THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ENGINEERING SCIENCE AND MECHANICS

# IMPROVING TRUST IN DEEP LEARNING SYSTEMS BY AUGMENTED DEEP K-NEAREST NEIGHBORS

SARAH TICKNER

SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Engineering Science
with honors in Engineering Science

Reviewed and approved* by the following:

Bruce Einfalt
Senior Research Engineer, Applied Research Laboratory
Thesis Supervisor

Gary L. Gray
Associate Professor of Engineering Science and Mechanics
Honors Adviser

Judith A. Todd
Department Head
P.B. Breneman Chair and Professor of Engineering Science and Mechanics

* Electronic approvals are on file in the Schreyer Honors College and Engineering Science and Mechanics Office.

# ABSTRACT

Deep Learning is a black box process in which the input and the output are visible but the inner workings of how an input is processed are difficult to trace. Although automated classification can increase the efficiency of many processes, the outputs of machine learning algorithms are sometimes incorrect, and the lack of visibility into the processing done by these algorithms makes it difficult to assure operators that the classifications generated by these algorithms should be trusted. Many machine learning systems have a significant impact on important functions in society, so it is important that results from these systems are trustworthy. It is important to find ways to increase the assurance that Neural Networks can provide regarding the correctness of their classifications so that these algorithms can be employed safely and without causing unintended negative side-effects.

This thesis examines Machine Learning and Deep Learning systems and discusses ways that these systems can provide increased assurance that they are operating only in the ways that are intended. It focuses in particular on Deep k-Nearest Neighbors, a method to look at the results of several layers of a deep neural network and gain more confidence that the result being output is correct. This method can be improved upon, and a plan for exploring several potential improvements is developed and presented in this thesis.

# TABLE OF CONTENTS

# LIST OF FIGURES



—Remainder of Page Intentionally Left Blank—

# LIST OF TABLES



—Remainder of Page Intentionally Left Blank—

# ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of many people. I would like to extend a huge thank you to the Applied Research Laboratory, Penn State University for hiring me to explore Machine Learning systems. In particular, I want to thank Andrew Shaffer for recruiting me and giving me support and feedback. Also, I would like to thank my research advisor and supervisor, Bruce Einfalt, for his guidance throughout the process of researching, developing, and writing this thesis. Additionally, I would like to thank Brian Reinhardt for lending his technical knowledge and coding skills to the development of this thesis. My academic advisor, Gary Gray, was a pillar of encouragement and I am grateful for his advice. Finally, I would like to thank my family and friends who have stood by me and been a network of support throughout my undergraduate career.

—Remainder of Page Intentionally Left Blank—

# Introduction

Machine Learning (ML) has the potential to make life safer and easier by enabling people to delegate more tasks to computers. Deep Learning is a specialized branch of machine learning that uses artificial neural networks with many layers to recognize patterns in data and come to conclusions. The potential capabilities of this style of mimicking human thinking are vast, including applications ranging from autonomous vehicles to autonomous security protocols. These systems can affect the economy by increasing productivity and efficiency. As the applications of deep learning systems become increasingly pervasive and the responsibilities delegated to these systems continue to increase, the need for assurance that these systems will consistently make correct choices also increases. However, there is still much that is unknown about how these systems work. Oftentimes, DL systems function as black boxes in which the input and the output are visible but the inner workings of the processes are not easily understood. As a result, these systems can sometimes make decisions that are radically different from what a human operator would typically expect and this can have dire consequences. Autonomous vehicles are a prime example of a machine learning system that can have literal life and death consequences for both the people in the autonomous vehicle as well as others in the surrounding environment–if the system guiding an autonomous vehicle is not consistently making correct choices that ensure the safety of its passengers and surroundings, it should not be allowed on the road.

Previous ML systems have provided assurance that they are generating correct output by reporting confidence measures that reflect the probability that the output is correct. However, this trusting measure is not always accurate as an output with a high confidence score can still be incorrect. In order to develop a better system, a comprehensive dataset is needed for testing,

algorithms that can interpret the data need to be developed, and a meaningful evaluation of those algorithms is necessary. The purpose of this thesis is to develop an approach for increasing the trust that operators have in their neural networks. Specifically, this thesis focuses on convolutional neural networks and on developing an approach to increase assurance that these systems are working as intended by building upon the Deep k-Nearest Neighbors method. With improved trust assurance methods, widespread adoption of these systems will be more likely.

—Remainder of Page Intentionally Left Blank—

# Background

ML is an ever-growing field that aims to utilize past data to draw inferences about future data before it is observed or at the time it is observed. There are a number of ML strategies commonly used to analyze data. It is important to understand the basics before delving into the more complex Deep Neural Networks (DNN). It is also valuable to review the approaches for assuring trust that have already been researched as these provide a foundation that can potentially be built upon.

## Machine Learning Overview

In general, machine learning processes go through the same general series of steps: gathering data, preparing the data, training the system, and testing the system. First, the operator of a ML system must gather data that is representative of their problem. If the data is a different distribution than the real world data that the system will encounter, this can lead to issues of low accuracy in the ML system. This will be discussed in more detail later.

The preparation stage involves ensuring that the data is large enough and diverse enough for the model to learn, and also setting aside a portion of the data for validation. The validation data can be randomly split from the training data or manually chosen to ensure a distribution that is representative of the entire dataset that the system would see in operation. If there is not enough data or the available data does not cover the range of inputs that might be encountered during real world operation of the ML system, the operator can use data augmentation, which alters the available data to increase the size of the training data set. For a set of images this could mean changing the image sizes, flipping the images or applying random noise to the images [1].

This is important because if there is not a variety of data that is representative of real world input data, the system might learn correlations that have no actual basis in the classification. For instance, if a system is classifying cars and the training data is a set of images where the cars are all shown from the driver's side, the system would likely misclassify an input of a car where the image was taken from the passenger's side. Another way to increase data is to count the prediction of new input data as if it were training data. This has the caveat of possibly poisoning the network if the prediction of a data point was incorrect [2]. Additionally, there are three main types of learning within ML that have different requirements. Supervised learning is trained on data that has been labeled and it uses those labels to group the data and map inputs to outputs. Unsupervised learning trains on unlabeled data and it attempts to cluster the data without being explicitly told which data should be grouped together. Reinforcement learning trains by interacting with a dynamic model, receiving feedback in the form of rewards and punishments, and it attempts to optimize to the best possible behavior based off the feedback. There is also semi-supervised learning, which is the combined use of labeled and unlabeled data for training a system. If labels are needed for the ML system being used, it might be necessary for the operator to tag some or all of the training data with its correct classifications.

Training a ML system typically involves defining a function to represent the data, parameterizing it, calculating a loss function, and then minimizing the loss function in order to optimize the system. Testing the ML system is done using the validation data set aside earlier to see if the system is able to categorize it correctly. However, this validation step is not all encompassing, and even if the system performs well on the validation data it may not hold up with new inputs. The validation step could be taken a step farther using cross-validation. One common method is K-Fold Cross-Validation, which is done by randomly splitting a dataset into

k subsets. Then training is done with k minus one subsets and the validation step uses the subset that was not used for training. This is then repeated k times so that all subsets are used for validation once. The specifics of these general steps vary between different ML techniques, but the overall concepts are similar across the techniques.

The concepts of overfit and underfit are key ideas related to the training of ML models. When a model has too many data points and simply memorizes the information that is fed into it, it is considered to be overfit. A ML system that is overfit does not generalize well enough to make accurate predictions with future data and it will likely not perform well with inputs that are different from the training data. On the other hand, underfit occurs when a model does not train enough to make valid predictions about new data. A way to visualize this would be thinking of categorizing animals. An overfit model might be too specific to categorize a husky as a dog because it was not trained on a photo of a husky explicitly, whereas an underfit model would be too broad and might categorize a squirrel as a dog. Basically an overfit model is too specific and an underfit model is too general. It is difficult to find the perfect balance between over and under fitting in a model. This concept is illustrated in Figure 1. With the graph on the left, the dividing line is not fully representing the pattern of the data, but the graph on the right has a dividing line that may misclassify new points because of the orange star outliers residing within the green circle section.

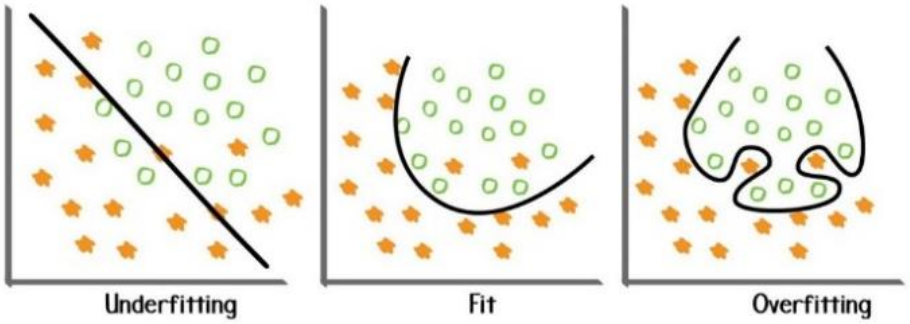


**Figure 1: The Balance Between Underfitting and Overfitting [25]**

Other parameters that are important when evaluating a neural network are bias and variance. Bias is the error caused by the difference between the expected output and the prediction while variance is the error caused by sensitivity to data, or how flexible the system is when shown new data. As the model becomes more complex the bias decreases, but the variance also increases. These two sources of error are inversely proportional to each other, so it is important to minimize them jointly; decreasing one will cause the other to increase. The fine line between underfit/overfit and bias/variance is adjusted by generalizing the model which can be achieved in different ways for the various methods of ML.

## K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised ML algorithm that groups data points into clusters using a distance metric [3]. The "k" is an adjustable hyperparameter that the operator chooses. KNN uses a set of known and labelled data points and maps them to a subspace where the distance between the points can be measured. If the data is not numerical, it is necessary to establish a comparison system so that the distance between points can be compared. When the algorithm evaluates a new point, it looks at the class of the "k" data points closest to it, which is calculated using the distance metric established earlier. The predicted label of the input is assigned using a plurality-based voting system so the class with the most points among the "k" closest points is chosen. If there is a tie, the output is decided by randomly

choosing from among the tied classes. Figure 2 shows an example of KNN; depending upon the chosen "k", 3 or 7, the red input square would be categorized as either a blue dot or green star. This ML method can work effectively when the data is easily grouped or clustered. However, it is less useful when the data has a large number of dimensions that reduce the distance between data points or when the clusters of data points overlap.

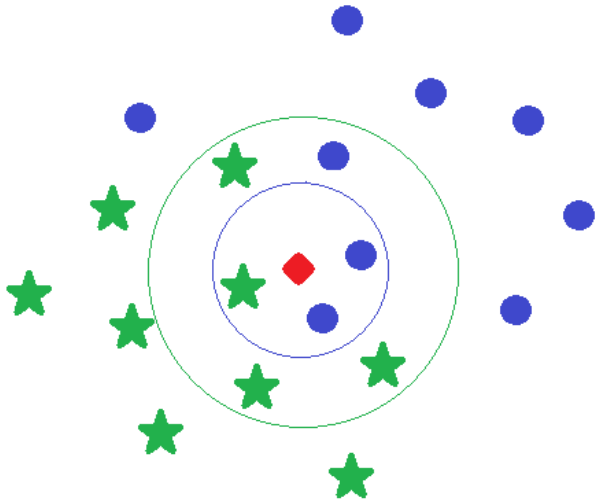

Figure 2: Illustration of K-Nearest Neighbors Algorithm

## Decision Trees

Decision trees are another type of supervised ML that separate data by defining characteristics at each of the branches to arrive at a categorization. The most basic one is a binary decision tree in which each split has only two choices [4]. Decision trees work best when they are shallow, with fewer branches, and where the root node contains the split that is most relevant to the final categorizations. The tree is built by first splitting the data into training and tuning data. The training data dictates the content of each split. It is important to keep the tree from becoming overfit; if the data continues to be split until it is 100% true for the training data, the decision tree may misclassify future data because of an outlier or overlap between data groups. To avoid this, tuning data is used to prune the less relevant splits. The relevance of each split is calculated using splitting measures which determine how much significance each attribute has in the final outcome. Some common splitting measures are Information Gain, Gini, and Entropy

[5]. Pre-pruning is used to decide whether to include a split as the tree is being built, while post-pruning occurs after the tree is built. In addition to pruning, stopping criteria are another method to keep the tree from becoming too overfit. The tree can stop training when it reaches a maximum depth, a maximum number of branches, a minimum number of samples in a node that is allowed to split, or a minimum decrease in relevancy.

Decision trees are typically built using a greedy algorithm, which means that they make the locally optimal choice at each step of their processing but they do not necessarily obtain a globally optimal result. Depending upon the original organization of the branches of the tree, the speed of classification when using a decision tree may vary. Ensemble ML is a strategy to increase the chances of generating correct classifications with decision trees by using a voting mentality. A random forest is a combination of many decision trees which were trained and pruned. The data is split into different sections and each section is used to train a different tree. Diversity between the ensemble members is important, otherwise the behavior of the ensemble would essentially be the same as the behavior of any of its constituent decision trees. The different ways to combine the classifiers are majority voting—either unanimous, simple majority, or plurality—or weighted majority voting. A popular vote or average from the trees in the forest is used to determine the final classification of each input. This approach reduces the likelihood of choosing a poor performing classifier because the forest increases the chances of finding an optimal tree [6].

## Support Vector Machines

A kernel is a class of algorithms for pattern recognition. The most recognized are Support Vector Machines (SVM). SVMs are supervised learning algorithms that are adept at classifying linearly separable data. These algorithms aim to find a hyperplane, or boundary line, by linearly separating the data with the maximum margin, which is the distance between the two closest data points of opposing classes and the hyperplane. Figure 3 shows several examples of hyperplanes and highlights the one with the largest margin. Overfitting can be prevented by using a soft margin which allows some points to be misclassified by the hyperplane. Data is not always linearly separable as the SVM requires; in this case, the kernel trick projects data into a higher dimension so that a linear hyperplane can separate it. Figure 4 shows an example of the kernel trick in action. This approach can be used to enable linear SVMs to solve some non-linear problems.

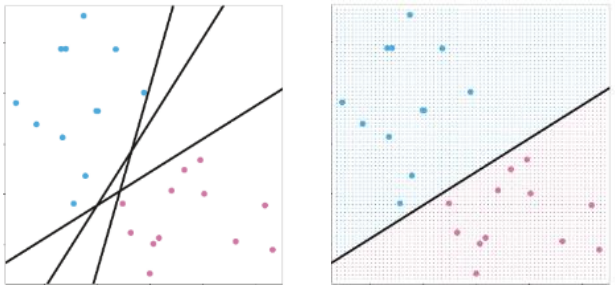

Figure 3: Optimizing a SVM [26]

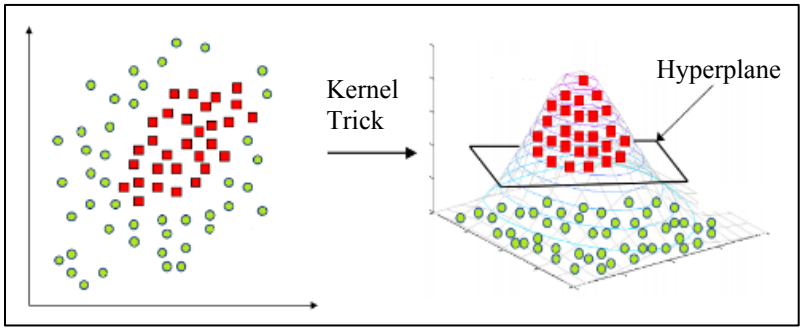


Figure 4: Example of the Kernel Trick [26]

## Perceptron

A perceptron, or artificial neuron, is the mathematical representation of a biological neuron present in human brains. The idea was first introduced in a paper in 1943 which explored the binary "all or none" nature of the neurological system [7]. Neurons in the human brain receive an electrical signal pulse which is modulated and only sent along to connected downstream neurons if it exceeds a threshold value. Similarly, an artificial neuron receives a numerical input which calculates a weighted sum and is passed on according to an activation function, which, in the basic model of a perceptron, is a step function. Figure 5 shows a visual representation of a perceptron. Only linearly separable data can be classified by a perceptron [7]. However, non-linearly separable data can be classified using a multilayer perceptron in which the output from a perceptron is the input to another perceptron and so on, creating layers [28].

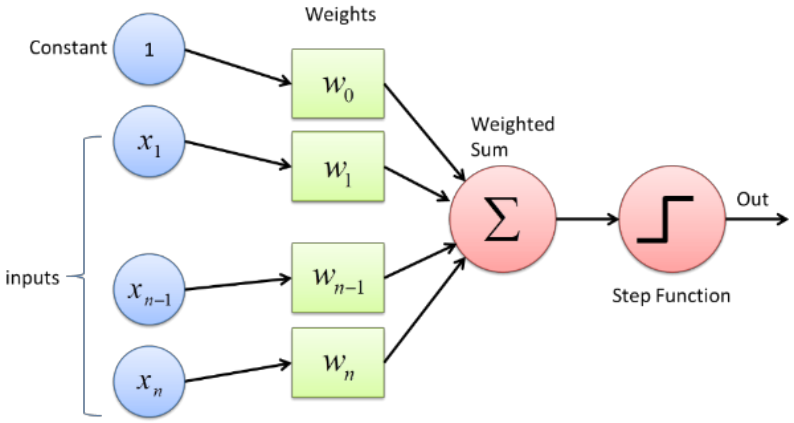


Figure 5: Visual Representation of a Perceptron [27]

## Neural Networks

Neural networks are an approach to machine learning in which computer software simulates the physiological activities involved in human thinking [7]. Many layers of artificial

neurons are arranged to mimic the behavior of real neurons in the human brain to create an Artificial Neural Network (ANN). Only the input layer and the output layer of the network are visible to the operators. The layers in between are called hidden layers and cause the black box nature of these systems. Deep learning gets its name from the deep aspect of the number of hidden layers in a neural network. The deeper a network is, the more layers it has, but also the more parameters and less transparency it has to provide insight into what the neural network is doing between the input and output. DNNs learn a hierarchical representation of the data that is input which is projected into an abstract space and then solved with a linear decision function [8]. Neural Networks are trained using large amounts of data. The data is passed through several layers of artificial neurons with weights and biases that are adjusted through back propagation. The layers of neurons form functions representing the patterns that are observed in the data [7]. Typical activation functions for neural networks are sigmoid, hyperbolic tangent, rectified linear unit (reLU), exponential linear unit (ELU), and leaky reLU [9]. The activation function allows neural networks to break beyond the linear barrier of a single perceptron and are useful for many classification tasks.

There are several hyperparameters that can be adjusted to fine tune the training of a neural network. An epoch is the complete set of data that a DNN is trained on. The number of epochs can be adjusted based on how many times the operator wants the network to train through the data. Gradient descent is a method to optimize the network by minimizing a cost function. It does this by finding an estimate of which direction points to a local minimum and moving towards that minimum. Different kinds of gradient descent include batch, mini-batch and stochastic. The learning rate is the step size between each gradient descent; if it is too large, the algorithm can overshoot, but if it is too small the learning process will take longer and there is a

greater risk of optimizing to a local minimum that is not a global minimum. Another danger of neural networks is that they might memorize their training data and become too overfit to assess new inputs. A way to prevent this is dropout, which randomly shuts off some neurons during the training stage [10]. DNNs are amazing tools that can be modified to suit different tasks.

## Convolutional Neural Networks

There are many types of neural networks that add to the basic ANN structure, but this thesis will focus exclusively on Convolutional Neural Networks (CNN). CNNs are a type of neural network that utilizes convolutions to help classify data [11]. These types of networks excel at image classification tasks because convolutions can make spatial patterns in the data more apparent.

CNNs are made of several different kinds of layers to assess the input data. A convolution layer applies a filter to the input and moves across the data. It can be used for flattening, or to reduce the dimension of the data in addition to finding patterns. The stride is how many elements, or pixels in the case of images, the convolution window moves between applying the convolution filter. A padding of zeros can be added so the edges and corners are not counted fewer times than the data spatially in the center. A fully connected layer is one in which each neuron is connected to every neuron in the following layer. A pooling layer is applied after one or several layers of convolutions to reduce the dimension of the data. Some common types of pooling are max, in which the maximum value is passed to the next layer, and average, where the average of the values in the pool is found and sent to the next layer. These networks are adept at finding spatially focused patterns in data so they excel at image recognition tasks.

## Architectures

Within CNNs, the layers of artificial neurons can be organized in different ways and with varying sizes of convolutions to create different architectures. Over the years, neural network design has improved to make the systems more effective. LeNet-5, developed by Yann LeCun, made history by accurately classifying handwritten numerical digits. It featured 2 convolutional and average pooling layers followed by a flattening convolutional layer and 2 fully connected layers [12]. The Visual Geometry Group developed VGG-16, named for its 16 layers. It stacks 13 small filter convolutional layers followed by 3 fully connected layers. This was important because it allowed the network to have more layers, but by keeping the convolution small it retained some generalization and prevented overfitting [13]. The drawback to this architecture is that it is large and requires an extended period to train. However, the results are excellent for image classification tasks, as might be expected from an algorithm created for the Large Scale Visual Recognition Contest (LSVRC). Another architecture, Network In Network (NIN), uses 1x1 convolutions to increase the power of convolutions and then spatially combines the features afterwards, which allows the convolutions to find patterns while avoiding the loss of data associated with dimensionality reduction [14]. The NIN approach inspired Inception, which uses different sized convolutional blocks to capture different parts of the data. The 1x1 convolution finds cross-feature correlations while larger convolutions retain spatial information [15]. These architectures have led to an increased understanding of neural networks and improved their ability to correctly classify inputs.

## Training Data

The performance of a ML system is dependent upon the quality of the data it is trained on for the task it is assigned. If the training data does not accurately represent the kinds of inputs that the system will encounter in real-world operations or if the data does not accurately reflect the relative frequencies of the types of input that the network will encounter, the results from the network may be incorrect. Similar to the way a student studies, a system will not be able to make sense of inaccurate or contextually inappropriate data. If a student studies an astronomy textbook that was written before Copernicus' theory of a heliocentric system was widely accepted, the student may be well prepared for an exam on the history of theories of the solar system, but not for an exam on how the solar system is understood today. With ML systems, it is also necessary to choose the right data and network for each application.

For supervised learning, it is also important that the training and validation data are labeled correctly. Looking through the massive amounts of data necessary to train a neural network, it can be difficult to identify data that is mislabeled. It is expensive to have people manually categorize images, so a more economical approach would to use the results from an image search engine. Unfortunately, this approach is not reliable and it will usually cause label noise [16]. Looking up images of tigers, a cartoon drawing could come up. If this image was included in the training data for a neural network whose purpose is to tell the difference between animals in the wild, it would interfere with the accuracy of the model.

A small amount of noise can have a large effect on a neural network. For example, if a small amount of static that is imperceptible to the human eye is added to an image of a motorcycle, a neural network might misclassify the picture as a car. A 2014 study, Intriguing Properties of Neural Networks, studied the reasons why slight perturbations in an image can

have a drastic impact on a neural network's output [17]. These small perturbations fool the neural network because the network must ultimately depend upon its training samples to infer the classification of new inputs. If new input data is dissimilar to the training data that was originally used to train the network, this can have a negative impact as the network has no good basis on which to assign a classification to the new input. The input-output mappings that neural networks learn can be significantly discontinuous if the training data is too sparse. Many kernel methods assume that the data is distributed smoothly, but this often clashes with the discontinuous input-output mappings that are actually available. A system should be trained with enough data to learn a more continuous mapping if it is to perform well. Training data needs to be tailored to each specific application and vetted to ensure that it is accurately representing the distribution of data the system will see in application.

## Confidence

As the outputs from neural networks are not always accurate, it is important to be able to appraise a model and measure how well it performs. Confidence scores provide a mechanism by which a neural network can report an assessment of how much confidence it has in the prediction that it has made. Confidence can be thought of as a measure of how different an input is to the training data [8]. One basic way to calculate confidence in predictions for neural network outputs is by applying a *softmax* function to the final output layer of the neural network which gives a probability distribution of the outputs.

Unfortunately, confidence is not always correlated to correct results. One particular area of concern is where the model makes predictions with high confidence but is wrong. On the

other hand, there are instances where the model makes predictions with low confidence but is correct. The former situation is more concerning because the operator is often less likely to pay attention to cases in which the machine learning algorithm reports that it has high confidence, whereas predictions that are made with low confidence are more likely to be flagged for manual operator review. Confidence is one way to increase operator trust in a neural network, but it is not consistently accurate enough to warrant complete faith in outputs assigned high confidence measures.

## Ethical Considerations

As ML and DNNs have increasingly widespread applications, further research in the area of trust assurance is important. Being able to trust the results of machine learning systems will facilitate the use of these systems in more areas. In an ideal situation, a correct result would have high confidence and incorrect result would have low confidence, but this is not always the case. Mistakenly high trust can lead to undesirable results when incorrect outputs are not appropriately flagged with a warning. For example, if an autonomous vehicle that analyzes video of a road to avoid crashing says with high confidence that it is going the correct path, but it is in fact mistaking some rocks on the side of the road for the road itself, the results could be catastrophic. In this instance, having a high confidence in a result that is incorrect has serious negative consequences.

There are biases present in all areas of life, whether these biases are intentional or not, and huge impacts on society can result when a seemingly small bias is consistently applied on a large scale. This is especially true in the realm of ML as the data that is available has a direct

impact on the output of these systems. There are some topics that don't have the same amount of data as others, which will result in those topics being underrepresented by the neural network. Additionally, if these systems are used for services that affect important aspects of life, prejudiced outcomes from past experiences can be present in the modern system. For example, if a ML system is used to help facial recognition, but primarily males are used as the training data, the system may not be able to detect females as well. The ImageNet Dataset identified a similar issue with its images and sought to rectify it [19]. However, even with the best purposeful intentions, data can still be skewed. There can also be a compound effect with biased data if the data used to train the network was collected through a means that was also biased. For instance, if a google image search which is trained to show the most popular images first is used to collect training data, the biases from the search engine and from previous searches will be present in the network as well. ML systems learn by looking at past data, so if that data contains hidden biases, the outputs of the system will hold those same biases.

Additionally, privacy is another issue of major concern. Neural networks must be trained on immense amounts of data, and it is possible to use the outputs of a neural network to reverse engineer the system and learn about the training data even if the training data is not directly available [20]. Neural networks that are trained using protected personal information could therefore potentially leak this information to an attacker. This raises the question of whether sensitive data should be used to train neural networks in the first place when disclosure of the training data cannot be fully prevented.

No matter how many assurances are in place, it is impossible to predict how a system will react in every single scenario. In these instances, there is the legal aspect of establishing who

needs to shoulder the liability for a misstep. For example, if an autonomous vehicle gets into an accident with another car that is driven by a human, how can society establish who is at fault—and if the autonomous car is found to have made the mistake, who should be liable? One could argue that the owner of the car, the manufacturer, and the designer of the system guiding the car would all be at least partly responsible. ML systems have the potential to improve life greatly by taking over the performance of tedious tasks, however, they introduce new legal and moral concerns that must be addressed before they can be widely adopted. This is not an all-inclusive review of the ethics involved in ML applications, as there are many aspects to consider.

—Remainder of Page Intentionally Left Blank—

# Existing Trust Methods for DNN

There are numerous methods beyond the *softmax* confidence calculation mentioned earlier that can be used to improve the assurance of trust in a system. Three of these methods are presented here.

## Deep Confidence

Deep COnfiDEnce (DECODE) is a framework that can be combined with an existing ML architecture in an end-to-end way to increase assurance that the output of the ML system is correct. DECODE was introduced in a paper, Deep Confidence Network for Robust Image Classification, that investigated data with mislabeled images [16]. If a network is trained on data that is incorrect, it will likely produce results that cannot be trusted. It can be beneficial to try to identify and remove suspicious, possibly mislabeled data from the training data and to re-train the neural network; however this can also backfire by reducing the total amount of training data too much and causing the re-trained neural network to under fit. One strategy to balance this is to purposefully include noisy images in the dataset. This reduces the effect of the mislabeled images on the outcomes of the network [16].

Given a training dataset, DECODE first applies some label noise. It then checks the confidence of the model for each sample and assigns small training weights to those samples whose confidence is low. During training, stopping the model early and applying the obtained weights and biases to all labeled training samples will cause noisy and mislabeled samples to have lower confidence. The training samples with lower confidence are assigned smaller weights to help reduce the impact of noisy and mislabeled samples on the network. Thus a model trained

with a small number of labeled samples can be applied to a large number of unlabeled samples. The unlabeled samples that have the highest confidence are treated as if they were labeled samples from that class, based on the assumption that their predicted labels are correct. This way the network has more data to train with and is less likely to underfit. It is still possible for incorrect training data to negatively affect the network, but this method reduces the probability of those incorrect points having a significant impact on the network.

The clustering hypothesis assumes that data points from the same class should form a cluster and be similar to each other. However, with real world data it is possible to have multiple cluster centers for a class. Keeping track of instances where a data point in a class is closer to the center of another class than its own can help to account for this. A violation factor can be defined as a count of the number of violations or a distance measuring how far each data point is from the center of its class. The confidence value can then be adjusted as a function of the violation factor which accounts for more than the *softmax* confidence. The two important factors of DECODE are training a deep network and then training a confidence evaluation module. The confidence module evaluates the confidence of a sample and assigns a training weight in order to suppress the influence of mislabeled samples. Overall, DECODE is similar to training a regular deep convolutional neural network, while also computing the violation factor, confidence, and training weight.

Another study applied deep confidence to medical data, but the principles and framework applied in the study are transferrable to other applications and contexts [21]. Conformal predictors were used to provide a level of confidence. In this case, the confidence has a lower bound for the fraction of predictions that will be correct. A snapshot ensemble, which is a collection of multiple neural network weights and biases captured during training, is one method

to create more usable data. This ensemble of deep neural networks is generated from the training of a single neural network so extra computation is not needed for their creation. The snapshots are created during the training by adjusting the learning rate and saving the weights and biases whenever a local minimum is found. Each snapshot is slightly different so predictions from the neural networks are correlated but they will have slightly different outputs. Noisy inputs can be beneficial to an Ensemble method by training a model with a clean dataset and comparing the output to that of a model trained with a noisy dataset [21]. An alternate strategy is to construct a dataset with clean labels to assist training with noisy labels. Then use both clean and noisy datasets to pre-train a model but fine tune only with the clean set. This way the noisy data can act as a restraint to keep the data from over fitting while the clean data makes sure it is correct. However, this strategy requires knowledge of which data is noisy and which is not, and making this determination may be a resource intensive endeavor. Overall, deep confidence is helpful in increasing the consistency of the confidence measure in a network.

## Defensive Distillation

Another method to protect DNNs from misclassification is to use Defensive Distillation to train them. This approach creates a network that is able to handle noisy input data. In general, the idea behind Defensive Distillation is to train one DNN using knowledge transferred from another DNN [22]. The specific steps for Defensive Distillation are as follows. First, a teacher DNN is trained and evaluated on each instance of training data to produce soft labels. Then a temperature parameter is applied in which a higher temperature normalizes the outputs. The outputs from the teacher network are used to create soft labels for the training data. The soft

labeled training data is then used to train a second, distilled network. This process essentially double trains the data which reduces overfitting and smooths the model so it generalizes better. As a result, it is more robust to adversarial examples. However, defensive distillation is not all encompassing and can still misclassify inputs. Defensive distillation is a good strategy to help protect neural networks and increase the likelihood that a DNN will output correct results, but it is not foolproof.

## Deep k-Nearest Neighbors

Assuring trust in the output of a neural network is difficult because of the black box nature of the system. Ascertaining how a DNN came to its decision will help increase trust in its outputs. One approach to increasing the interpretability of DNNs is to combine the machine learning technique of K-Nearest Neighbors (KNN) with the layers of a DNN. McDaniel and Papernot developed a trust assurance technique called Deep k-Nearest Neighbors (DkNN) which utilizes the KNN algorithm to compare the weights of each layer and make a classification prediction with added information about the output's relation to the training data [8]. KNN analyzes past data and makes predictions about data inputs by comparing the input to the k

**Algorithm 1 – Deep k-Nearest Neighbor.**

**Input:** training data $(X, Y)$, calibration data $(X^c, Y^c)$
**Input:** trained neural network $f$ with $l$ layers
**Input:** number $k$ of neighbors
**Input:** test input $z$
// Compute layer-wise $k$ nearest neighbors for test input $z$
**for** each layer $\lambda \in 1 \,..\, l$ **do**
  $\Gamma \leftarrow k$ points in $X$ closest to $z$ found w/ LSH tables
  $\Omega_\lambda \leftarrow \{Y_i : i \in \Gamma\}$ ▷ Labels of $k$ inputs found
**end for**
// Compute prediction, confidence and credibility
$A = \{\alpha(x, y) : (x, y) \in (X^c, Y^c)\}$ ▷ Calibration
**for** each label $j \in 1..n$ **do**
  $\alpha(z, j) \leftarrow \sum_{\lambda \in 1..l} |i \in \Omega_\lambda : i \neq j|$ ▷ Nonconformity
  $p_j(z) = \frac{|\{\alpha \in A : \alpha \geq \alpha(z,j)\}|}{|A|}$ ▷ empirical $p$-value
**end for**
prediction $\leftarrow \arg\max_{j \in 1..n} p_j(z)$
confidence $\leftarrow 1 - \max_{j \in 1..n, j \neq prediction} p_j(z)$
credibility $\leftarrow \max_{j \in 1..n} p_j(z)$
**return** prediction, confidence, credibility

**Figure 6: Pseudo code for the DkNN algorithm described [8]**

data points that are most similar to it using local sensitivity hash tables. The algorithm for this method is shown in Figure 6. This can be applied to neural networks on a layer-by-layer basis, with the output of each layer making a KNN prediction about the final output. If the predictions from each of the layers are consistent with the final output, the system can be more confident with its prediction than if the consensus was changing as the data went through the model. This is called credibility, which is a measure of how much the input data's prediction is supported by the training data. The credibility establishes the relevance of the training data to the input data, while confidence establishes the likelihood that the predicted label is correct. If the layers agree with the overall output from the network, it is more likely that the neural network's prediction was correct. DkNN can perform better with noisy data than a normal DNN because when noisy inputs run through a neural network, the perturbations can cause the input data to switch from the class it should be, but DkNN looks at the predictions at each layer and is able to detect when the predicted output changes. DkNN improves upon the typical *softmax* prediction that yields only a confidence score. These two measures in combination are able to give the operator a pretty good idea of how an input relates to the system, and thus a better idea of whether the resulting output can be trusted.

## Experimental Setup

DkNN is not immune to misclassification, so there is much more work to be done before this method can be considered reliable. After examining the DkNN algorithm and other trust assurance methods, potential ways to improve DkNN were investigated. A few additions to the algorithm may lead to more insight into DNNs as well as more trust in their outputs. These experiments are designed to be carried out using a CNN, specifically VGG-16. The recommended training and validation data is the ImageNet Database.

### Augmentations

At its core, DkNN is an application of the KNN algorithm to DNNs, so exploring the base KNN algorithm can potentially shed light on the system as a whole. If KNN can be improved, then DkNN can potentially be improved along with it. The proposed exploration into the base KNN algorithm involves investigating the purity of each of the neighbors. In this case, purity refers to whether the given neighbor is a good representation of its class. However, it makes sense to save time and resources by experimenting with KNN before trying to apply it to the more resource intensive DkNN. This algorithm is hypothesized to be improved by assigning weights to the "k" closest neighbors, so that data points that are more clearly part of each group will be more influential in the calculation. This is accomplished by calculating a KNN score for each of the known data points so if the "k" points around it agree with its own class, it is very likely that the point is a good representation of the class. On the other hand, if the "k" points around a training sample don't agree with its class, it may be an outlier or on a boundary between classes and therefore less representative of the class distribution. The points with higher

agreement can be assigned a larger weight and therefore be given a larger impact on the calculation as a whole. This weight can be calculated as the "k" nearest points in its same class divided by "k." Figure 7 describes this augmented algorithm in greater detail. The recommended plan of action is to explore this method by using plain KNN first and if the weighted "k" values lead to more consistently accurate results, future research could integrate it into the DkNN method.

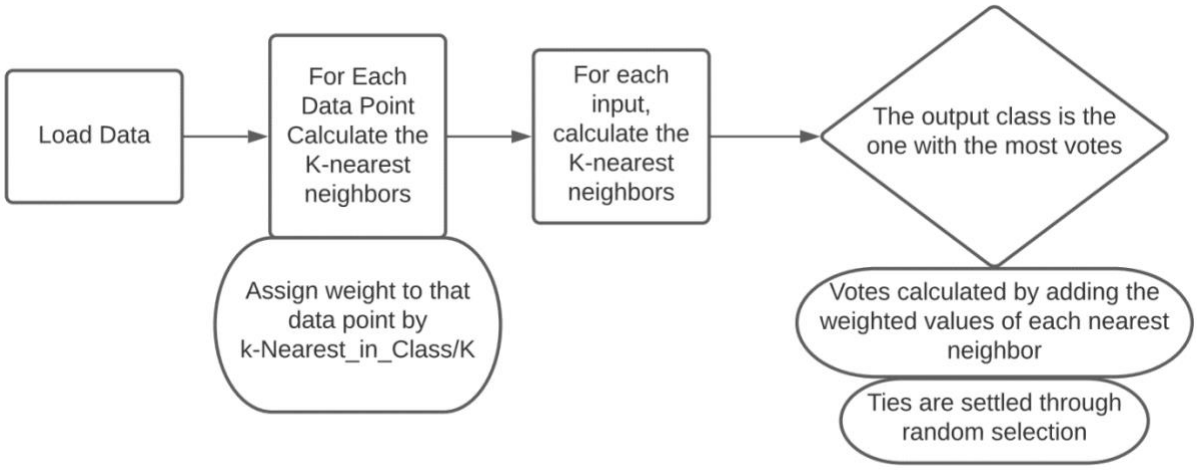


**Figure 7: Flowchart of KNN Purity Experiment**

The effectiveness of the KNN algorithm can change with varying values of "k" based on the distribution of the data as seen in Figure 2 earlier. However, there is not an accepted statistical way to choose the best "k" beyond starting with a random value, and then updating "k" to find the number that yields the most consistently correct output with the validation data. As such, adjusting values for "k" when applying its principles to neural networks in DkNN might alter its output as well. To test this with DkNN, the network would train once and then the validation step would be rerun several times with different "k" values to find the optimal "k" for the given training data. This process could be automated by starting with a "k" value that is equal to the square root of the number of samples minus half the square root of the number of samples

and incrementing the "k" value until it is the square root of the number of samples plus half the square root of the number of samples. This range should cover the optimal "k"—as it is typically found to be around the square root of the number of training data and the total samples should be a large number since it is training a DNN—while still increasing the efficiency of checking all possible values for "k." The optimal value can be chosen by looking at the accuracy of the validation data with each "k." Then the "k" with the highest accuracy is chosen. This augmentation is outlined Figure 8. Examining the distributions of data—how many classes are defined and how many examples are in the training data and validation data of each class compared to the effective "k"—will shed light on how an optimal "k" can be chosen moving forward and on how this strategy might be applied to other applications and neural networks. An effective "k" value for DkNN will have a high correlation of high credibility scores and high confidence scores with correct outputs.

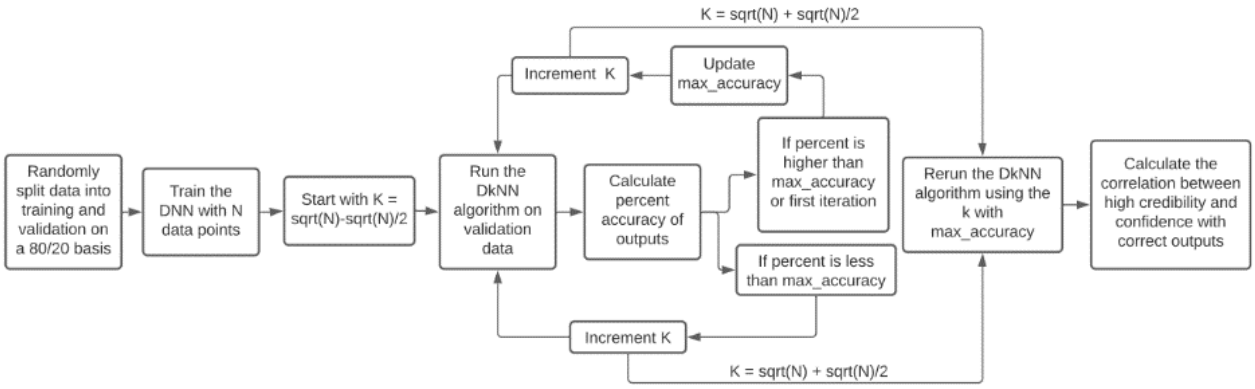


**Figure 8: Flowchart of Varying k Augmentation**

Another addition to DkNN would be to explore the effectiveness of focusing on the different layers of the neural network. Papernot established that DkNN performs well with noisy data because it detects a change in the predicted output between the layers, demonstrating the layer by layer comparison is important [8]. For this experiment, the outputs of the KNN

calculations from the different layers for each input should be analyzed to see if there is a correlation between a certain KNN value and the final output. KNN class labels will be assigned at different layers and the accuracy of the class labels will be assessed in accordance to their agreement with the final outcome as well as the correct outcome. This could be validated by determining if the first layer which correctly classified inputs that agree with their KNN-based classification is relatively consistent for many different input samples. This experiment will also shed some light on the area of interpretability by collecting data on how and where the network makes its decisions. A hypothesis is that earlier layers find more broad patterns within the data while later layers find specifics and are able better able to distinguish between highly similar classifications. It stands to reason that this experiment could help to confirm this by looking to see if general classifications are made earlier on but the specifics are decided later in the network. An example of how to think about this would be an input photo of a mailman that may at first be categorized as a security guard but a later layer specifies it to be a mailman. In this case the concepts of a person and a uniform may be identified earlier in the network, but deciding the specifics of that uniform and to which class it belongs are developed later.

A further area that is worth investigating is the efficacy of the augmentations for various types of classifications. To ensure that the data collected from these experiments is not overly dependent on the data the neural network is trained on, it is beneficial to repeat these experiments using several subsets of data which can be randomly selected and tested. In addition to the VGG-16 recommended architecture, another adjustment that could be explored would be changing the architecture to see how that changes the outcomes and whether the results differ or remain the same. The order, number, and sizing of the layers could be adjusted. As seen with the evolution of some famous CNNs that were outlined in the Architectures section, the layers of a

network can affect its outputs. Repeating the earlier experiments using different datasets and architectures would help to evaluate whether any observed increase in the effectiveness of the DkNN algorithm is due to the added augmentations or whether it is a coincidental side effect of the data or a specific architecture.

A summary of the experiments is presented in Table 1. These augmentations may lead to more effective credibility and confidence scores that will increase an operator's trust in their DNN. When discussing these changes from now on, they will be referred to as Augmented Deep k-Nearest Neighbors (ADkNN).

| Experiment | Summary | Evaluation | Impact |
|---|---|---|---|
| K Purity | The KNN algorithm is adjusted so that the closest neighbors are given different weights based on how well they represent their class | It is successful if there is an increase in the accuracy of the model's predictions | The KNN algorithm could have increased likelihood that it is correct and this could be applied on the DkNN level in the future |
| Varying k | The DkNN algorithm is altered so that it tests several different values of k | It is successful if there is a correlation between high confidence/ credibility scores and correct classifications | This could increase the trust assurance measure put forth by the DkNN |
| Layer Exploration | The KNN scores at each layer of the DNN are analyzed to see where the system makes decisions | Chart which layers the KNN score matches the output and whether it was classified correctly | The results from this experiment could shed more light onto interpretability of DNNs |
| Type of Classification | Repeat earlier experiments on different selections of data and adjust the architecture being used | It is successful if the correlations between high confidence/credibility scores and correct classifications remain high across the different data subsets and architectures | This would ensure the results from earlier are consistent across different datasets and architectures |

**Table 1: Summary of Presented Experiments**

## Framework

Before the proposed experiments can be attempted, it is necessary to first develop the tools to be used and the framework on which the experiment will be run. Selecting the coding language in which to develop the proposed trust assurance augmentations is another important decision. Keras and TensorFlow are a prime choice because they are well documented and based in Python and therefore familiar to many programmers. These tools offer many features which assist in the creation of neural networks. For instance, a model can easily be built by choosing the number and type of layers that are desired for each step of the system. Double checking that the model was built as intended is easy with a model summary feature. This is ideal for the presented experiments because they revolve around the validation step, which is where the new code will need to be written, while Keras and TensorFlow are able to assist with the creation and training of the model. The activation functions and other hyperparameters can be adjusted easily by changing parameters in these commands. TensorFlow also has helpful toolkits like TensorBoard that are able to create visuals about the features of the neural network which will help in evaluating the outputs. Thus Keras and TensorFlow are recommended to conduct this experiment. Additionally, these augmentations involve a large number of program runs and each test will likely be time consuming, so automating this would be better than manually executing each test. This can be accomplished by writing a shell script that will run each experiment incrementally through the different “k” values, datasets, and architectures.

## Experimental Data

A ML system can only perform as well as the data that it is trained on, so it is important to use appropriate datasets when evaluating them. As KNN has less intensive training data needs than DNNs, the KNN purity experiment can use the irises dataset. This is a simple dataset that contains three classifications of irises and information about their sepal and petal width and length [32]. This dataset is appropriate for evaluating the augmentation of KNN that was presented because it is a relatively small dataset that will not take much time to compute, but the data distribution and format works well for evaluating KNN.

On the other hand, the experiments involving the DNN require a more complex dataset. Deep learning requires an immense amount of data to train and validate the system, so it is necessary to choose a dataset with enough information to support the training and validation stages. Additionally, comparing different network architectures and methods can be difficult because they are tailored to the specific task for which they were designed. Competitions like the LSVRC use the same large dataset with the goal of having neural networks accurately categorize images. The LSVRC uses the ImageNet Database which was developed by the Stanford Vision Lab [23]. It contains a large number of images with labels that have been verified as accurate. This allows experiments to be run with confidence in order to test different methods without worrying about incorrect data. This dataset has proven effective in the past as many of the architectures that improved CNNs, and were mentioned earlier, were introduced as a part of the LSVRC and trained using the ImageNet dataset [13]. This thesis proposes to use the ImageNet dataset because it contains annotated images and has been used for research in the past.

The ImageNet dataset is built on the backbone of the WordNet structure, which is a database that identifies connections between words. This creates an expansive way for the

classes to be annotated in ways that make sense and that can connect an image to several different concepts. Additionally, this allows for the creation of a hierarchy between classes—for example, there can be an overarching cat category as well as different breeds of cats within that class. The annotations are split up using synonym sets (synsets) which group data elements that go together. When ImageNet was first released, it had 5,247 synsets and about 3.2 million images in total [31]. Since then it has grown in size and accuracy. Each image must be viewed by several human actors with a certain level of agreement about the synsets present before it is added to the database.

Furthermore, the ImageNet dataset has addressed concerns that have been brought up through research and by members of the community. The ImageNet team is actively working to correct inherent bias in the dataset [29]. Additionally, there are privacy concerns relating to images that include people's faces, so the ImageNet team is working to make sure that the dataset is still able to be an effective dataset while protecting the privacy of those pictured [30]. There is still work to be done to ensure that ML systems are as fair and unbiased as possible, but starting with a dataset that is actively working towards good practices is a good starting point. Training data needs diversity, quantity, and accuracy. All of these are present in the ImageNet dataset, and it is widely used by researchers, making it a good choice for evaluating trust assurance methods.

## Evaluation

The KNN purity experiment can be evaluated by comparing the accuracy of a plain KNN on the Iris dataset with a KNN that has the purity weights added to the nearest neighbors. By

incrementing the “k” value and comparing the purity KNN with the regular KNN for each value of “k”, the potential benefits of the purity addition can be evaluated. If the KNN algorithm is more accurate with the purity calculations, then the purity score can be used to enhance KNN and possibly be added to the DkNN algorithm as well.

As this is a trust assurance technique, it is important that the results proving that this technique is successful, or unsuccessful, can also be trusted. Having a faulty evaluation system can be worse than not having one at all because false confidence may lead to misplaced trust and increased skepticism of ML systems. As with any experiment, a key point is to isolate the changes so that improvements can be attributed to the augmentations. It is necessary to compare the results from a regular softmax confidence CNN, plain DkNN, and ADkNN. The baseline CNN confidence correlation and DkNN confidence and credibility correlations can be directly compared with the ADkNN confidence and credibility correlations. It will be especially important to look at instances where these models’ outputs disagree.

The correlation of the output prediction and label with the credibility and confidence scores will show the effectiveness of ADkNN. For the varying K and changing architecture and dataset experiments, the augmentations are increasing the interpretability and trust assurance of the network if the credibility and confidence scores are high with correct output predictions and low with incorrect output predictions. This can be calculated by splitting the correctly classified images into groups of high, medium, and low credibility and confidence. High credibility and confidence scores can be defined as values over 67%, medium scores as between 34% and 66%, and low scores as below 33%. Tables 2 and 3 demonstrate how to make these correlation calculations and how they relate to the success or failure of these methods.

| | Correctly Classified Images ($I_C$) | | |
|---|---|---|---|
| | Number of High Confidence Instances ($F_{HC}$) | Number of Medium Confidence Instances ($F_{MC}$) | Number of Low Confidence Instances ($F_{LC}$) |
| | Number of High Credibility Instances ($D_{HC}$) | Number of Medium Credibility Instances ($D_{MC}$) | Number of Low Credibility Instances ($D_{LC}$) |
| Confidence Correlation | $\frac{F_{HC}}{I_C}$ | $\frac{F_{MC}}{I_C}$ | $\frac{F_{LC}}{I_C}$ |
| Credibility Correlation | $\frac{D_{HC}}{I_C}$ | $\frac{D_{MC}}{I_C}$ | $\frac{D_{LC}}{I_C}$ |
| Success if Correlation is | High | Medium | Low |

**Table 2: Calculating the Correlation of Success with Correctly Classified Images**

| | Incorrectly Classified Images ($I_I$) | | |
|---|---|---|---|
| | Number of High Confidence Instances ($F_{HI}$) | Number of Medium Confidence Instances ($F_{MI}$) | Number of Low Confidence Instances ($F_{LI}$) |
| | Number of High Credibility Instances ($D_{HI}$) | Number of Medium Credibility Instances ($D_{MI}$) | Number of Low Credibility Instances ($D_{LI}$) |
| Confidence Correlation | $\frac{F_{HI}}{I_I}$ | $\frac{F_{MI}}{I_I}$ | $\frac{F_{LI}}{I_I}$ |
| Credibility Correlation | $\frac{D_{HI}}{I_I}$ | $\frac{D_{MI}}{I_I}$ | $\frac{D_{LI}}{I_I}$ |
| Success if Correlation is | Low | Medium | High |

**Table 3: Calculating the Correlation of Success with Misclassified Images**

# Closing Remarks

This thesis has presented an overview of ML strategies with an emphasis on DNNs, and also several augmentations that may improve the DkNN trust assurance strategy. ML is a growing field that is already utilized in many areas of life, from search algorithms to suggested words when sending a text, and work is being done to expand ML into other areas of life as well, like autonomous vehicles. DNNs have the potential to make life easier by automating many classification tasks; however, they are black box systems so more interpretability and greater assurances of accuracy are needed before they can safely be trusted with more important responsibilities.

Improving the trust assurance methods for neural networks may enable the widespread use of these systems to increase productivity by assuring that the systems will perform correctly and as intended. The experiment proposed to augment DkNN can be implemented and hopefully will increase the level of assurance that the algorithm can provide that its outputs are correct. There is much more work that must be done before DNNs are more interpretable and able to be trusted enough to be applied safely and ethically in many situations.

# Appendix: Non-Technical Abstract

This thesis explores Machine Learning (ML) techniques with an emphasis on Deep Neural Networks (DNN) and expands upon an existing method of trusting their outputs. DNNs are able to mimic the way a human learns through artificial neurons, which are a mathematical representation of the biological neuron. Similar to the way a child can learn that a dog is the fluffy animal that lives at the neighbor's house because the child's parents point to it and call it a dog, a DNN can train through the data that an operator provides so that it is able to identify a dog. Unfortunately this is not foolproof. Just as a child may misidentify a cat as a dog, a DNN can misclassify its inputs. It can be assumed that the child saw a similarly shaped animal and came to the conclusion that it must also be a dog, but there is no way to know for sure how they came to that assumption until they are old enough to explain their thinking. A DNN can be thought of in a comparable way, in that it is not able to explain how it arrived at its output. In order for these systems to be used in important parts of life, the operators need to be assured that their systems will make the correct choices or else the system cannot be relied upon. Deep k-Nearest Neighbors is a method that gives the operator a greater indication as to whether the output of the DNN is correct. This thesis presents possible methods to increase the effectiveness of DkNN along with a plan for evaluation.

# BIBLIOGRAPHY

[1] Shorten, Connor and T. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning.” *Journal of Big Data* 6 (2019): 1-48.

[2] Nguyen A, Yosinski J, Clune J. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In Computer Vision and Pattern Recognition (CVPR ’15), IEEE, 2015.

[3] Cover, T. “Estimation by the nearest neighbor rule.” *IEEE Transactions on Information Theory,* IT-14 (1968): 50-55.

[4] Belson, W. A.. “Matching and Prediction on the Principle of Biological Classification.” *Journal of The Royal Statistical Society Series C-applied Statistics* 8 (1959): 65-75.

[5] Fournier, Dominique. (2000). Using Impurity and Depth for Decision Trees Pruning.

[6] Zhang, Cha, and Yunqian Ma. *Ensemble Machine Learning*. Springer, 2012.

[7] McCulloch, W. and W. Pitts. “A logical calculus of the ideas immanent in nervous activity.” *Bulletin of Mathematical Biology* 52 (1990): 99-115.

[8] Papernot, Nicolas and P. McDaniel. “Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning.” 13 Mar. 2018. *ArXiv* abs/1803.04765 (2018).

[9] Nwankpa, Chigozie et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning.” *ArXiv* abs/1811.03378 (2018): n. pag.

[10] Srivastava, Nitish et al. “Dropout: a simple way to prevent neural networks from overfitting.” *J. Mach. Learn. Res.* 15 (2014): 1929-1958.

[11] Krizhevsky, A. et al. “ImageNet classification with deep convolutional neural networks.” *Communications of the ACM* 60 (2012): 84 - 90.

[12] LeCun, Y. et al. “Gradient-based learning applied to document recognition.” (1998).

[13] Simonyan, K. and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” *CoRR* abs/1409.1556 (2015): n. pag.

[14] Lin, M. et al. “Network In Network.” *CoRR* abs/1312.4400 (2014): n. pag.

[15] Szegedy, Christian et al. “Going deeper with convolutions.” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015): 1-9.

[16] G. Ding, Y. Guo, K. Chen, C. Chu, J. Han and Q. Dai, "DECODE: Deep Confidence Network for Robust Image Classification," in IEEE Transactions on Image Processing, vol. 28, no. 8, pp. 3752-3765, Aug. 2019, doi: 10.1109/TIP.2019.2902115.

[17] Szegedy, Christian, et al. “Intriguing Properties of Neural Networks.” *Arxiv.org*, 19 Feb. 2014, arxiv.org/pdf/1312.6199.pdf.

[18] Shrivastava, Abhinav et al. “Training Region-Based Object Detectors with Online Hard Example Mining.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016): 761-769.

[19] Recognition., IEEE Conference on Computer Vision and Pattern. *2009 IEEE Conference on Computer Vision and Pattern Recognition: Miami, Florida, USA, 20-25 June 2009*. IEEE, 2009.

[20] Bourtoule, Lucas et al. “Machine Unlearning.” *ArXiv* abs/1912.03817 (2019): n. pag.

[21] Cortés-Ciriano, Isidro, and Andreas Bender. “Deep Confidence: A Computationally Efficient Framework for Calculating Reliable Prediction Errors for Deep Neural Networks.” *Journal of Chemical Information and Modeling*, vol. 59, no. 3, 17 Oct. 2018, pp. 1269–1281., doi:10.1021/acs.jcim.8b00542.

[22] Papernot, Nicolas, et al. “Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks.” Https://Arxiv.org/Pdf/1511.04508.Pdf. *37th IEEE Symposium on Security & Privacy*, 14 Mar. 2016, pp. 582–597., doi:10.1109/SP.2016.41.

[23] *ImageNet*, Stanford Vision Lab, image-net.org/.

[24] Goodfellow, Ian J. et al. “Explaining and Harnessing Adversarial Examples.” *CoRR* abs/1412.6572 (2015): n. pag.

[25] Kumar, Janaki. “Human Impact of Biased AI in Business - and How to Go Beyond.” *SAP Blogs*, SAP Community Blogs, 15 Jan. 2021, blogs.sap.com/2018/07/01/human-impact-of-biased-ai-in-business%E2%80%8A-%E2%80%8Aand-how-to-go-beyond-2/.

[26] Zhang, Grace. “What Is the Kernel Trick? Why Is It Important?” *Medium*, Medium, 11 Nov. 2018, medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d.

[27] Sharma, Sagar. “What the Hell Is Perceptron?” *Medium*, Towards Data Science, 11 Oct. 2019, towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[28] Ivakhnenko, A. G. and Lapa, V. G. (1965). Cybernetic Predicting Devices. CCM Information Corporation.

[29] Yang, Kaiyu et al. “Towards fairer datasets: filtering and balancing the distribution of the people subtree in the ImageNet hierarchy.” *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (2020): n. pag.

[30] Yang, Kaiyu et al. “A Study of Face Obfuscation in ImageNet.” *ArXiv* abs/2103.06191 (2021): n. pag.

[31] Deng, Jia et al. “ImageNet: A large-scale hierarchical image database.” *CVPR* (2009).

[32] Fisher, R. A.. “The Use of Multiple Measurements in Taxonomic Problems.” *Annals of Human Genetics* 7 (1936): 179-188.

—Remainder of Page Intentionally Left Blank—

# ACADEMIC VITA

**The Pennsylvania State University**
**Schreyer Honors College**
*Bachelor of Science in Engineering Science*
*Minor in Computer Engineering*

**Penn State Applied Research Laboratory**
Research and Development Intern
*January 2020 – May 2021*

- Independently researched Machine Learning systems, focusing on Deep Learning

**SPS Technologies-A PCC Company**
Engineering Intern
*May 2019 – August 2019*

- Investigated root cause of defects and developed corrective actions
- Implemented procedures for transporting heavy tools as part of a company safety initiative
- Streamlined bolt manufacturing processes alongside experienced engineers

**Penn State Learning**
Peer Math Tutor
*September – December 2018 and 2019*

- Assisted other students in learning math topics from algebra through multivariable calculus
- Attended an adult learning course to develop multiple ways to convey complex information

**Environex Inc.**
Engineering Lab Technician Intern
*May – August 2017 and 2018*

- Collaborated with Project Engineers and a Laboratory Specialist to predict catalyst lifetime
- Customized Excel macros to create unique worksheets to improve efficiency of data entry

**Penn State Formula Racing (FSAE)**
Aerodynamics and Electronics Subsystems Member
*March 2018 – May 2020*

- Contributed to a multi-disciplinary team to design, build, and test a Formula One style racecar

**Li Reactive Water Group**
Undergraduate Research Assistant
*January 2018 – December 2018*

- Analyzed large data files using MATLAB to advance watershed models as part of a research group

**Study Abroad in Italy**
Student
*January 2019 – April 2019*

- Navigated a foreign country and tutored local students in English