
| Number of parameters | 133 | 133 | 134 | 138 | 144 |

Figure 1.22: VGG Number of Parameters [91]

The VGG-16 and VGG-19 models use 16 layers and 19 layers respectively. The majority of these layers are convolutional for feature mapping. Most of the filters are 3 x 3, but some 1 x 1 filters are also used. The stride is uniformly set to 1. Max pooling is also implemented in five layers using a 2 x 2 filter with a stride of 2. For all models three fully connected layers are added to the end of the network to prepare to map the output. The final layer has a 1000 nodes to map the output to one of the 1000 categories in the ImageNet classification competition. Every hidden layer uses the ReLu (Rectifyer-Linear) activation function.

VGG overall is different than the two previous ImageNet competition winners because it consistently uses small filters in its convolutional layers. VGG uses 3 x 3 filters whereas AlexNet[92] used 11 x 11 filters with a stride of 4, and ZFNet[93] used

90. Simonyan and Zisserman, "Very deep convolutional networks for large-scale image recognition."

91. He et al., "Deep residual learning for image recognition"

92. Sutskever, Hinton, and Krizhevsky, "Imagenet classification with deep convolutional neural networks."

93. Matthew D Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision* (Springer, 2014), 818–

7 x 7 filters with a stride of 2. The 1 x 1 filters used in VGG were taken from the GoogLeNet project (Inception-v1) to increase the depth of the model without hitting diminishing marginal returns of added complexity.

## *Machine Learning and Neural Networks for Cybersecurity*

In general, there is no machine learning/deep learning technology or transfer model built uniquely for cybersecurity. Rather, existing ML models are trained to solve a problem relating to cybersecurity. One example is training machine learning model to identify a virus based on viewing the code binary.[94] The ML model may be the same as a model trained to identify a binary specific to a certain company or organization, but is trained on data unique to cybersecurity to solve this problem.

Additionally, ML models used in cybersecurity may be altered to harden them against adversarial attacks intended to trick the model or reverse engineer it. It is recommended that these models be hardened against attacks because they likely perform a critical function that must not fail. An adversary may be motivated to trick the model to, for example, get a piece of malware to be classified as benign and able to be installed on a system. An adversary may also be motivated to reverse engineer a model to be able to test on it in a way that does not garner suspicion to ultimately know how to trick the model in the future. How adversarial may try to trick or reverse engineer ML models will be discussed in more detail later.

---

833.

94. Olivier Henchiri and Nathalie Japkowicz, "A feature selection and evaluation scheme for computer virus detection," in *Sixth International Conference on Data Mining (ICDM'06)* (IEEE, 2006), 891–895.

## References

Bailey, Kyle O, James S Okolica, and Gilbert L Peterson. "User identification and authentication using multi-modal behavioral biometrics." *Computers & Security* 43 (2014): 77–89.

Barrett, Brian. "Hack Brief: An Astonishing 773 Million Records Exposed in Monster Breach." 2019. Accessed January 16, 2019. https://www.wired.com/story/collection-one-breach-email-accounts-passwords/.

Bhattacharyya, Debnath, Rahul Ranjan, Farkhod Alisherov, Minkyu Choi, et al. "Biometric authentication: A review." *International Journal of u-and e-Service, Science and Technology* 2, no. 3 (2009): 13–28.

Blaschko, Matthew B, and Christoph H Lampert. "Learning to localize objects with structured output regression." In *European conference on computer vision*, 2–15. Springer, 2008.

Buczak, Anna L, and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection." *IEEE Communications surveys & tutorials* 18, no. 2 (2015): 1153–1176.

David Kaeli, Dong Ping Zhang. "Convolution Filter." 2019. Accessed December 10, 2019. https://www.sciencedirect.com/topics/computer-science/convolution-filter.

Fraley, James B, and James Cannady. "The promise of machine learning in cybersecurity." In *SoutheastCon 2017*, 1–6. IEEE, 2017.

Gavriluţ, Dragoş, Mihai Cimpoeşu, Dan Anton, and Liviu Ciortuz. "Malware detection using machine learning." In *2009 International Multiconference on Computer Science and Information Technology*, 735–741. IEEE, 2009.

Godefroid, Patrice, Hila Peleg, and Rishabh Singh. "Learn&fuzz: Machine learning for input fuzzing." In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 50–59. IEEE, 2017.

Gogoi. "ann-vs-bnn." 2019. Accessed December 9, 2019. https://blog.knoldus.com/first-interaction-artificial-neural-network/.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Advances in neural information processing systems*, 2672–2680. 2014.

Güera, David, and Edward J Delp. "Deepfake video detection using recurrent neural networks." In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 1–6. IEEE, 2018.

Hall, Charles H. *Operational Art in the Fifth Domain.* Technical report. NAVAL WAR COLL NEWPORT RI JOINT MILITARY OPERATIONS DEPT, 2011.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778. 2016.

He, Warren, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. "Adversarial example defense: Ensembles of weak defenses are not strong." In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17).* 2017.

Henchiri, Olivier, and Nathalie Japkowicz. "A feature selection and evaluation scheme for computer virus detection." In *Sixth International Conference on Data Mining (ICDM'06)*, 891–895. IEEE, 2006.

Hong, Sanghyun, Michael Davinroy, Yiitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. "Security analysis of deep neural networks operating in the presence of cache side-channel attacks." *arXiv preprint arXiv:1810.03487*, 2018.

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167*, 2015.

Kali. "Kali Linux." 2020. Accessed March 29, 2020. https://www.kali.org/.

Kocher, Paul, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, et al. "Spectre Attacks: Exploiting Speculative Execution." In *40th IEEE Symposium on Security and Privacy (S&P'19).* 2019.

Kostegian, Nathan. "Wolf in Denali National Park." 2019. Accessed December 9, 2019. https://www.nps.gov/dena/learn/nature/wolves.htm.

Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale." *arXiv preprint arXiv:1611.01236*, 2016.

Lipp, Moritz, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, et al. "Meltdown: Reading Kernel Memory from User Space." In *27th USENIX Security Symposium (USENIX Security 18).* 2018.

Liu, Fangfei, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. "Last-level cache side-channel attacks are practical." In *2015 IEEE Symposium on Security and Privacy*, 605–622. IEEE, 2015.

Lyons, MacNeil. "Wolf with Snow in Background." 2019. Accessed December 10, 2019. https://www.flickr.com/photos/usfwsmidwest/6545954933.

Magee, Kenneth. "The CISSP Domains - An Overview." 2020. Accessed March 28, 2020. https://resources.infosecinstitute.com/the-cissp-domains-an-overview/#gref.

Monitor, Respondus. "Respondus Monitor." 2020. Accessed April 17, 2020. https://web.respondus.com/he/monitor/.

Olsen, Christoffer. "A Machine Learning Approach to Predicting Passwords," 2018.

Raschka, Sebastian. "Activation Functions." 2019. Accessed December 9, 2019. http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/.

Ross, Candace, Andrei Barbu, Yevgeni Berzak, Battushig Myanganbayar, and Boris Katz. "Grounding language acquisition by training semantic parsers using captioned videos." In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2647–2656. 2018.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision* 115, no. 3 (2015): 211–252.

Sadeghi, K., A. Banerjee, J. Sohankar, and S. K. S. Gupta. "Performance and Security Strength Trade-Off in Machine Learning Based Biometric Authentication Systems." In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 1045–1048. 2017.

Samala, Ravi, Heang-Ping Chan, Lubomir Hadjiiski, Mark Helvie, Jun Wei, and Kenny Cha. "Mass detection in digital breast tomosynthesis: Deep convolutional neural network with transfer learning from mammography." *Medical Physics* 43 (December 2016): 6654. doi:10.1118/1.4967345].

Shon, Taeshik, and Jongsub Moon. "A hybrid machine learning approach to network anomaly detection." *Information Sciences* 177, no. 18 (2007): 3799–3821.

Silipo, Rosaria. "Can AI write like Shakespeare?" 2020. Accessed March 28, 2020. https://towardsdatascience.com/can-ai-write-like-shakespeare-de710befbfee.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556*, 2014.

"simple-neural-network." 2019. Accessed December 9, 2019. https://www.researchgate.net/figure/A-simple-neural-network-with-two-hidden-layers-of-two-nodes-each-four-inputs-and-a_fig1_327637282.

Sivakorn, Suphannee, Iasonas Polakis, and Angelos D Keromytis. “I am robot:(deep) learning to break semantic image captchas.” In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 388–403. IEEE, 2016.

Stark, Fabian, Caner Hazrba, Rudolph Triebel, and Daniel Cremers. “CAPTCHA Recognition with Active Deep Learning.” September 2015.

Sutskever, Ilya, Geoffrey E Hinton, and A Krizhevsky. “Imagenet classification with deep convolutional neural networks.” *Advances in neural information processing systems*, 2012, 1097–1105.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions.” In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9. 2015.

Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. “Inception-v4, inception-resnet and the impact of residual connections on learning.” In *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision.” In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826. 2016.

VIRUSTOTAL. “VIRUSTOTAL.” 2020. Accessed March 28, 2020. https://www.virustotal.com/gui/intelligence-overview.

Will, Craig. “Max Pooling Filter.” 2019. Accessed December 10, 2019. https://principlesofdeeplearning.com/index.php/2018/08/27/is-pooling-dead-in-convolutional-networks/.

Yamaguchi, Fabian, Felix Lindner, and Konrad Rieck. “Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning.” In *Proceedings of the 5th USENIX conference on Offensive technologies*, 13–13. USENIX Association, 2011.

Yarom, Yuval, and Katrina Falkner. “FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack.” In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 719–732. 2014.

Yüksel, Ulas, and Hasan Sözer. “Automated classification of static code analysis alerts: a case study.” In *2013 IEEE International Conference on Software Maintenance*, 532–535. IEEE, 2013.

Zeiler, Matthew D, and Rob Fergus. “Visualizing and understanding convolutional networks.” In *European conference on computer vision*, 818–833. Springer, 2014.