**Introduction**

This document reports on Task 2.1 (Transfer Learning) from the DHS funded project, QuASAR. The focus of Task 2.1 was applying Transfer Learning (TL) to the problem of identifying and classifying malware and vulnerabilities in binaries. Machine Learning (ML) has attracted substantial interest in the cyber security community [1]. Applications include defending network attacks, malware classification, system-event-based anomaly detection, security-oriented program analysis, return-oriented programming attacks, control-flow integrity, memory forensics, and fuzzing for software security. While promising, early ML techniques (before 2010) were stuck in the labs and not in actual use. Even today, commercial use remains limited. Another identified gap is the practical application of ML to software quality assessment.

The main culprits for these gaps in applying ML to cybersecurity appear to be (1) finding well-defined targets specific enough to reduce false positives and (2) finding the appropriate feature set from which to learn. Deep Learning (DL) [2] has impacted solving these hurdles in many domains. DL exploits large data sets with many hidden layers and has made significant leaps in natural language processing, image classification, object detection, and speech recognition—problems that have challenged researchers for decades. DL is good at defining targets to increase accuracy and select relevant features. Thus, DL could be a game changer for cyber security as well.

Unfortunately, DL requires a large dataset of labeled examples. Given the ever-changing nature of cybersecurity, finding enough current code samples is often impractical. Enter Transfer Learning (TL). Humans learn better by relating to something they already know. For instance, an expert saxophonist should learn to play the clarinet faster than a musical novice. TL is the embodiment of this concept for ML and works by transferring knowledge learned in one task to become the starting point for another. TL is the primary way to address limited data by transferring models, rules, and examples from similar tasks. TL has proven to increase accuracy, simplify the workflow, and reduce computational requirements [3]. Figure 1 shows three ways TL can increase accuracy per the number of examples: a higher accuracy with few examples, a steeper slope in learning accuracy as the number of examples increases, or a higher asymptote of accuracy as examples become numerous [4]. It is important to know when and how to transfer since negative transfer can occur. TL applies to deep learning since data requirements for increased accuracy grow linearly with the size of the network . Thus, the question arises, can one ever have enough data? The related question is, will TL always be applicable?

Diagram

Description automatically generated

**Figure 1.** Three ways TL improves accuracy (From Torrey & Shavlik [4]).

Many predict TL will be the next top trend for ML since it helps businesses overcome cold-start problems and the high cost of amassing training data [5] [6] [7]. TL already has a substantial presence in workshops, proceedings, and publications [8] [9] [3]. Andrew Ng, a top AI scientist, presented Figure 2 in his popular NIPS talk [5]. The chart maps the commercial success of the four top components of ML: supervised learning (labeled data), unsupervised learning (unlabeled data), reinforcement learning (multiple actions before feedback), and TL. The chart points out the current success of supervised learning and predicts TL will follow a similar path.

Diagram

Description automatically generated

**Figure 2**. Drivers of Success [5]

The reported research aims to demonstrate the utility of TL in cybersecurity, then provide a path forward by proposing a platform for commercialization.

**Background & Methodology**

**Learning Algorithms.** This project investigated the efficacy of transfer learning on three supervised learning algorithms (see the Background section for more information):

1. Feedforward Neural Networks (FNNs).
2. Convolutional Neural Networks (CNNs), including ResNet50 for digital images.
3. Long Short-Term Memory (LSTM) networks for strings of instruction counts.

**Feedforward Neural Networks (FNNs).** The first neural network we evaluated was a fully connected feedforward neural network trained with backpropagation. Our networks contained five hidden layers with ten nodes apiece. We set the hyper-parameters for the feedforward nets through PyTorch Lightning. For example, the early stopping setting determines the number of epochs. The learning rate of 1e-3 is the default found empirically to be most successful, so we use that learning rate throughout our experiments to ensure consistency between the tested models.

**Convolutional neural networks (CNNs).** A CNN is a deep learning neural network with many feedforward layers [2]. CNNs excel at computer vision problems and can operate directly on raw images rather than require preprocessing. The power of CNNs in computer vision comes from stacks of convolutional layers which progressively identify more complex features in the image. A CNN typically consists of combinations of convolutional layers, activation layers, and pooling layers.

A convolutional layer consists of many small square templates (i.e., convolutional kernels) which look for patterns by sliding over the image. The kernel returns a large value when the part of the image matches the kernel; otherwise, it returns a small value. A pooling layer reduces a set of numbers into a smaller set. A common approach is to take each 2x2 set of numbers from the convolutional image and reduce it to one number by taking the average, max, or min of the numbers four numbers, thus reducing the image size by a factor of four. An activation layer is like the conventional network where the weighted inputs come into a unit function as input to an activation function (e.g., sigmoid function or rectified linear unit), which then outputs a single number. Non-linear activation units allow the network to learn non-linear concepts.

Several researchers have proven the success of CNNs in image classification problems. Application areas in addition to image classification include natural language processing, self-driving cars, text classification, and drug discovery. This study used a CNN architecture with five convolutional layers and an adaptive pooling layer to manage variable input sizes.

**Long Short Term Memory Networks (LTSM)**. An LTSM [10] is a particular type of recurrent neural network (RNN). RNNs have persistent memory units to allow units to remember earlier information sequences, such as text, speech, genomes, and the stock market. For instance, when classifying a word's meaning, one should consider the surrounding words of the sentence. LSTMs are a type of RNN that retains long-term information processed earlier in a sequence and has proven successful in various problems, such as speech recognition, image captioning, and language modeling.

**ResNet50.** Pre-trained networks are readily available that can recognize a thousand different object categories with high accuracy. We pick a network, ResNet50 [11], which is pretrained on more than a million images from ImageNet, a commonly used image database. ResNets50 is a CNN with fifty hidden layers that have learned feature representations from images in its lower layers.

**Transfer Learning.** For transferring knowledge, this study leveraged previously trained networks on related domains to warm start the learning process for each algorithm. When transferring, we investigate: (1) freezing the lower weights of the transferred network and only allowing the last interconnected layer to change, and (2) allowing all weights in the network to change. Freezing is a common mechanism for transferring knowledge to maintain the model's integrity while allowing minor changes at the final network stages to avoid overfitting small datasets.

**Input Representations.** We evaluated two ways to represent code for a learning algorithm: (1) Instruction Counts and (2) Digital Images. The Instruction Counts representation provides a count for the number of times an instruction occurs in a code segment's Control Flow Graph (CFG). Analogous to using word counts to classify a text document, the vector of instruction counts provides information as input to a learning algorithm. Rather than give the networks the number of times an instruction occurs, we convert the vector into two forms:

1. Term-Frequency/Inverse-Document-Frequency (TF-IDF) popular in text classification.
2. A Doc2Vec embedding found in PyTorch, which we set to train for thirty epochs, gave a vector embedding size of three hundred, and chose the *distributed-bag-of-word* setting since instruction counts are agnostic to order.

We use the angr binary analysis platform for Python to obtain individual instructions in binaries. Figure 1 outlines the process of creating a vector. Specifically, angr generates a graph representation for each binary. The graph generation can be computationally expensive and fail on a binary, although it does succeed on most of them. The graph nodes are functions, and the edges between the nodes are calls to and from the respective functions. Within the functions are a group of blocks composed of numbers stored in a dictionary where each number corresponds to an instruction. Analogous to document classification, this dictionary is equivalent to a document, and the individual instruction counts are equivalent to words. Like document classification, we then use TF-IDF and Doc2Vec to postprocess the instruction counts.

A picture containing diagram

Description automatically generated

**Figure 1.** Process of generating vectors from binaries in conjunction with Task 2.2.

(Image from Veronika Strnadová-Neeley’s August, 2021 report for this contract.)

The study’s next representation converts binaries into digital images and classifies the binaries by recognizing patterns in the images. Images can capture small patterns while retaining a global structure. Previous research has demonstrated the efficacy of this representation and can offer a better resilience against obfuscation, particularly packing.

The study created images by mapping the binary’s first byte to pixel[0,0], the next to pixel[0,1], and so forth (see Figure 1). Since code comes in varying lengths, we determine the image's width by the file's size divided by 1024 (as shown in the following table). For instance, a binary’s divided size between 100 and 200 will have a pixel width of 256. We append 0s to reach the rectangular size if the code bytes run out.

|  |  |
| --- | --- |
| **Size Range (210 bytes)** | **Pixel Width** |
| <11 | 1 |
| 11-30 | 32 |
| 31-60 | 64 |
| 61-100 | 128 |
| 101-200 | 256 |
| 201-500 | 384 |
| 501-1000 | 512 |
| 1001 | 768 |
| >1001 | 1024 |

We then use PyTorch to resize (i.e., stretch) each source image to match the image size required by pre-trained networks (ResNet50 in this case). We then normalized this image according to the specifications required by ResNet50 and other pretrained networks within Torchvision. We then duplicate the image into three bands to match the number of inputs ResNet50 requires.

Diagram

Description automatically generated

**Figure 1.** Converting binary code into a grayscale image [],

**Datasets.** We created two datasets for this document’s investigation:

1. A malware dataset from the Practical Security Analytics website ([www.practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/](http://www.practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/)). For each malicious code, the database gives the antivirus software that flagged this code as being malicious. For our transfer tests, we set aside 135 code samples that managed to evade detection from all antivirus software. For creating background knowledge, we trained a network using 4,000 malicious code samples detected by at least one antivirus software package and 4,000 benign code samples. Our tests determine if we can use prior systems to create knowledge that we can quickly fine-tune to new threats as they emerge.
2. We created a vulnerability dataset for Common Weakness Enumerations (CWEs). CWEs are a list of software (and hardware) weakness types that provide a measuring stick for vulnerability identification, mitigation, and prevention efforts. Community-led efforts try to create a common baseline standard that scores each CWE based on how easy that weakness type is to find and exploit. Vulnerable CWEs allow adversaries to take over systems, steal data, or prevent applications from working. The main goal of CWE is to eliminate the most common mistakes before releasing a product. The following table lists the CWEs we compiled from code obtained from the NIST SARD dataset (<https://samate.nist.gov/SARD/>). The number of examples differs between the two representations evaluated since our Instruction Count algorithm did not process all code.

|  |  |  |  |
| --- | --- | --- | --- |
| **CWE #** | **CWE Description** | **Number of**  **Examples**  **(Images)** | **Number of Examples**  **(Instruction Counts)** |
| 78 | Improper Neutralization of Special Elements used in an OS Command (OS Command Injection | 2057 | 1570 |
| 121 | Stack-based Buffer Overflow in C | 453 | 453 |
| 124 | Buffer Underwrite in C | 280 | 280 |
| 126 | Buffer Overread in C | 168 | 168 |
| 127 | Buffer Underread in C | 280 | 278 |
| 134 | Use of Externally-Controlled Format String | 185 | 111 |
| 190 | Integer Overflow or Wraparound | 75 | 74 |
| 479 | Signal Handler Use of a Non-reentrant Function | 18 | 18 |
| 680 | Integer Overflow to Buffer Overflow | 144 | 144 |
| 758 | Reliance on Undefined, Unspecified, or Implementation-Defined Behavior | 234 | 234 |
| 761 | Free of Pointer not at Start of Buffer | 276 | 207 |

We generate this model by training the matching model (FNN, CNN, or LSTM) on 4,000 benign and 4,000 malicious binaries. At least one antivirus software had detected each malicious file.

**Results**

We conducted hundreds of ten-fold cross-validations. Each cross-validation splits the training set into ten equal sets. The learner trains on nine of the sets and tests generalization accuracy on the held-out set. We repeat this ten times, with each set left out once and only once. Accuracy comes from the sum of the ten test sets. Given the positives and negatives in the sets are evenly split, we find reporting overall accuracy sufficient to convey efficacy for the purpose of this report.

Every dataset has an equal number of positive and negative examples. As such, reporting overall accuracy sufficiently conveys efficacy for our purposes. We evaluated FNNs, CNNs, and LSTM with the Instruction Counts representation and FNNs, CNNs, and ResNet50 with the Digital Image representation. We compared freezing some weights for each transfer learning algorithm to letting them all change. The frozen weights are all network weights except the last layer, which training fine tunes to fit the new dataset. The idea behind freezing some weights is to keep the bulk of the transferred knowledge intact while allowing the network to alter how it uses this knowledge.

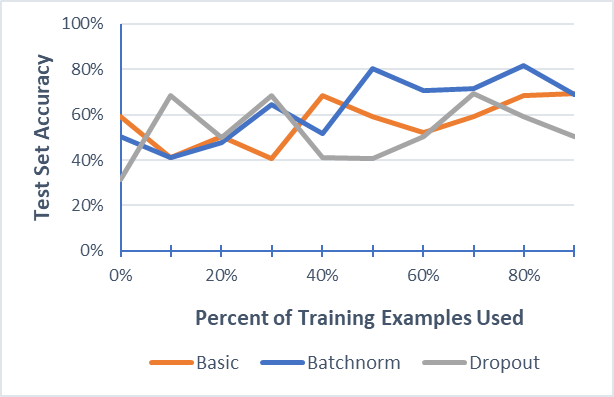
The research investigated augmenting the training procedures with Batchnorm [12]and Dropout [13]. Batchnorm is a common technique that normalizes the batch of data that comes in for a layer by computing the mean and standard deviation for that layer and then rescales the data for that layer with the intent to provide a better, normalized training Dropout is a common technique that tries to reduce the complexity of the network and thus avoid overfitting by dropping weights and nodes during training. Conventional wisdom dictates not using Batchnorm with LSTM networks and not using Dropout for CNNs (including ResNet50). Our results confirm this; thus, we do not present results for those combinations here.

**Instruction Count Representation**

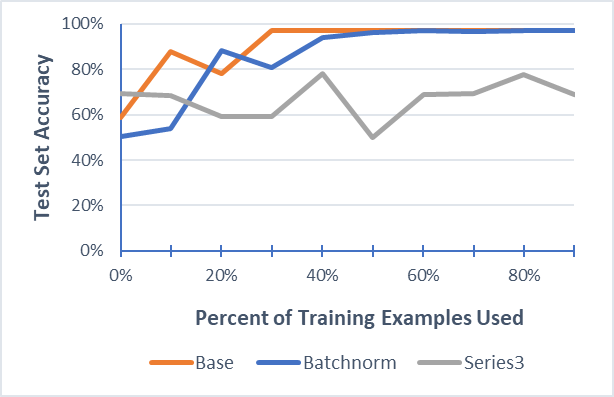
***CWE Detection****.* The results for using our instruction count representations for the CWEs did not produce good results. Most were stuck near a random guess of about 50% correct. The raw results are presented in the accompanying spreadsheet.

***New Malware Detection.*** Using both the TF-IDF and Doc2Vec representations, we obtained promising results for predicting the evasive malware that none of the antivirus software could detect.

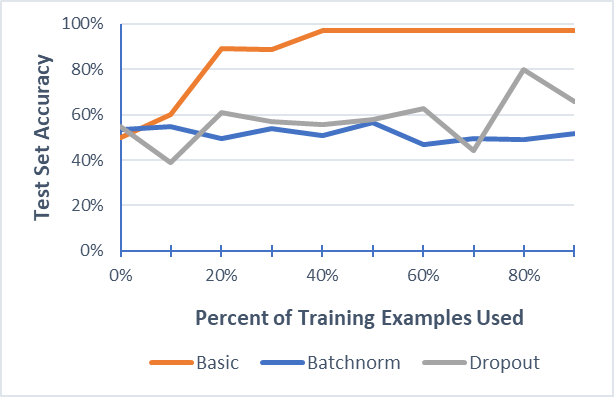
*TF-IDF Representation.*For non-transfer networks, training the ANNs, CNNs, and LTSMs without using either batchnorm or dropout worked best and are the results we present here.



**Graph 1. Training methods on ANNs (no transfer of knowledge).**

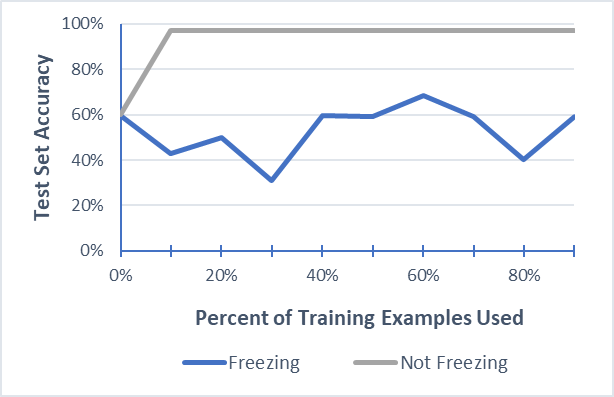


**Graph 2. Training methods on ANNs (no transfer of knowledge).**

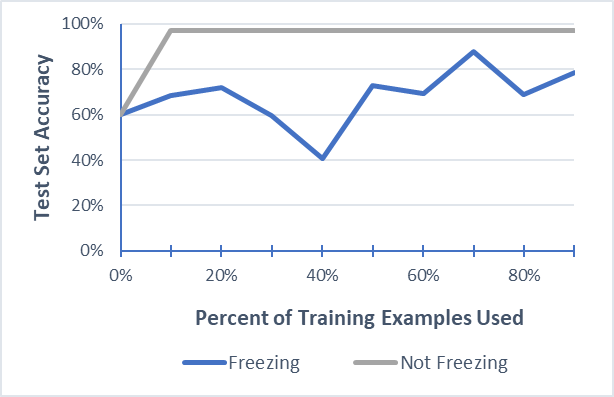
****

**Graph 3. Training methods on ANNs (no transfer of knowledge).**

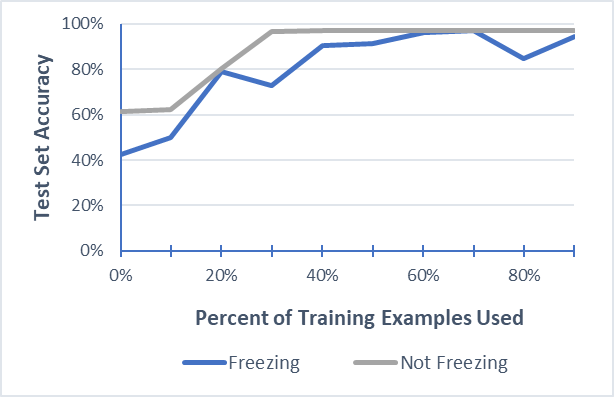
We generally obtained the best results by training without dropout or batchnorm. The next three graphs show not freezing the lower levels of weights in the networks works better than freezing them.



**Graph 4. Not freezing weights versus freezing some weights during transfer in ANNs.**

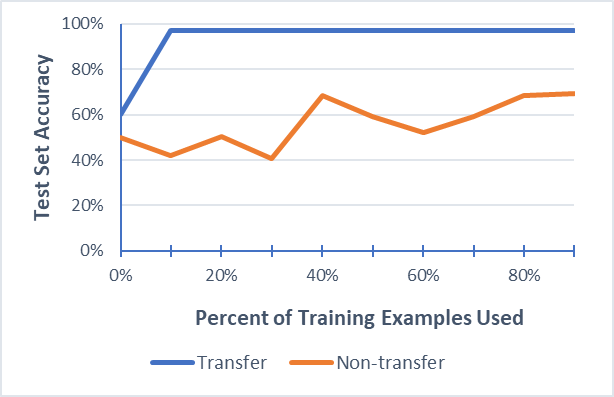


**Graph 5. Not freezing weights versus freezing some weights during transfer in CNNs.**

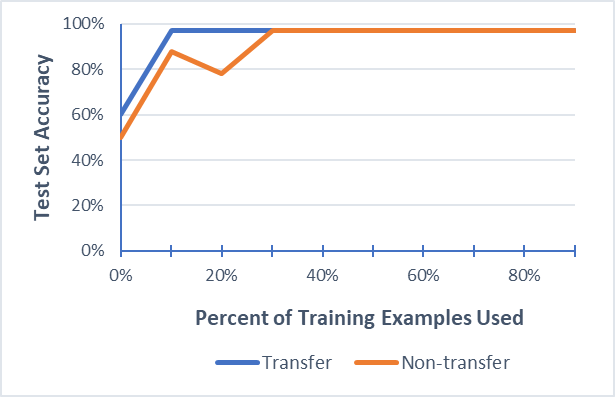


**Graph 6. Not freezing weights versus freezing some weights during transfer in LSTMs.**

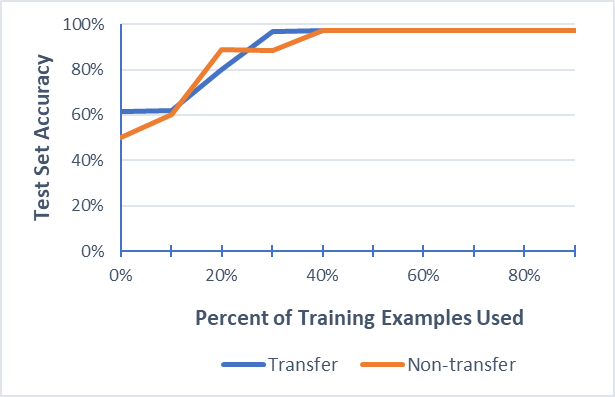
Graphs 7-9 show that starting with the network trained on previously known malware works better than starting without knowledge transfer. Graph 10 compares the best transferred network with the best non-transfer network.



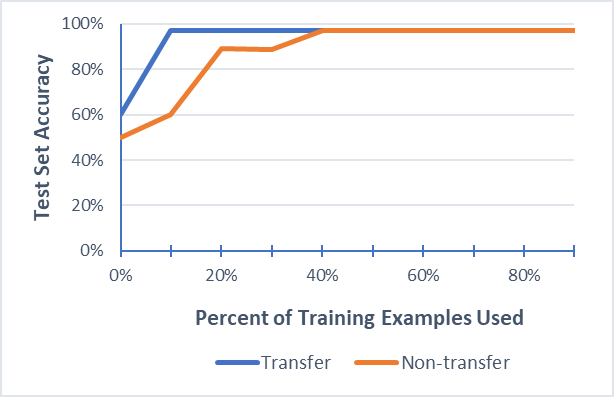
**Graph 7. TL ANN versus non-TL ANNs.**

****

**Graph 8. TL CNN versus non-TL CNNs.**

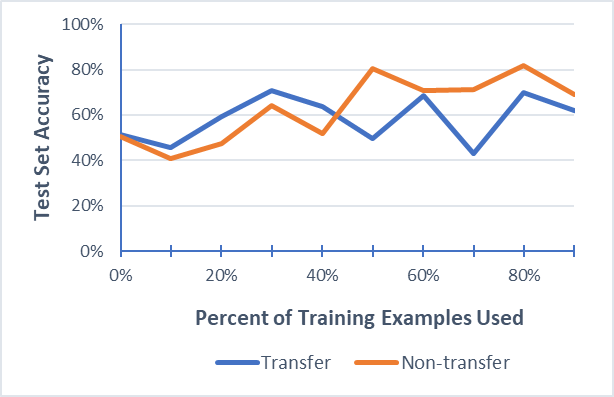
****

**Graph 9. TL LSTM versus non-TL LSTMs.**



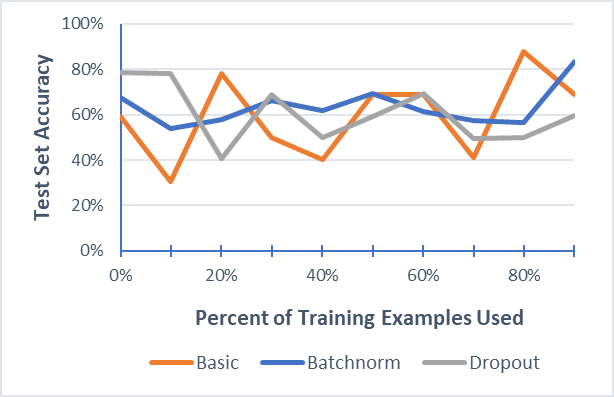
**Graph 10. Best TL network (ANN, CNN without freezing) versus best non-TL network (LSTM).**

Graph 11 shows negative transfer can also happen when the knowledge or training technique isn’t conducive.

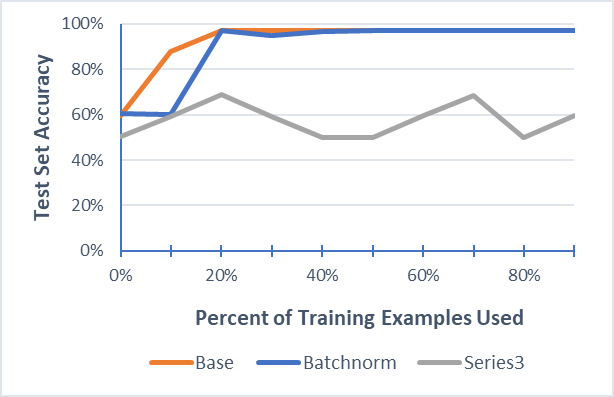


**Graph 11. Example of Negative Transfer. In this case, training ANNs with Batchnorm while freezing weights for the TL network.**

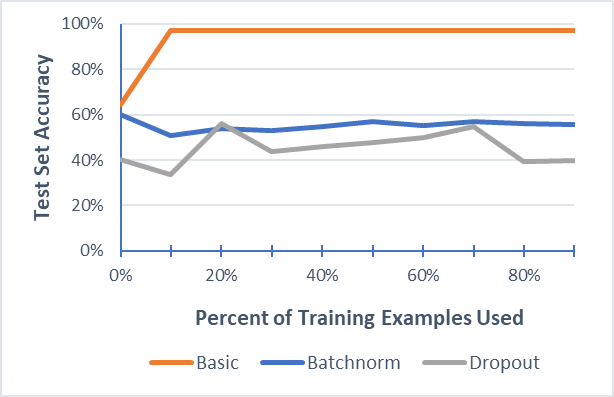
**D2V Rep.** As with our non-transfer networks, training CNNs and LTSMs without using either batchnorm or dropout worked the best. Training ANNs with or without batchnorm also worked well, but not as good as the CNNs or LTSMs.

****

**Graph 12. Training methods on ANNs (no transfer of knowledge).**

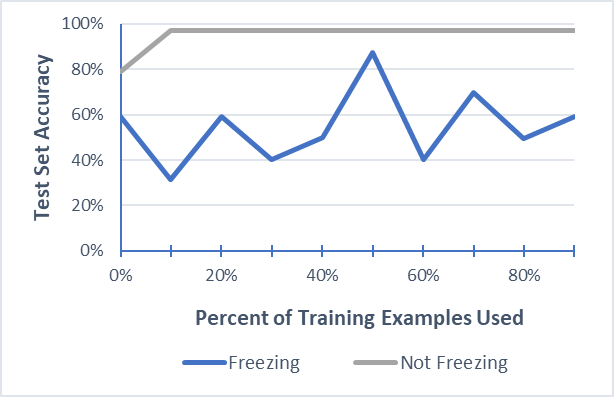
****

**Graph 13. Training methods on CNNs (no transfer of knowledge).**

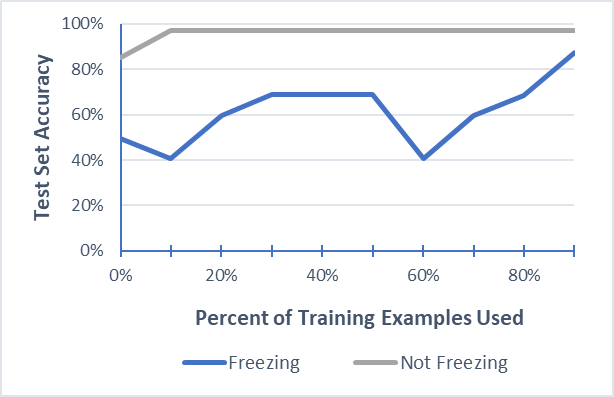
****

**Graph 14. Training methods on LSTMs (no transfer of knowledge).**

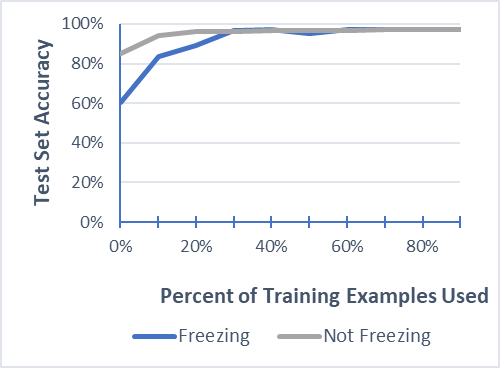
With TL networks, it was better to not freeze than freeze with ANNs. CNNs did not obtain good results either with or without freezing. three graphs compare freezing weight versus not freezing the lower weights (when training without batchnorm or dropout).

****

**Graph 15. Not freezing weights versus freezing some weights during transfer in ANNs.**

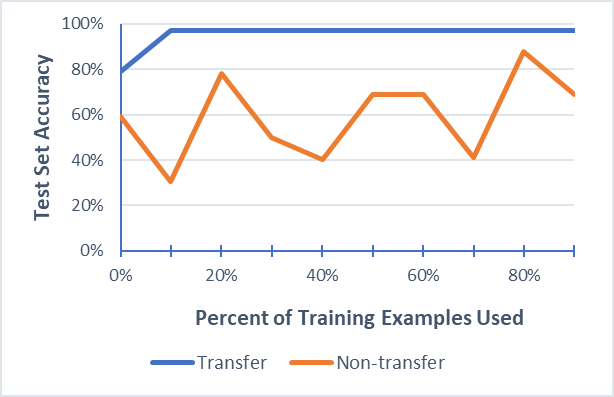


**Graph 16. Not freezing weights versus freezing some weights during transfer in CNNs.**

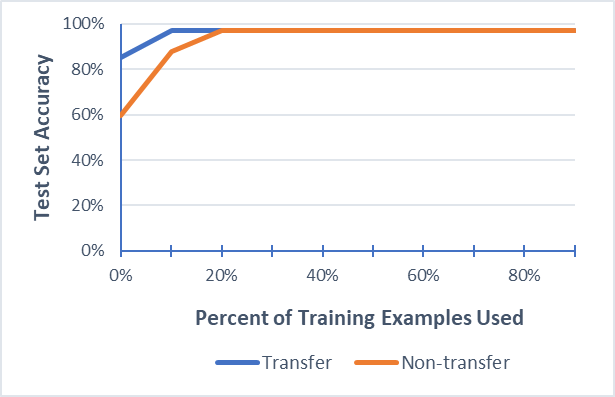


**Graph 17. Not freezing weights versus freezing some weights during transfer in LSTMs.**

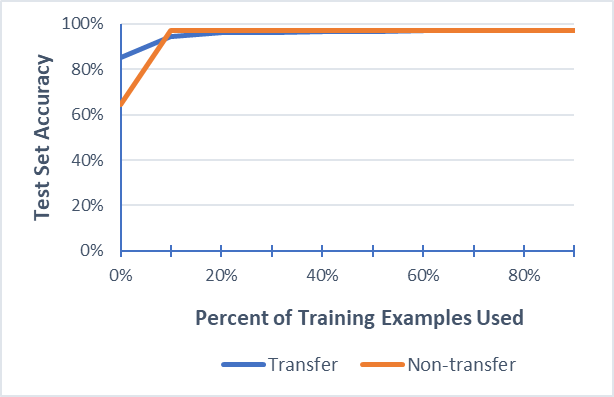
Starting with the network trained on previously known malware generally works better than starting without knowledge transfer.

****

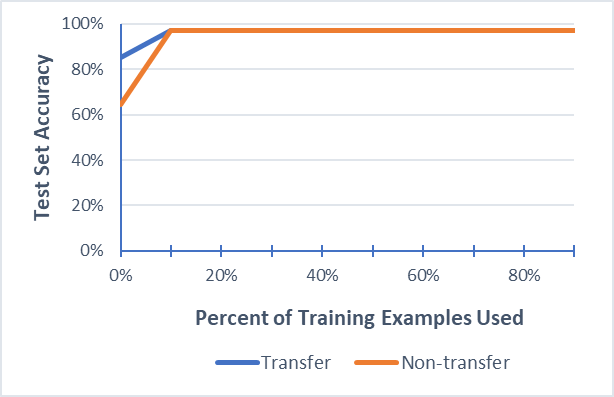
**Graph 18. TL ANN versus non-TL ANNs.**

****

**Graph 19. TL CNN versus non-TL CNNs.**

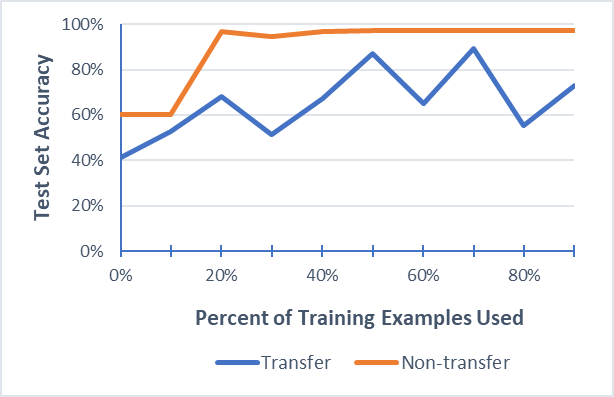
****

**Graph 20. TL LSTMs versus non-TL LSTMs.**



**Graph 21. Best TL network (ANN without freezing) versus best non-TL network (LSTM).**

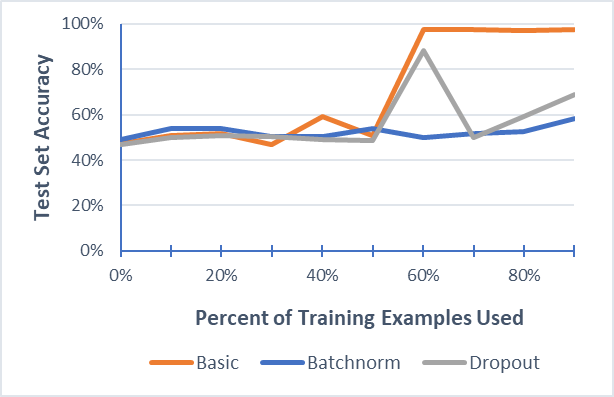
As before, negative transfer can happen when the knowledge or training technique isn’t conducive.

****

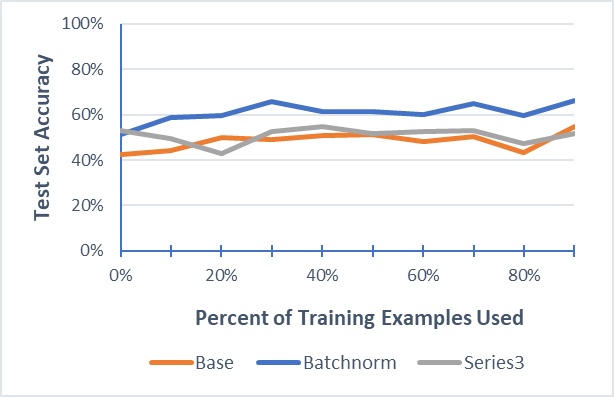
**Graph 22. Example of Negative Transfer (CNNs trained with Batchnorm; some weights frozen).**

**Image Representation**

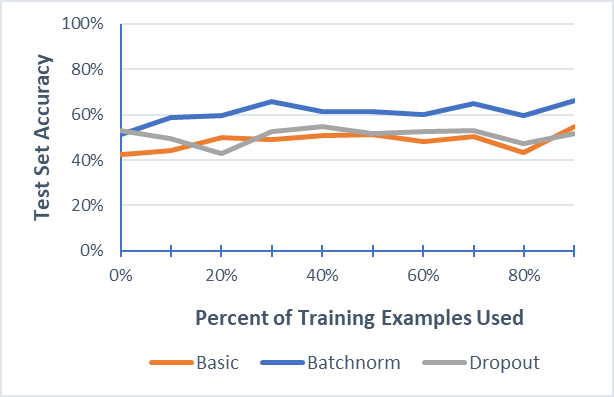
***New Malware Detection.*** When using image representations and training non-transfer networks, training the ANNs without batch norm or dropout worked best, while training CNNs and Resnet with Batchnorm worked best.

****

**Graph 23. Training methods on ANNs (no transfer of knowledge).**

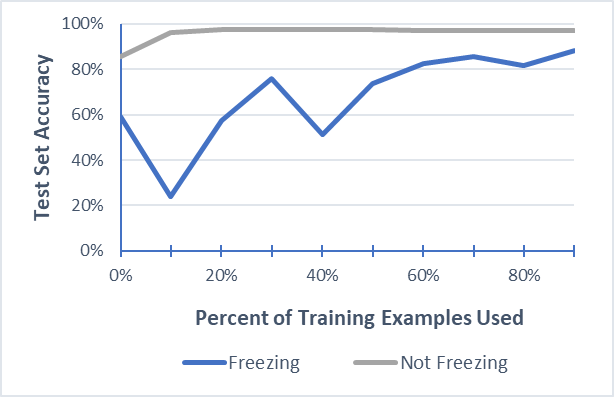
****

**Graph 24. Training methods on CNNs (no transfer of knowledge).**

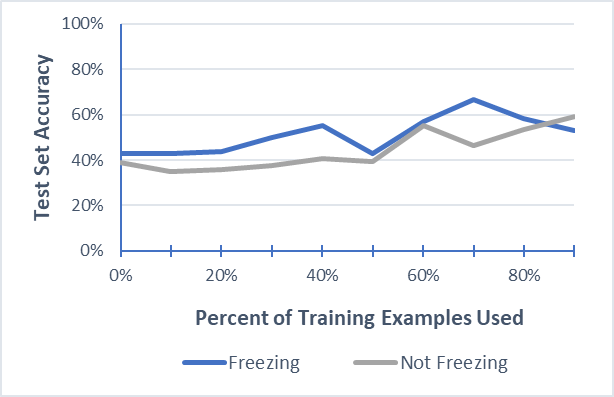
****

**Graph 25. Training methods on ResNet (no transfer of knowledge).**

With TL network, it was better to not freeze than freeze with ANNs. CNNs did not obtain good results either with or without freezing. The ResNet did better than CNNs, but not as good as ANNS.

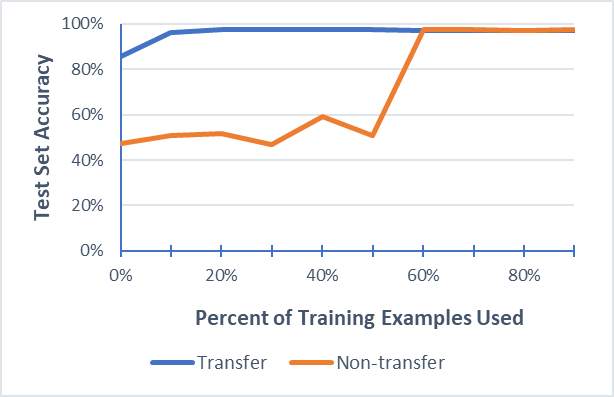
****

**Graph 26. Not freezing weights versus freezing some weights during transfer in ANNs (no Batchnorm or Dropout).**

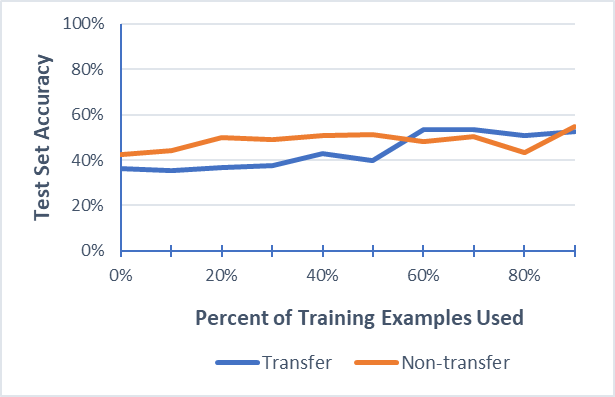
****

**Graph 27. Not freezing weights versus freezing some weights during transfer in CNNs (with Batchnorm).**

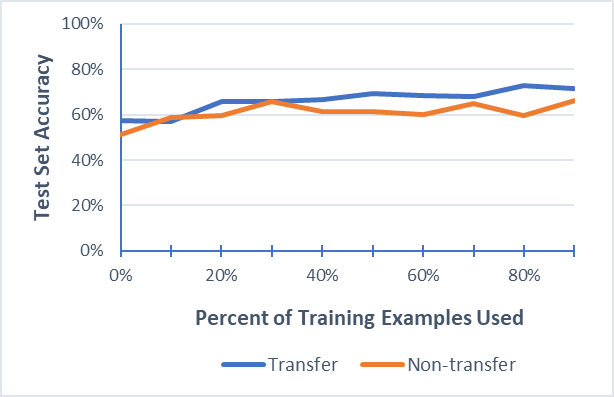
Starting with the network trained on previously known malware works better than starting without knowledge transfer.



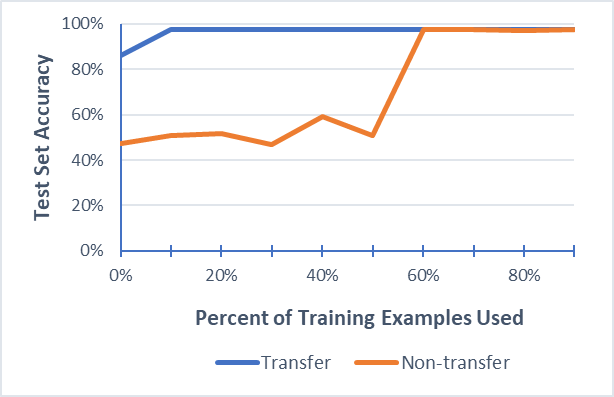
**Graph 28. TL ANN (no frozen weights) versus non-TL ANN. Neither used Batchnorm or Dropout.**



**Graph 29. TL CNN (frozen weights and with Batchnorm) versus non-TL CNN (with Batchnorm).**

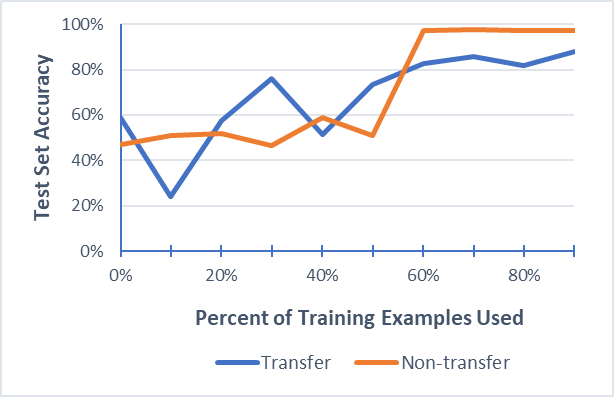


**Graph 30. TL ResNet versus non-TL ResNet (both with Batchnorm; TL has frozen weights).**



**Graph 31. Best TL network (ANN with Dropout and without freezing) versus best non-TL network (ANN without using Batchnorm or Dropout).**

Like above, negative transfer can also happen when the knowledge or training technique isn’t conducive.



**Graph 32. Example of Negative Transfer. In this case, ANNs with base training while freezing many transferred weights.**

**Image Representations with CWEs**

This section presents results for using the image representation to test the transfer of knowledge from the malware dataset to learning various CWEs. Similarly, freezing weights with ANNs and CNNs resulted in largely random guessing. Thus, these results are omitted in this section.

**Table 1. TL networks seeded with Malware network versus those initialized without.**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CWE** | **78** | **121** | **124** | **126** | **127** | **134** | **190** | **476** | **680** | **758** | **761** |
| **ANN-Transfer** | 100% | 50% | 50% | 50% | 50% | 50% | 94% | 92% | 50% | 96% | 50% |
| **ANN-Blank** | 89% | 80% | 78% | 68% | 80% | 68% | 62% | 67% | 99% | 68% | 61% |
| **CNN-Transfer** | 100% | 99% | 100% | 98% | 99% | 98% | 93% | 91% | 75% | 97& | 99% |
| **CNN-Blank** | 100% | 99% | 99% | 99% | 95% | 88% | 69% | 91% | 69% | 97% | 97% |
| **Resnet-Transfer** | 99% | 99% | 98% | 92% | 97% | 96% | 84% | 86% | 96% | 97% | 96% |
| **Resnet-Blank** | 98% | 99% | 99% | 99% | 99% | 88% | 69% | 62% | 99% | 99% | 97% |

**Table 2. TL networks seeded with Malware network versus those initialized without using only 10% of the training data. This indicates how well TL works with few training examples**.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CWE** | **78** | **121** | **124** | **126** | **127** | **134** | **190** | **476** | **680** | **758** | **761** |
| **ANN-Transfer** | 92% | 50% | 50% | 50% | 50% | 50% | 93% | 90% | 50% | 90% | 50% |
| **ANN-Blank** | 58% | 52% | 51% | 57% | 57% | 50% | 51% | 57% | 62% | 60% | 50% |
| **CNN-Transfer** | 60% | 55% | 71% | 73% | 66% | 58% | 63% | 68% | 55% | 60% | 60% |
| **CNN-Blank** | 53% | 51% | 49% | 62% | 59% | 52% | 52% | 57% | 48% | 49% | 46% |
| **Resnet-Transfer** | 60% | 60% | 50% | 62% | 66% | 58% | 60% | 49% | 55% | 66% | 57% |
| **Resnet-Blank** | 52% | 51% | 495 | 62% | 59% | 52% | 52% | 57% | 61% | 49% | 46% |

**Results Analysis**

A summary of the results is as follows:

* *It is ineffective to use either Instruction Counts representation to detect CWEs*. The investigation used the 8,000 malware/benign examples network as background knowledge (as did all our transfer experiments). The study investigated if we could use our knowledge for detecting malware to kickstart our ability to learn to detect CWEs. Unfortunately, the Instruction Count representation produced transferred and non-transferred results that hovered around random guesses for all learning approaches.
* *Refining models of known malware can detect new and evasive malware quicker using both Instruction Counts representations.*
  + Transferred knowledge for TL networks is a network generated from the 8,000 malware/benign examples detected by an antivirus software.
  + For the TF-IDF representation of Instruction Counts:
    - Not using Batchnorm or Dropout worked best for all networks, with transfer and without.
    - It worked best not to freeze the weights for all transfer networks and let them adjust during training.
    - When comparing the corresponding networks (e.g., transfer FNNs with non-transfer FNNs), the transfer networks could detect new malware with fewer examples than those starting from nothing.
    - If we trained the networks sub-optimally (e.g., transfer FNNs with freezing and Dropout), negative transfer occurs, and the non-transfer FNNS work better. Negative transfer is when TL results in decreased performance.
  + For the Doc2Vec representation of Instruction Counts:
    - Overall, the Doc2Vec representation tended to produce slightly better accuracy than the TF-IDF representation.
    - Non-TL networks: (1) were more accurate without using Batchnorm or Dropout, and (2) the CNNs and LSTMs produced more accurate results than the FNNs.
    - TL networks had: (1) more accurate results without freezing weights, (2) reduced accuracy when using either Batchnorm or Dropout, and (3) produced more accurate results with FNNs than CNNs or LSTMs.
    - Appropriately trained TL networks produced more accurate results than corresponding appropriately trained non-TL networks.
    - Negative transfer occurred when training techniques are not conducive to the network.
* *Refining models of known malware can detect CWEs using the Digital Image representation and learn faster by transferring knowledge*.
  + The transferred knowledge for TL networks is a network generated from the 8,000 malware/benign examples detected by an antivirus software.
  + Both the TL and non-TL networks showed the ability to learn eleven or the twelve CWE codes, failing only with CWE 479, which had only eighteen examples.
  + Neither the TL nor non-TL networks produced satisfactory results with Dropout.
  + For FNNs, TL was either much more accurate than regular FNNs or much less accurate (meaning, either a large transfer or a significant negative transfer occurred).
  + With TL, the CNNs and ResNet50 worked better than FNNs.
  + With few examples available, TL with CNNs and ResNet50 produced more accurate results than their corresponding non-TL networks. The TL and non-TL networks produced similar asymptotes when more data became available.
  + Unfit TL resulted in substantial negative transfer with suboptimal settings.
* *Refining models of known malware can detect CWEs using the Digital Image representation and learn faster by transferring knowledge*.
  + Transferred knowledge for TL is a network generated from the 8,000 malware/benign examples that the antivirus approach detected.
  + For non-TL networks: (1) FNNS produced more accurate results without either Batchnorm or Dropout, and (2) CNNS and ResNet50 worked best with including Batchnorm.
  + With TL, FNNs produced better accuracy than ResNet50, which was more accurate than CNNs.
  + TL networks did better on average than non-TL networks, particularly with small datasets.
  + Once again, negative transfer occurs if training is not conducive to the TL network.

The experiments demonstrated the power of TL in detecting malware and code weaknesses and vulnerabilities across multiple domains and learning techniques. The results also showed that TL must carefully be applied, or it can negatively influence learning accuracy.

**Suggested Future Work**

The initial efforts for the project determined the gaps in technology and created the datasets. The latter efforts involved hundreds of cross-validation tests in conducting an initial survey of learning algorithms and hyperparameter settings, then additional cross-validation tests to produce the results presented in the report. These efforts demonstrated:

* TL can increase the accuracy of detecting malware and code weaknesses, sometimes dramatically when only a few examples exist.
* One must be careful when applying TL as negative transfer is a distinct possibility.
* Multiple effective ways to represent binary code for supervised learning exist, and TL works across these different representations.
* Multiple ML algorithms can learn threats in binary code, and TL works across these different learning approaches.
* Knowledge transferred may come from many diverse sources, sometimes from problem domains that appear unrelated on the surface (such as ResNet50 for image classification).

While these tests demonstrate the potential for TL in cybersecurity, they only scratch the surface. Continued recommended research includes:

* *Improved datasets*. Data is critical for supervised learning (hence, the usefulness of TL). This study used 8,000 code samples to generate background knowledge. The larger the dataset, the more effective the background knowledge can become. For deep learning, breakthroughs happen when there are millions of data points. The same could happen with code analysis and a community effort to create such a dataset. At that point, TL will become essential.
* *Additional ways of transferring knowledge.* There are many ways to transfer knowledge. The source task can transfer knowledge from two places:

1. *Source input.* Raw data in the form of ground truth data, the data space, and data distribution.
2. *Source model.* Knowledge in the form of learned models, feature data generated by the model, feature data spaces, and feature data distributions.

Methods for transferring *from the source input* include:

1. *Transferring to Target Feature Mapping*. These methods focus on learning a feature extractor adapted to the source task that fits the target task.
2. *Transferring to Target Model*. These include domain adaptation techniques for supervised and unsupervised learning, where the algorithms assume the source and target data share the same input space but have different data distributions.
3. *Transferring to Target Predictor*. These methods generally reweight or select a part of the source data and combine it with target data to generate a combined feature mapping. These include boosting-based methods, Discriminative TL methods, and Source Domain Selection.

Methods for transferring from source models include:

1. *Transferring the Source Feature Mapping*. These techniques reuse the pre-learned feature-extractor portion of the model for the new target tasks. These include TL methods for supervised learning (e.g., fine-tuning deep-learning methods, lifelong learning, and metric learning) and unsupervised learning (e.g., transferring unsupervised deep-learning methods, deep-belief networks, auto-encoders, generative adversarial networks, and contrastive learning).
2. *Transferring the Source Predictor*. One can transfer just the last part of the source model, the predictor parameters. These techniques include discriminative models (e.g., Support Vector Machines) and generative models (e.g., Bayesian models).
3. *Transferring the Source Model*. These approaches translate the whole model. Methods include knowledge distillation for deep networks, lifelong/reinforcement learning, and meta-learning (e.g., learn optimizers, learn hyperparameter settings, learn parameter changes in deep networks, or learn how and what to transfer from source networks).

* *Different learning techniques.* We provided learning approaches for FNNs, CNNs, and LSTMs. There are numerous other learning approaches and ways of training these different approaches.
* *Different input representations.* How you represent the problem for supervised learning is of critical importance. Future work should investigate other ways of presenting binary code to a learner.

For TL to be effective, researchers need to be able to share models efficiently. Unfortunately, merely placing them in a repository like GitHub has proven ineffective because TL has burdens like:

* *Intricacies*. TL requires expertise in knowing when, where, and how to transfer knowledge.
* *Cross-domain obstacles.* Must transfer across data distributions and feature representations.
* *Disjointed workflows.* There are several ML packages in use with a variety of features.
* *Model complexity.* ML models are often black boxes that are uninterpretable by humans.
* *Formatting issues.* Many popular ML packages with differing model formats make sharing difficult.
* *Lack of Coordination.*Information exchange requires structure, awareness, and leadership.

We propose to address these hurdles by building a commercial web system in a central location (perhaps a secure location) that provides the following:

* *Knowledge-Transfer Station.* TL methods with instructions for when, where, and how to transfer.
* *Knowledge Repository*. Intuitive tools to store and retrieve models, data, and instructions.
* *Learning Station*. Full ML capabilities that seamlessly integrate with all parts of the software,
* *Crowdsource Station*. Tools to set competitions, projects, and goals to promote community action.
* *End-to-end workflows*. The entire process should be united, from obtaining a pretrained model to retraining it.
* *Interoperability.* Ability to read/write models from popular ML packages to facilitate sharing.
* *Education Center.* Support for information exchange and awareness, R&D coordination for projects, user forums, video training, portals for open source, FAQs, and help pages.

# **References**

|  |  |
| --- | --- |
| [1] | K. SHaukat, S. Luo, V. Varadharajan, I. Hameed and M. Xu, "A Survery of Machine Learning Techniques for Cyber Security in the Last Decade," 23 Dec 2020. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9277523. [Accessed 2020]. |
| [2] | Y. Lecun, Y. Bengio and G. Hinton, "Deep Learning," *Nature,* vol. 521, no. 7553, pp. 436-444, 2015. |
| [3] | S. Pan and Q. Yang, "A Survey of Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering,* vol. 22, no. 10, pp. 1717-1724, 2014. |
| [4] | L. Torrey and J. Shavlik, "Transfer Learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, IGI Global, 2009, pp. 242-264. |
| [5] | A. Ng, "The Nuts and Bolts of Machine Learning," in *Neural Information Processing Systems*, 2016. |
| [6] | R. Nakod, "Five Deep Learning Trends that will Rule 2019. eInfoChips," 2019. [Online]. Available: www.einfochips.com/blog/5-deep-learning-trends-that-will-rule-2019. |
| [7] | D. Olaosabikan, "Artificial Intelligence Predictions for the year 2019," Dataconomy, 21 Dec 2018. [Online]. Available: https://dataconomy.com/2018/12/artificial-intelligence-predictions-for-the-year-2019. |
| [8] | K. Weiss, T. Khoshgoftaar and D. Wang, "A Survey of Transfer Learning," *Jornal of Big Data,* vol. 3, no. 1, 2016. |
| [9] | F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu and Z. Zhu, "A Comprehensive Survey on Transfer Learning," arXiv:1911.02685, 2019. |
| [10] | S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997. |
| [11] | K. He, X. R. S. Zhang and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. |
| [12] | S. Ioffe and C. Sededy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv,* vol. 1502.03167, 2015. |
| [13] | N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR),* vol. 15, no. 56, pp. 1929-1958, 2014. |