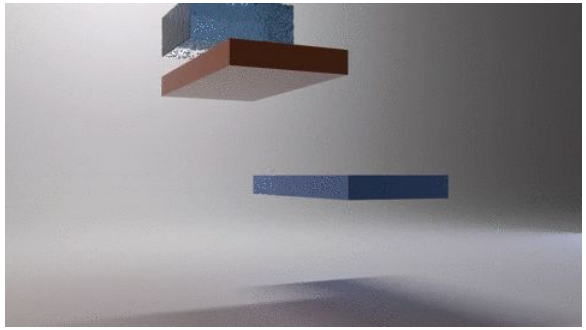
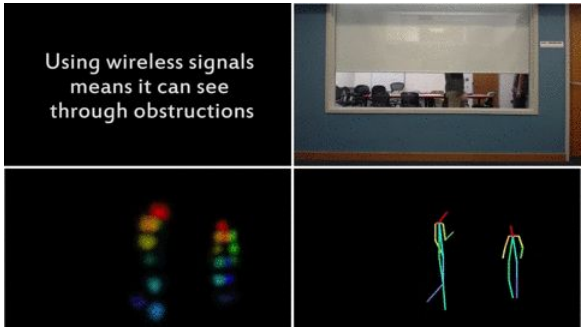
Improving the Robustness of Convolution Neural Networks using N-Version Programming

Carl Hildebrandt, Advait Kulkarni and Tyler Williams

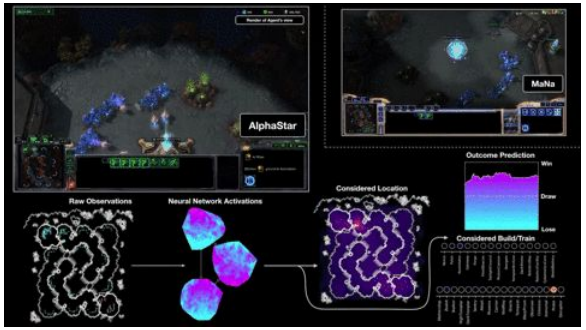
Neural Networks



Liquid Splash Modeling with Neural Networks



Through-Wall Human Pose Estimation Using Radio Signals



AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

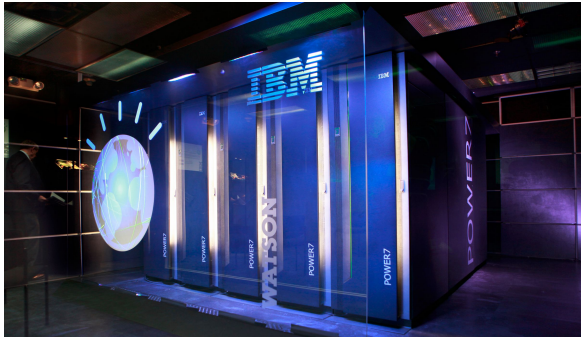
Elon Musk clarifies Tesla's plan to retrofit cars for 'Full Self-Driving' with new HW3 computer

A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots

Transforming Healthcare to Transform Lives

Neural Networks

Chinese billionaire’s face identified as jaywalker



IBM Watson gives wrong recommendations on cancer treatments

Google Translate shows gender bias in Turkish-English translations

Uber self-driving car kills a pedestrian

Facebook allowed ads to be targeted to racist or anti-Semitic views

Apple iPhones face recognition can be tricked by mask

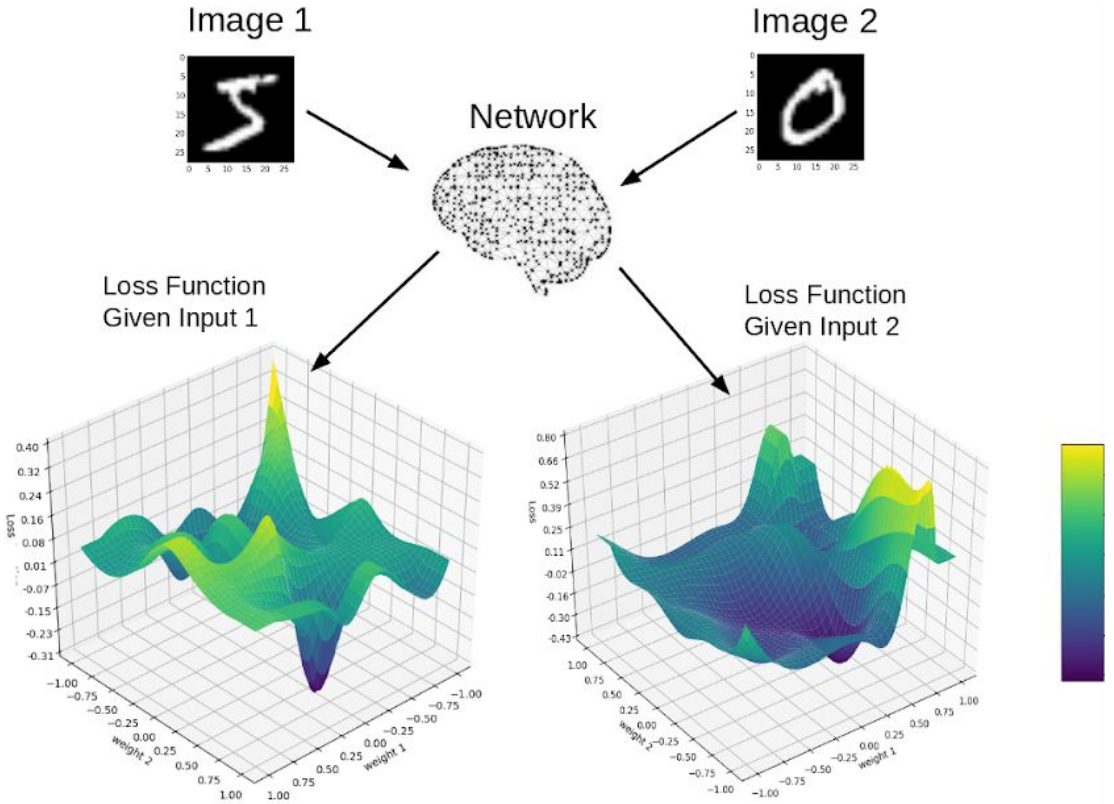
Can we **increase** the **robustness** of a network using **N-Version programming**

Goals

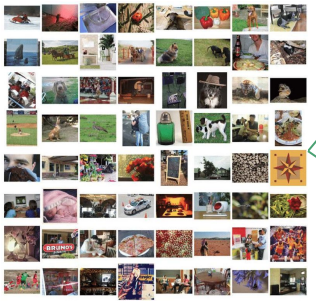
- How robust are neural networks?
- Which adversarial attacks are most successful?
- Are adversarial attacks defendable through N-Version programming?
- Can we use N-Version programming to make more robust neural networks?

Background - Neural Networks

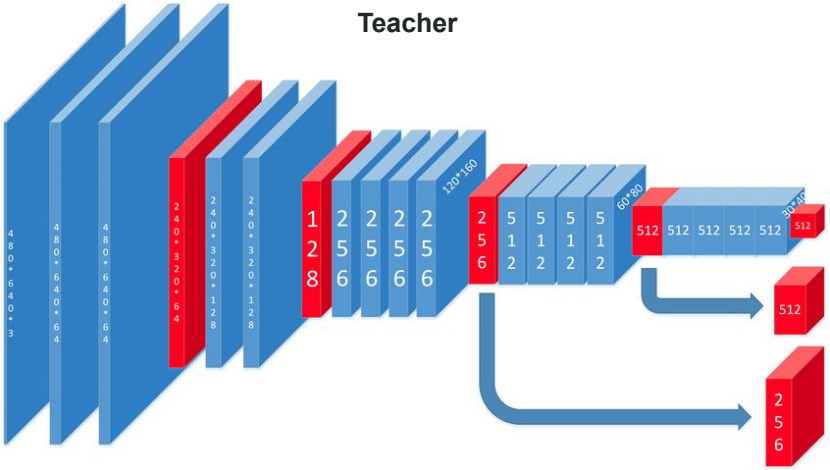
- The goal of a neural network is to minimize a loss function.
- Uses gradient descent to find a minima.
- This minima might not give us the most robust network performance.



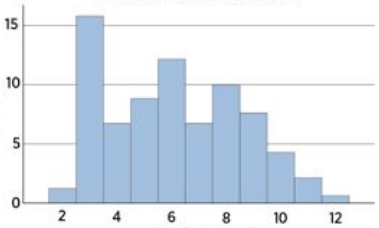
Background - Network Distillation



Teacher

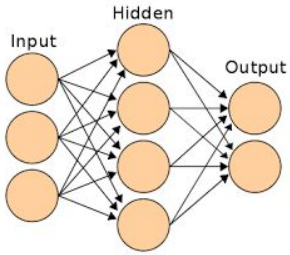


VGG-19: Roughly two week training time

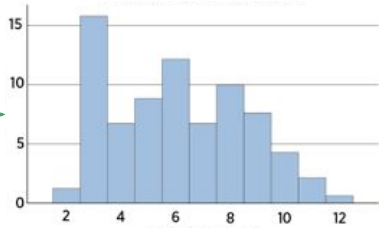


Try to match

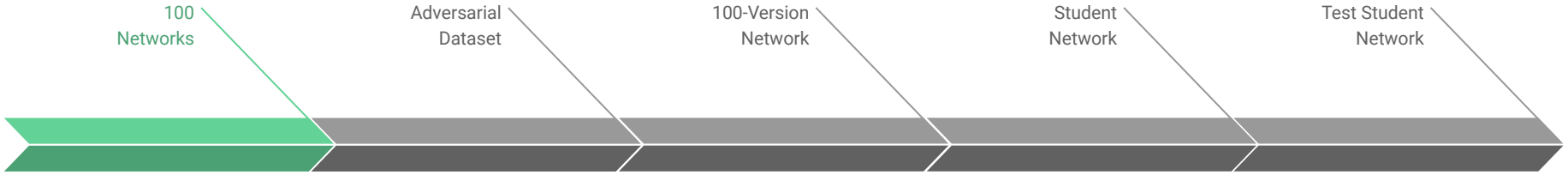
Small Neural Network: Training time couple of seconds



Student



Method Overview



Network Generation

We would generate 100 networks of different architectures.

Create an Adversarial Dataset

We generated 120 adversarial images for each of the 100 networks.

Use N-Version Programming

We would create a 100-Version program using each of the networks.

Train student network

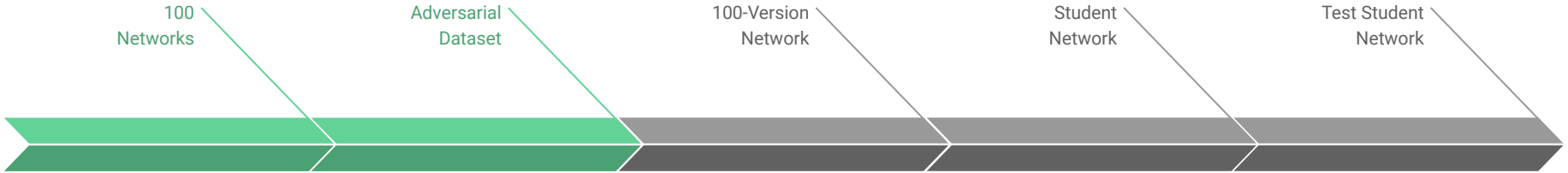
We will then train a student network to using the 100-Version network. This network we hope will be more robust than training without a teacher.

Test Student Network

We will test the student network on the original adversarial examples as well as attempt to formally verify it. We will create a baseline test network to compare against.

Neural Network 1 - Architecture A
Neural Network 2 - Architecture B
Neural Network 3 - Architecture C
...
Neural Network 100 - Architecture D

Method Overview



Network Generation

We would generate 100 networks of different architectures.

Create an Adversarial Dataset

We generated 120 adversarial images for each of the 100 networks.

Use N-Version Programming

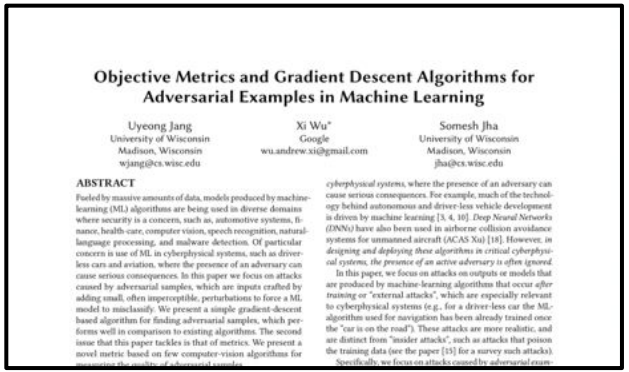
We would create a 100-Version program using each of the networks.

Train student network

We will then train a student network to using the 100-Version network. This network we hope will be more robust than training without a teacher.

Test Student Network

We will test the student network on the original adversarial examples as well as attempt to formally verify it. We will create a baseline test network to compare against.



Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning

Uyeong Jang
University of Wisconsin
Madison, Wisconsin
wjang@cs.wisc.edu

Xi Wu*
Google
wu.andrew.xi@gmail.com

Somesh Jha
University of Wisconsin
Madison, Wisconsin
jha@cs.wisc.edu

ABSTRACT

Fueled by massive amounts of data, models produced by machine-learning (ML) algorithms are being used in diverse domains where security is a concern, such as, automotive systems, finance, health-care, computer vision, speech recognition, natural-language processing, and malware detection. Of particular concern is use of ML in cyberphysical systems, such as driver-less cars and aviation, where the presence of an adversary can cause serious consequences. In this paper we focus on attacks caused by adversarial samples, which are inputs crafted by adding small, often imperceptible, perturbations to force a ML model to misclassify. We present a simple gradient-descent based algorithm for finding adversarial samples, which performs well in comparison to existing algorithms. The second issue that this paper tackles is that of metrics. We present a novel metric based on few computer-vision algorithms for

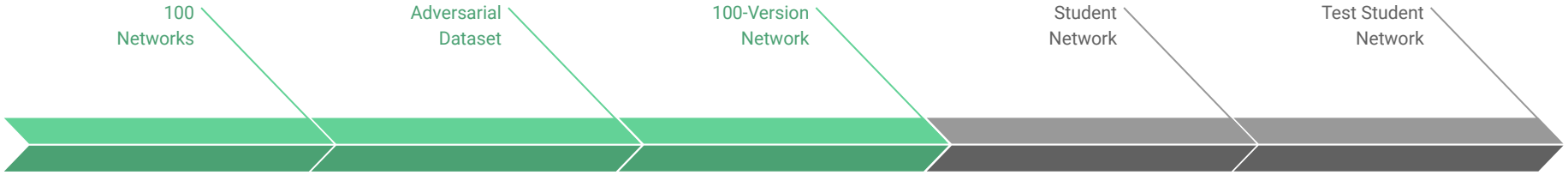
cyberphysical systems, where the presence of an adversary can cause serious consequences. For example, much of the technology behind autonomous and driver-less vehicle development is driven by machine learning [3, 4, 10]. *Deep Neural Networks (DNNs)* have also been used in airborne collision avoidance systems for unmanned aircraft (ACAS Xu) [18]. However, *in designing and deploying these algorithms in critical cyberphysical systems, the presence of an active adversary is often ignored.*

In this paper, we focus on attacks on outputs or models that are produced by machine-learning algorithms that occur *after training* or "external attacks", which are especially relevant to cyberphysical systems (e.g., for a driver-less car the ML-algorithm used for navigation has been already trained once the "car is on the road"). These attacks are more realistic, and are distinct from "insider attacks", such as attacks that poison the training data (see the paper [15] for a survey such attacks).

Specifically, we focus on attacks caused by *adversarial exam-*

Implemented 10 attacks from literature of those 9 were selected.

Method Overview



Network Generation

We would generate 100 networks of different architectures.

Create an Adversarial Dataset

We generated 120 adversarial images for each of the 100 networks.

Use N-Version Programming

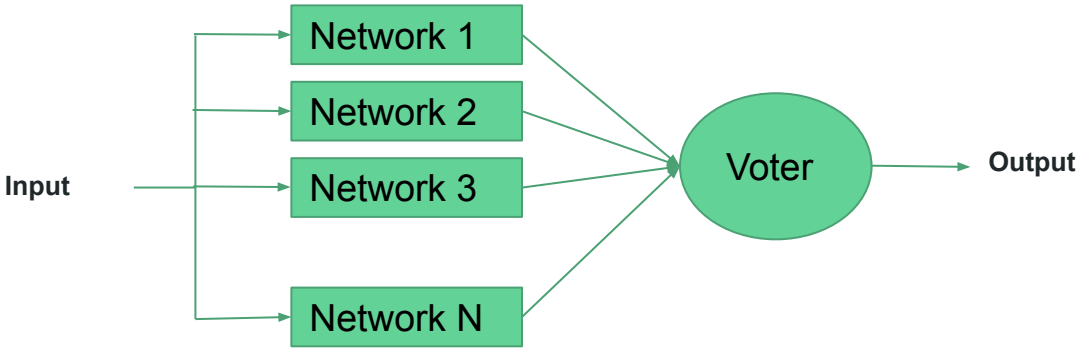
We would create a 100-Version program using each of the networks.

Train student network

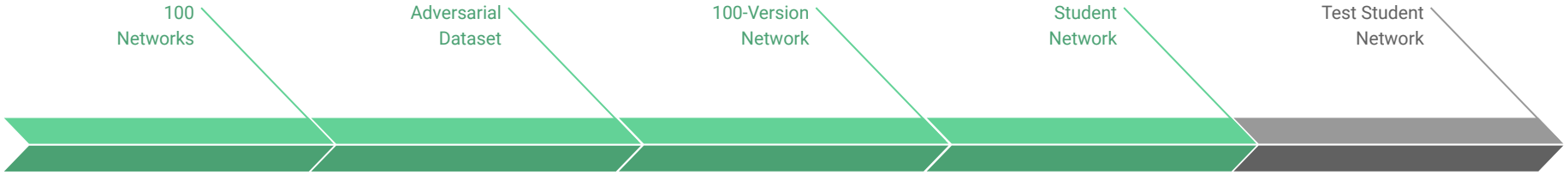
We will then train a student network to using the 100-Version network. This network we hope will be more robust than training without a teacher.

Test Student Network

We will test the student network on the original adversarial examples as well as attempt to formally verify it. We will create a baseline test network to compare against.



Method Overview



Network Generation

We would generate 100 networks of different architectures.

Create an Adversarial Dataset

We generated 120 adversarial images for each of the 100 networks.

Use N-Version Programming

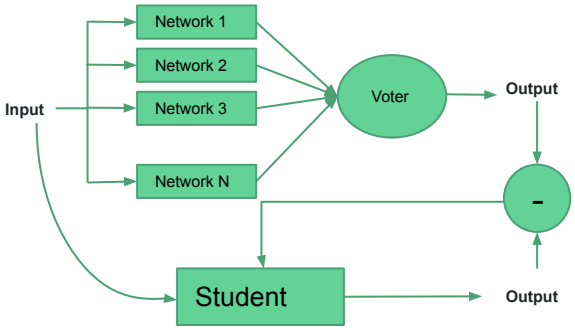
We would create a 100-Version program using each of the networks.

Train student network

We will then train a student network to using the 100-Version network. This network we hope will be more robust than training without a teacher.

Test Student Network

We will test the student network on the original adversarial examples as well as attempt to formally verify it. We will create a baseline test network to compare against.



Method Overview

100 Networks → Adversarial Dataset → 100-Version Network → Student Network → Test Student Network

Network Generation

We would generate 100 networks of different architectures.

Create an Adversarial Dataset

We generated 120 adversarial images for each of the 100 networks.

Use N-Version Programming

We would create a 100-Version program using each of the networks.

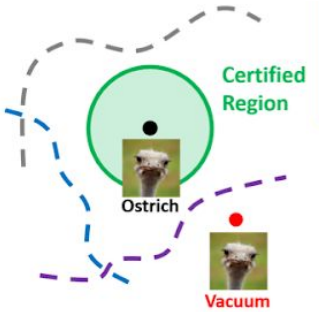
Train student network

We will then train a student network to using the 100-Version network. This network we hope will be more robust than training without a teacher.

Test Student Network

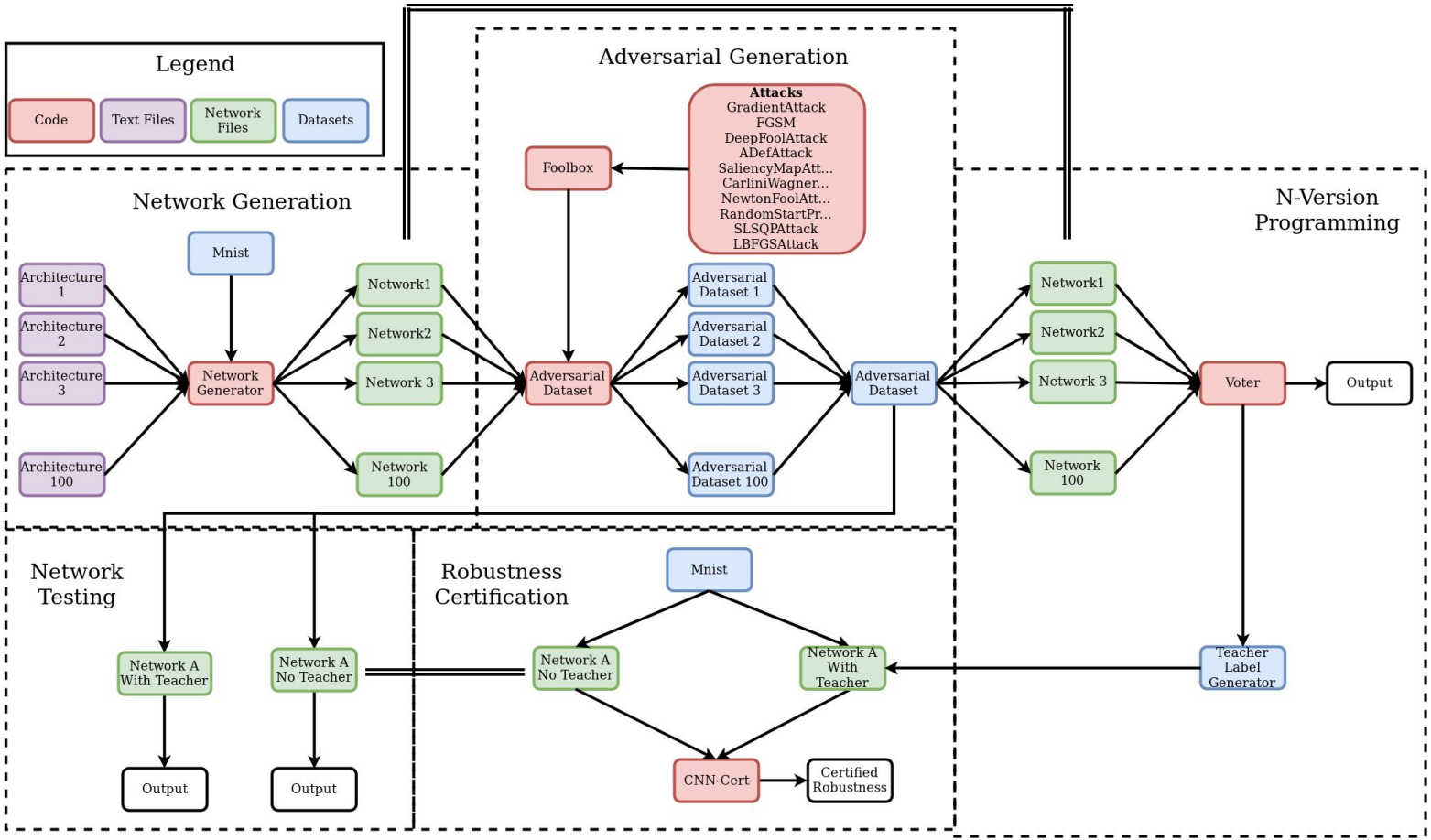
We will test the student network on the original adversarial examples as well as attempt to formally verify it. We will create a baseline test network to compare against.

IBM CNN-Cert

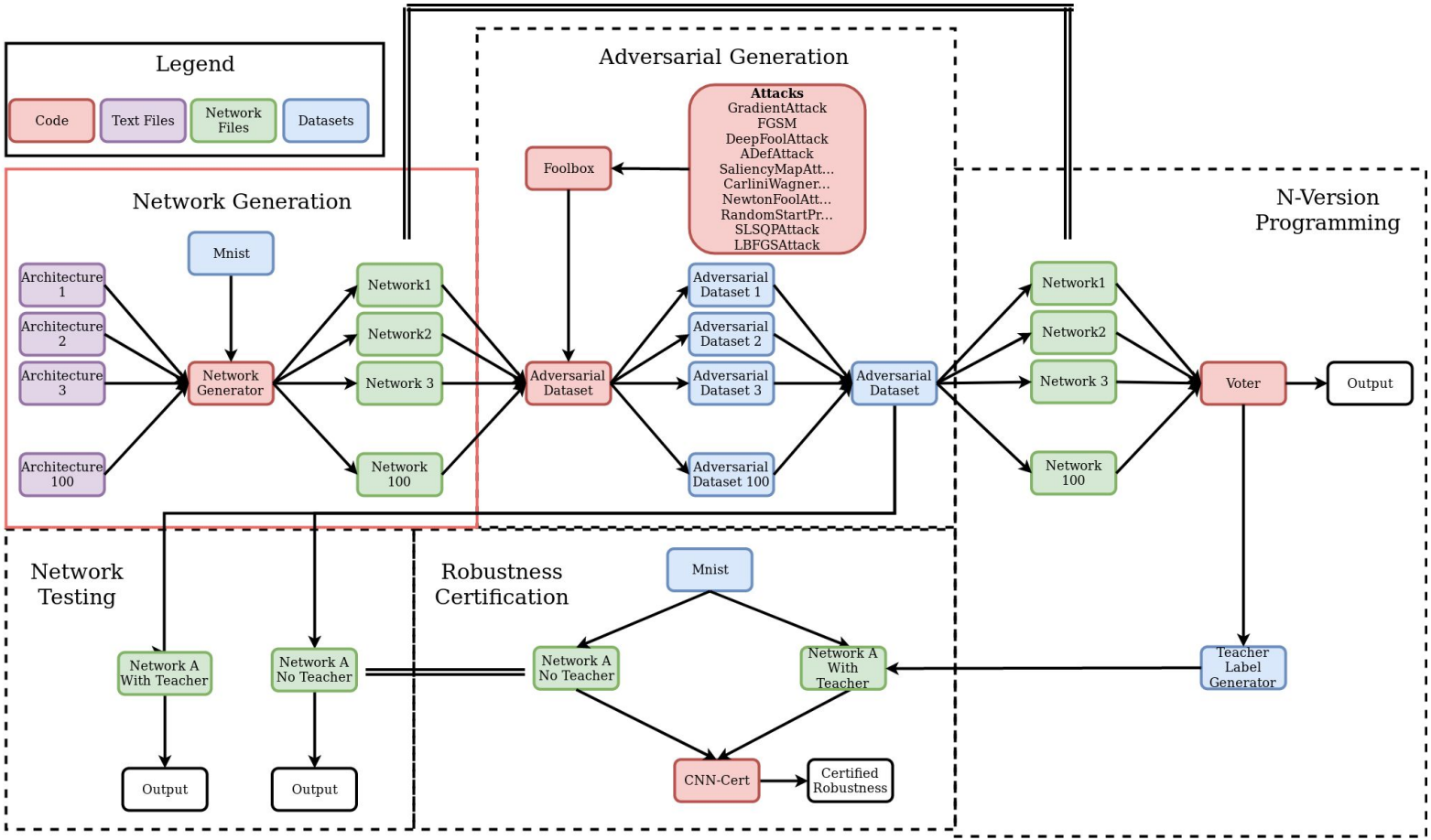


Adversarial Dataset

Implementation



Results - Network Generation

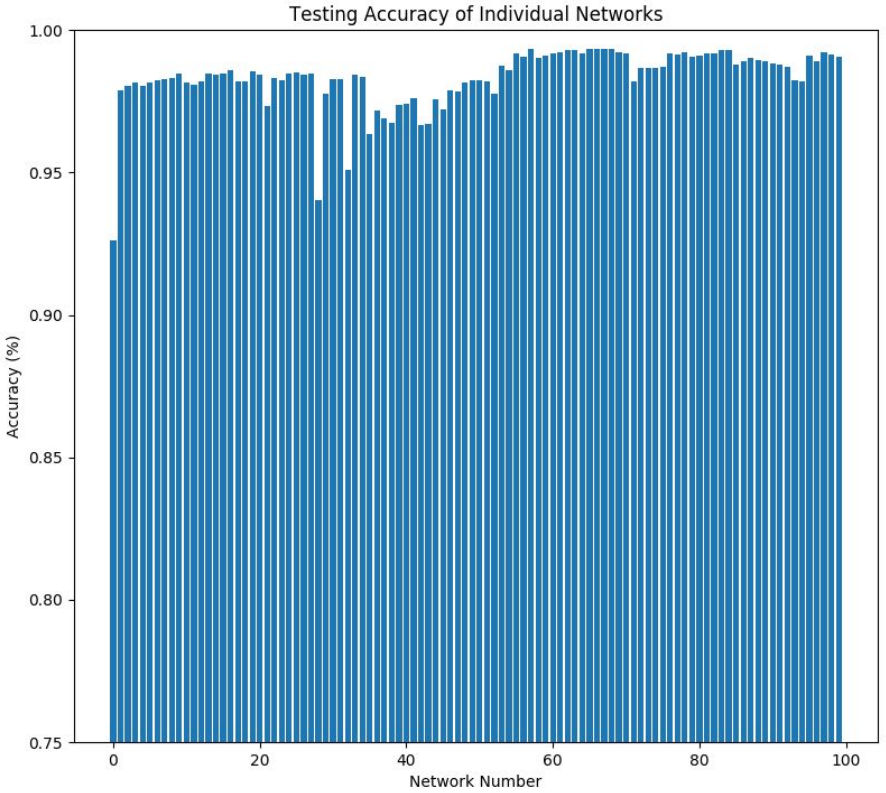


Results - Training

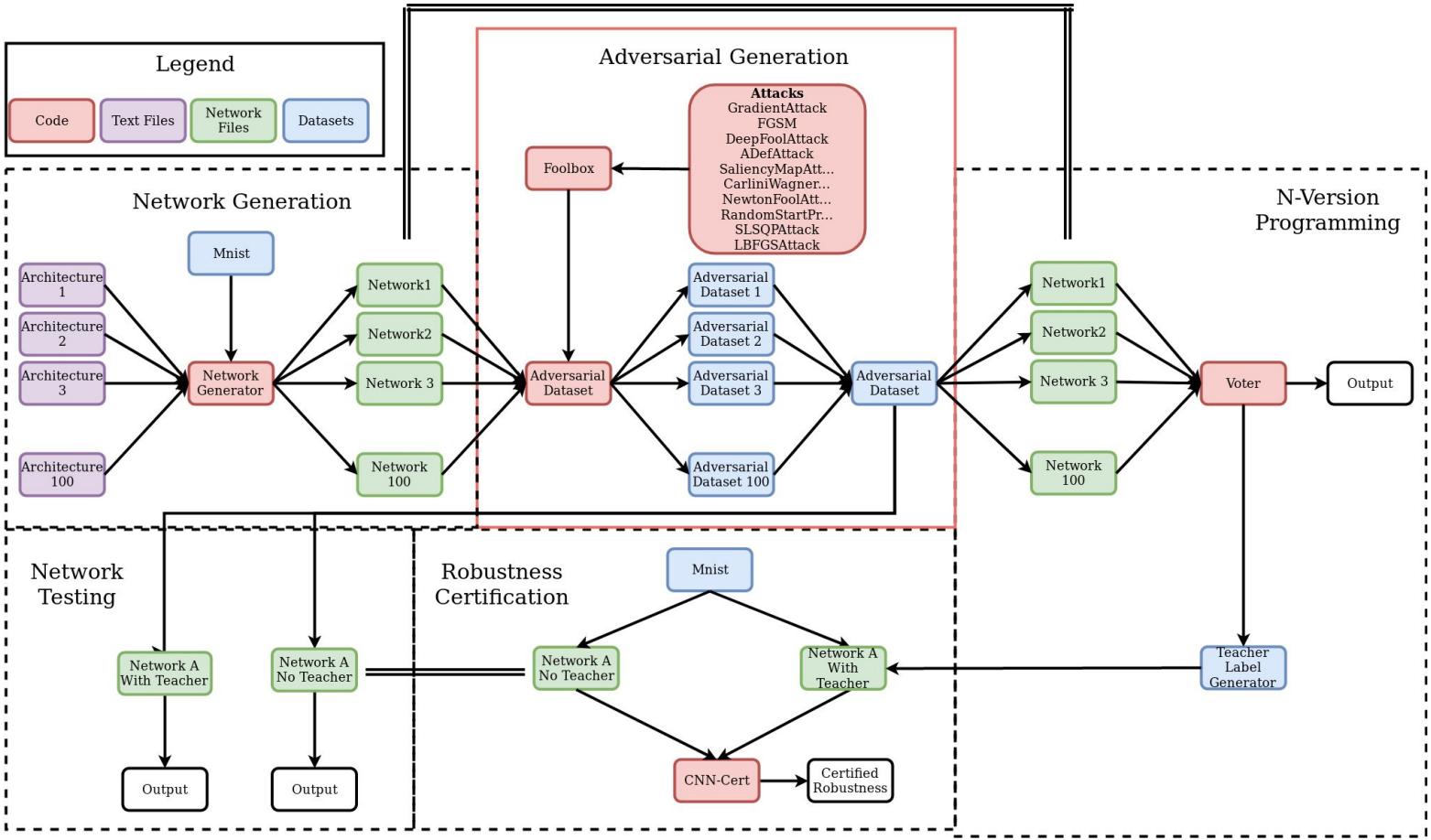
Trained 100 networks.

Networks test accuracy (92%, 99%)

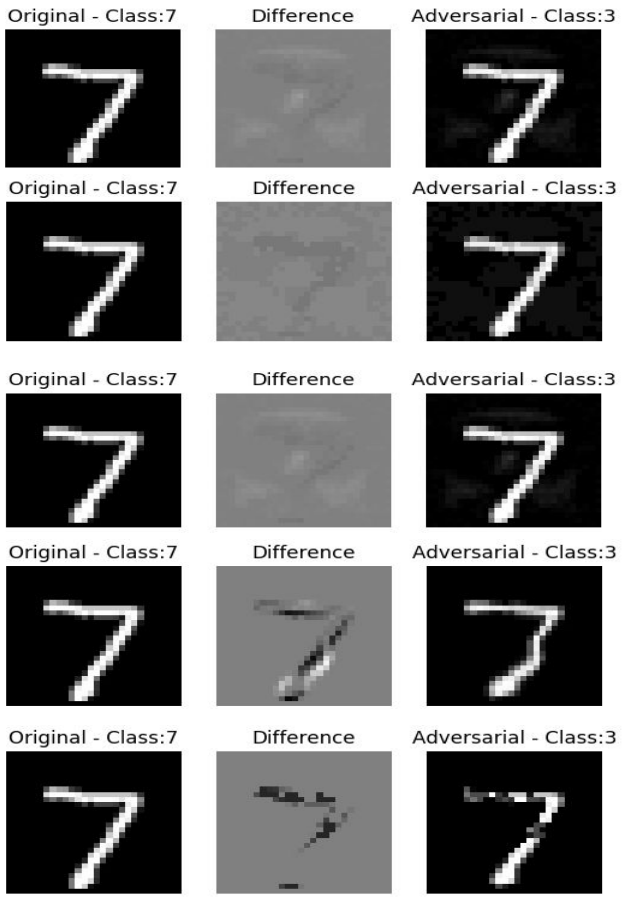
All networks were trained **successfully**

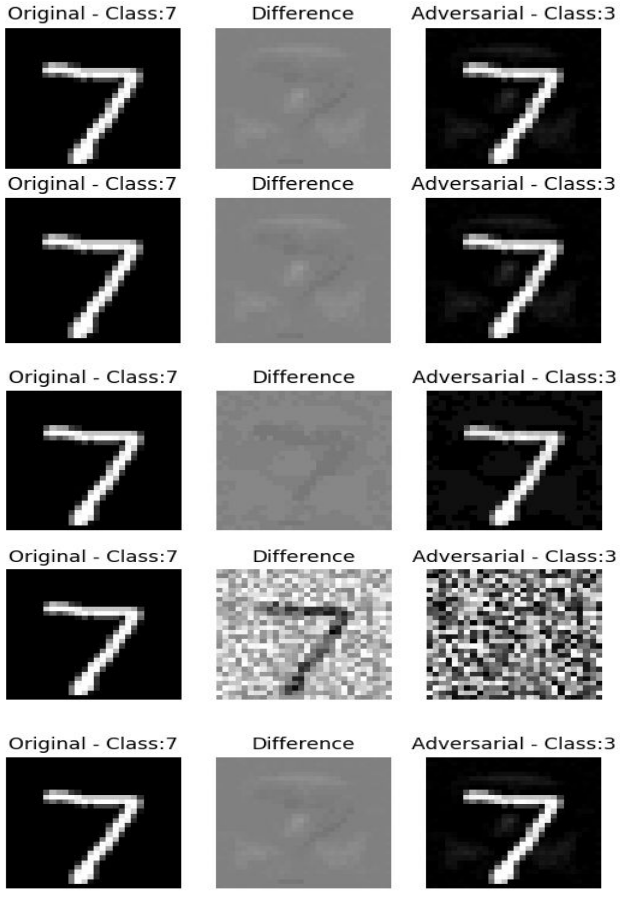


Results - Adversarial Generation

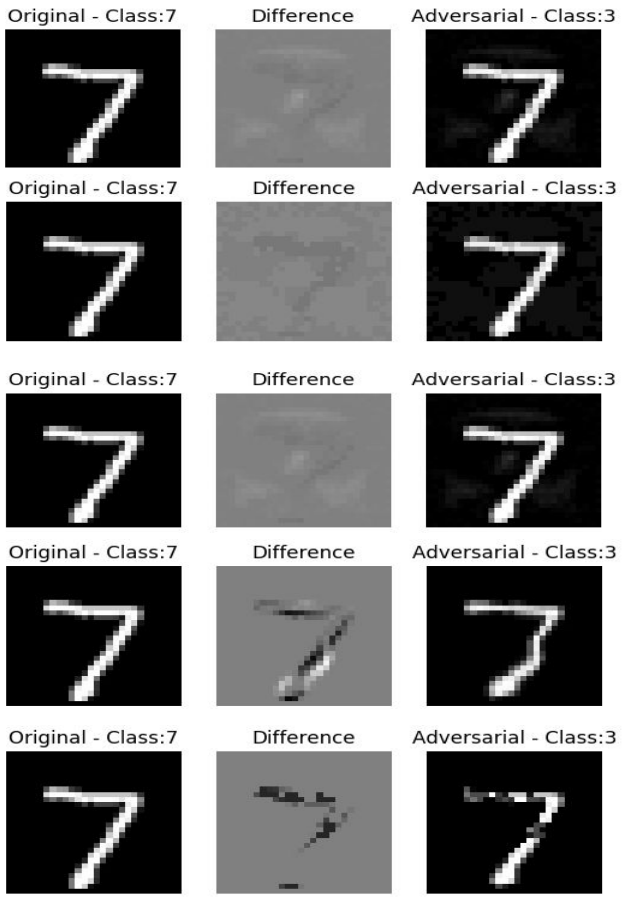


Results - Adversarial Generation





Results - Adversarial Generation

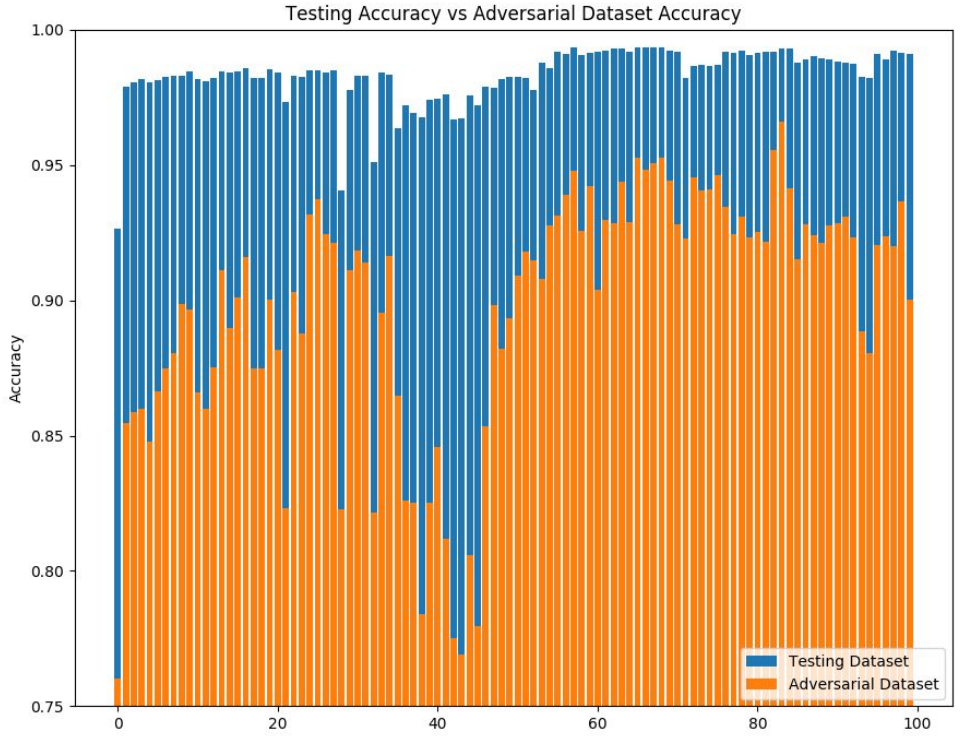


Results - Adversarial Generation

Ran adversarial attacks on each network.

Combined the results to generate an adversarial dataset.

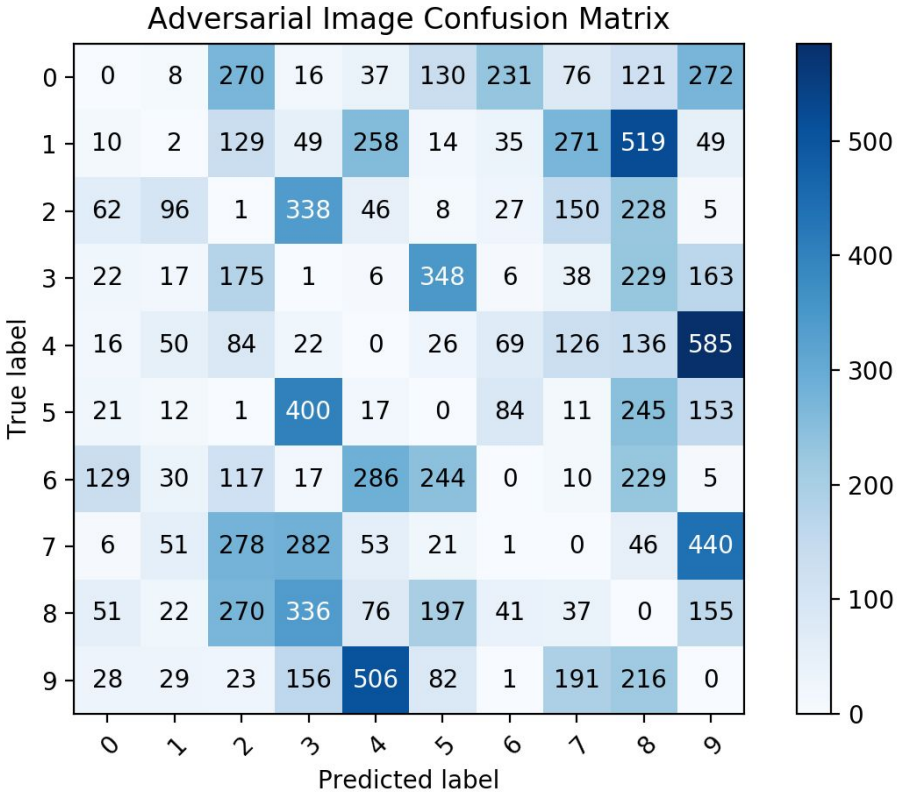
All networks accuracy **decrease** on adversarial dataset.



Results - Adversarial Generation

Which classes were the least robust?

The most common was taking the value **4** and converting it to a **9**.

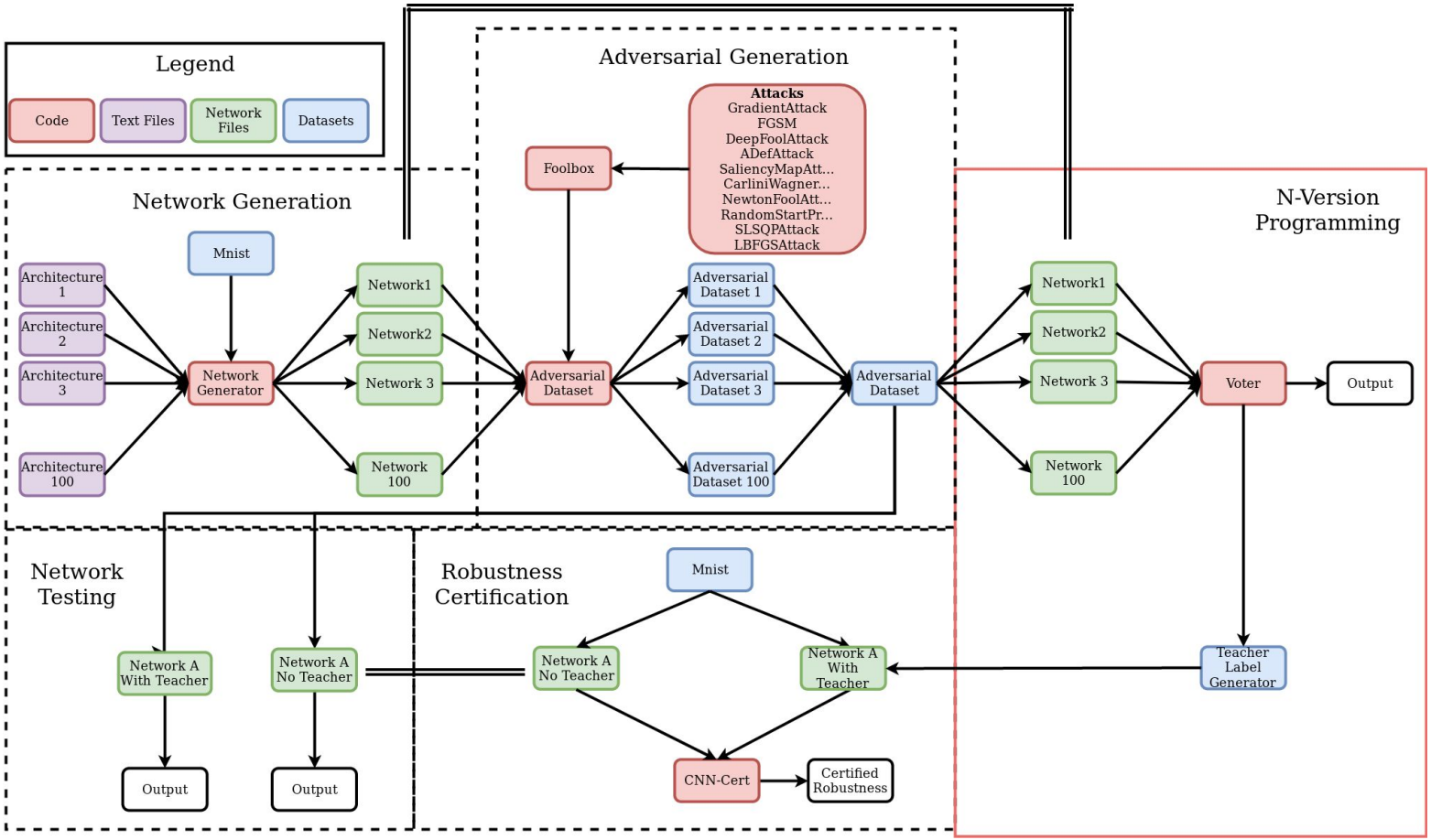


Results - Adversarial Generation

Attacks were effective 95.5% of the time

Attack	**Count**	**Effective Percentage**
Gradient	939/1300	73.23%
Gradient Sign	1272/1300	97.84%
Deep Fool	1281/1300	98.53%
Adef	1240/1300	95.34%
Saliency Map	1254/1300	96.46%
Carlini Wagner L2 Attack	1300/1300	100%
Newton Fool Attack	1297/1300	99.77%
Projected Gradient Attack	1300/1300	100%
LBFGS Attack	1300/1300	100%

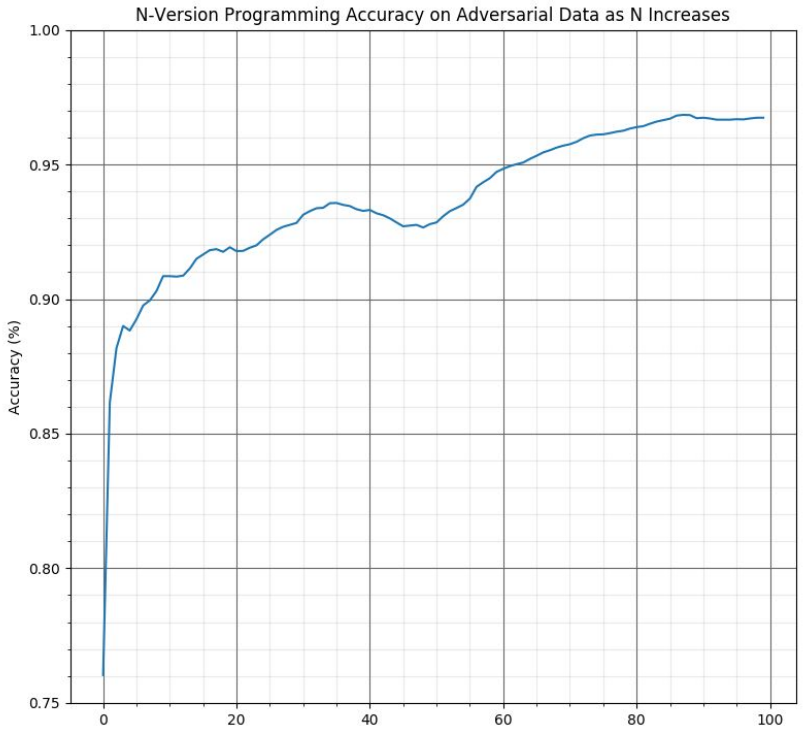
Results - N-Version Programming



Results - N-Version Programming

Increasing N increases the accuracy.

- N-Version Programming Accuracy: **96.85%**

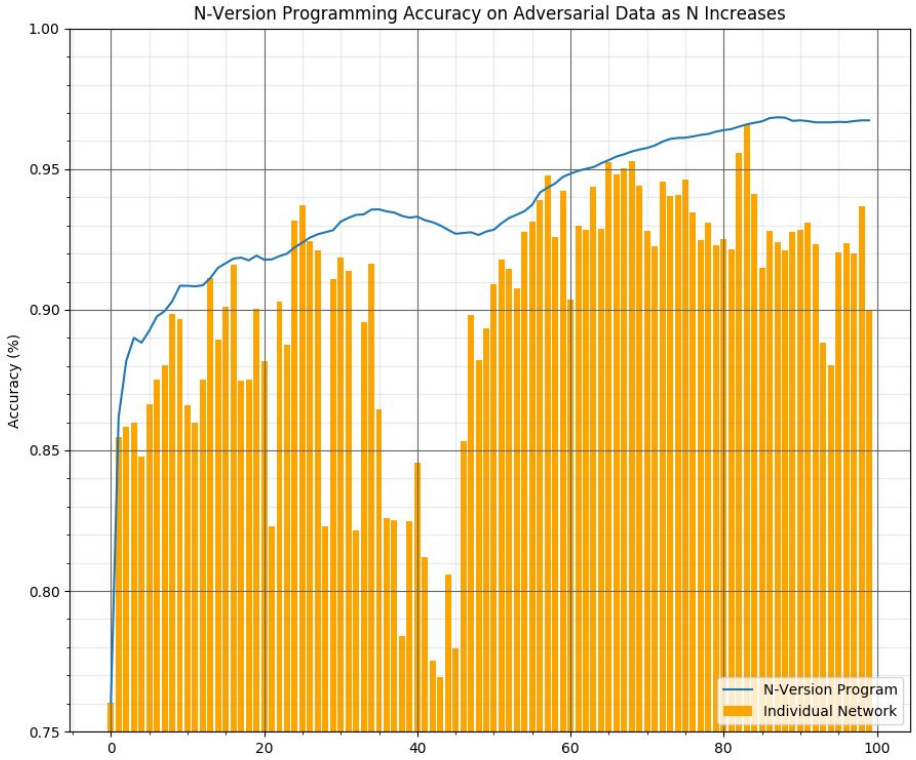


Results - N-Version Programming

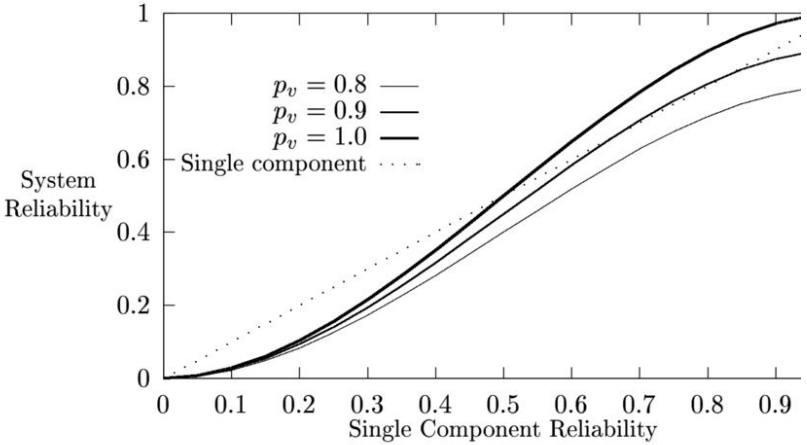
Increasing N increases the accuracy.

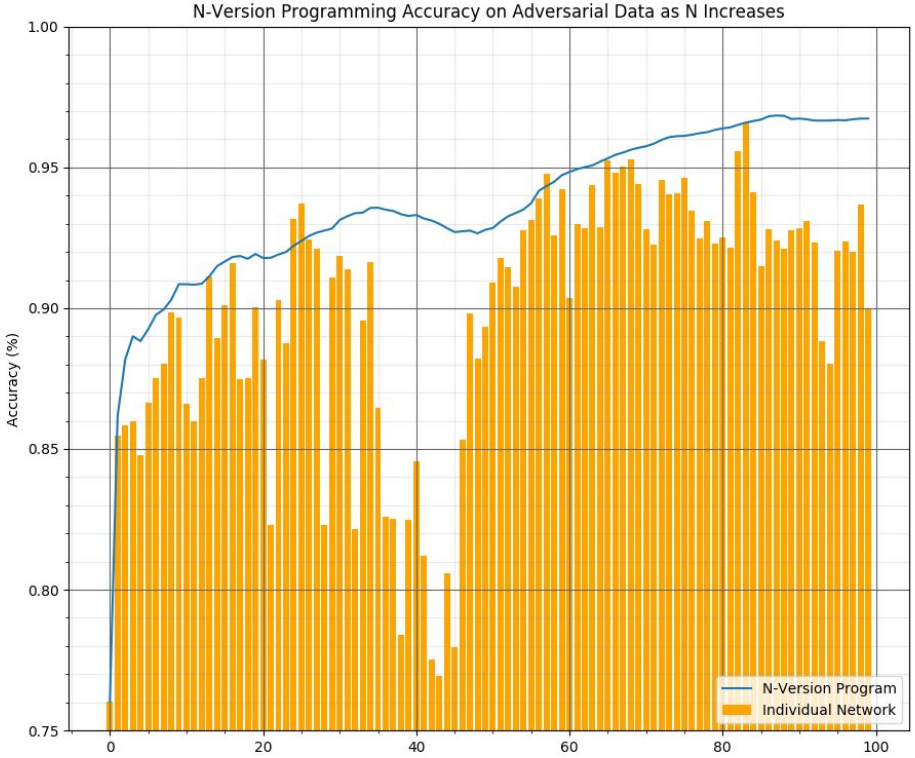
- N-Version Programming Accuracy: **96.85%**
- Individual Networks Average Accuracy: **89.77%**
- Individual Networks Best Accuracy: **96.58%**

N-Version programming does **increase** networks performance

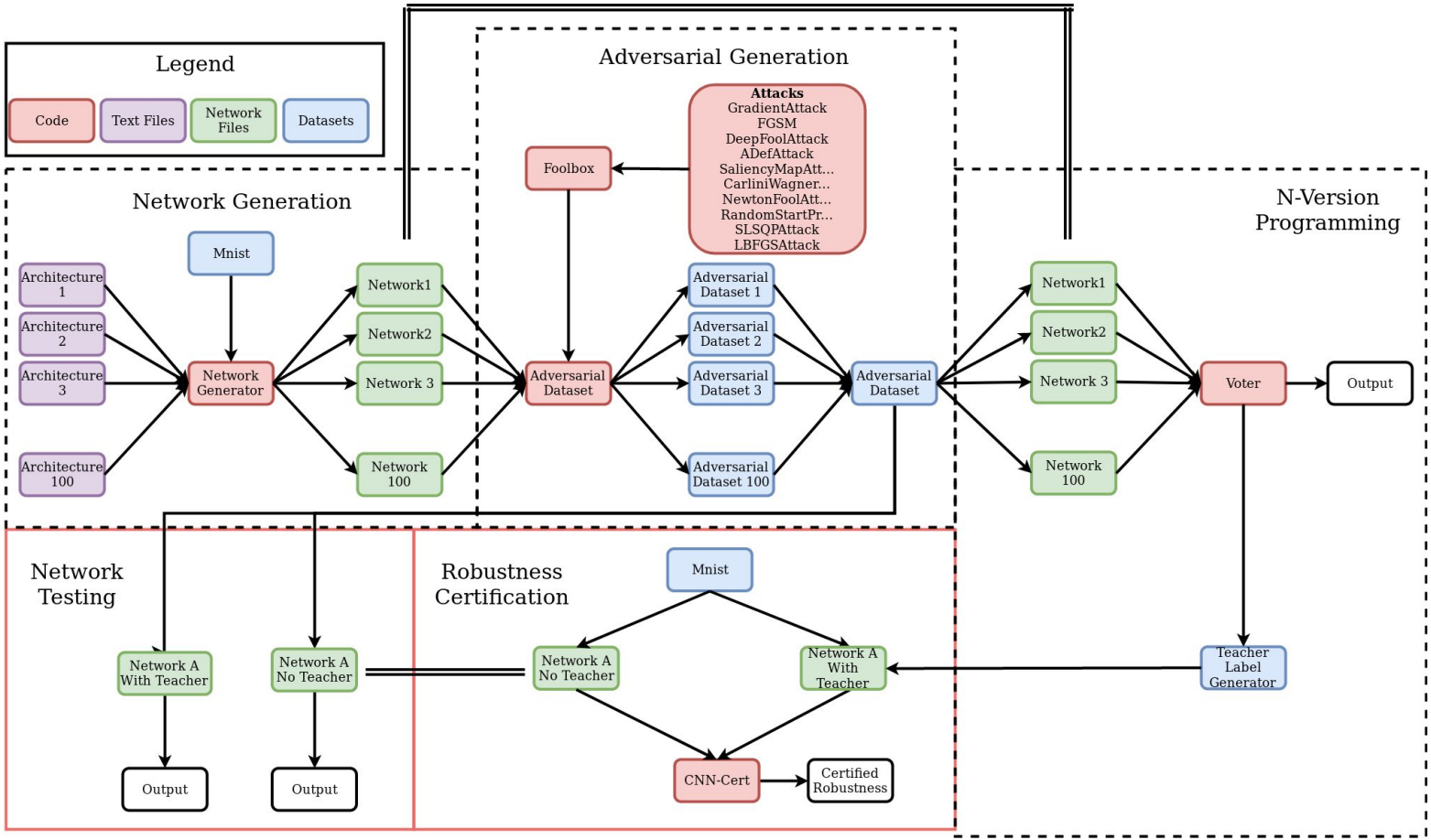


Results - N-Version Programming





Results - Testing

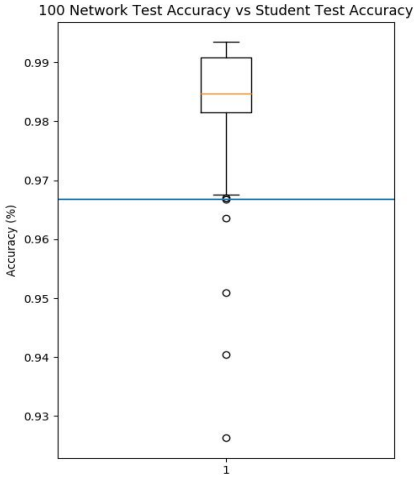


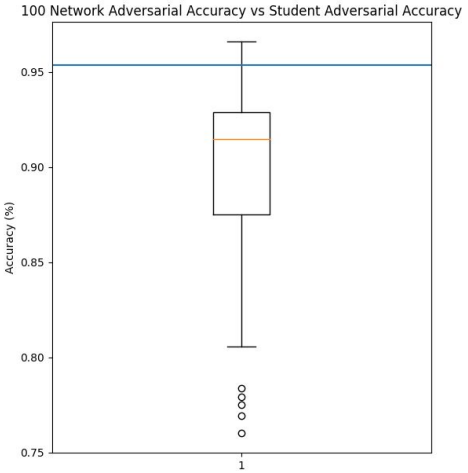
Results - Learning From N-Version Programming

Student Compared to Baseline

Metric	Baseline	Student Network
Accuracy	99.16%	98.68
L1 Norm (Formal Verification)	0.1526	0.1609
L2 Norm (Formal Verification)	0.0743	0.0809
Adversarial Dataset Accuracy	96.04	95.35
Adversarial Frequency	94.33	98%

Student Compared to 100 Networks





Reviewing our Goals:

- How robust are neural networks?
  - Not very robust, our adversarial attacks were 95% successful.
- Which adversarial attacks are most successful?
  - All are very effective.
- Are adversarial attacks defendable through N-Version programming?
  - Yes we are able to defend against these attacks using N-Version programming.
- Can we use N-Version programming to make more robust neural networks
  - Possibly, we generate a network who was the 4th best against adversarial data using our technique.

Questions

Demo