University of Central Florida

Defeating Fingerprint Scanners Using Antagonistic Neural Networks

Sponsored by Dr. Sumit Kumar Jha
Assisted by Sunny Raj and Dwaipayan Chakraborty

Serra Abak, Wyatt Waterfield, Emily Pelchat, Patrick Maier
COP 4934 Senior Design 1, Group 18
Dr. Mark Heinrich
April 25, 2018

Table of Contents

1. Executive Summary

In consumer devices, fingerprint authentication has quickly become one of the predominant methods for securing one's information. With the rise of the use of fingerprint authentication for simply unlocking your devices, many companies have also allowed their users to sign into their services using their fingerprint as well. Since this form of biometric security is increasing, it is imperative to explore its possible vulnerabilities.

This project, conducted by Serra Abak, Wyatt Waterfield, Emily Pelchat, and Patrick Maier, sponsored by Dr. Sumit Jha, aims to explore an increasingly effective vulnerability in this form of biometric authentication. In this project, we explored the use of generative adversarial networks to create adversarial fingerprints. This subfield of adversarial machine learning is gaining serious momentum as machine learning-based algorithms are more commonly implemented. These algorithms train based on data it has seen before in order to make future predictions. Since these algorithms are often used in constantly changing, adaptive environments, adversarial examples can be specifically crafted to exploit various vulnerabilities. In the case of this project, we plan to use new developments in generative adversarial networks to recreate images of fingerprints that when applied to an authentication system, the system will believe with a substantial amount of confidence that the generated fingerprint is an authorized user.

Our project aims to drive attention to this particular vulnerability, and show just how effective it can be in various circumstances. To accomplish this, we will make a holistic product consisting of a web interface, which a user can upload a fingerprint dataset and retrieve a generated, seemingly authorized fingerprint, and the backend portion which will use the latest research around generative adversarial networks to create these adversarial fingerprints. In order to be completely comprehensive, we will also develop our own authentication neural network which we will use to test the rates of authentication. This will be used to show how effective adversarial examples can be against the various acceptance rates in fingerprint readers.

At a high-level, this method of creating adversarial examples to exploit various machine learning based algorithms can be applied to a wide range of technologies. This makes it critical that these vulnerabilities are explored and techniques to implement these algorithms in a safe manner are thoroughly evaluated.

2. Project Background

2.1 Objectives

Many modern fingerprint systems use machine learning to learn features from users for recognition. In recent time it has been found that such system may be robust in ways humans cannot match but still fail on minor changes that wouldn't bother any human. This is because the machines learn from a very limited feature set from the feature space due to processing constraints.

The problem we have been given is to implement or utilize a fingerprint authentication system and attack it with fingerprints to learn how to build a "master key" that will work for the majority of users in the system. We have been asked to do this by building an antagonistic neural network to analyze the security system and learn how to break it.

Our goal is to create an adversarial machine learning system that can generate a set of master fingerprint keys with a given data by finding which features (called minutiae) are important to authenticating with the fingerprint authentication systems in our mobile phones and other devices. These master fingerprint keys will be used to unlock devices that don't belong to the owner of the master fingerprint key, essentially undermining the security of all devices that use biometrics. As long as the master fingerprint matches enough users, since multiple master fingerprint keys can be produced, we will prove that the system of static biometric data is insecure.

2.2 Motivations

Serra Abak

My motivations in regards to working on this project are mainly in combining the new, exciting developments in deep learning with cybersecurity, which has been gaining traction and importance in the world. I work in cybersecurity and wish to pursue a career in security engineering, so this project perfectly represents my goals in life. I believe that machine learning is the future of technology, and cybersecurity is one of the most important entities in development to apply machine learning. Cybersecurity is one of the most arduous fields in Computer Science, because it only takes one vulnerability to bring down a system. Systems have to be monitored 24/7 to keep fully available and accessible. Machine learning can revolutionize cybersecurity in vulnerability research, networking, fuzzing, and more.

Two years ago, I got the chance to attend DefCon 24, which was the year the DARPA Cyber Grand Challenge took place. I got to personally experience six systems programmed by six teams attack each other and defend themselves completely independent from any human administrator. Attending this event and getting to talk to one of the teams that competed, Shellphish, really inspired to me look into development in security, rather than concentrating on just vulnerability research. Building software that can automatically defend and attack is a completely new field. This project is the definition of cutting-edge technology, and any aspiring researched would want to be involved in this kind of research with practical application. Using these new developments and combining it with neural networks could lead to drastic improvements in finding vulnerabilities and exploits much faster than malicious hackers can find and exploit them.

Because I have no personal experience with deep learning, I was very excited about this project. It's difficult to find a small, personal project to do on my own time that involves deep learning, because it's a very involved topic. I knew that this was my chance to gain experiences that I couldn't otherwise gain easily. I have also never worked on a large project in Python, and when TensorFlow was mentioned, I presumed that this may be my chance to get a handle on Python. Python is one of the leading languages for the industry, so it's one of my top priorities to be experienced in this language.

Having the guidance of a professor and two Ph.D. students were one of the main reasons I wanted to be involved in this project. I felt like I could learn a lot through people who have dedicated their careers to understanding and creating adversarial machine learning systems. In addition to this, the project will give me that extra bit of motivation to research adversarial machine learning--something I would not have done on my own. With both more academic knowledge in cybersecurity and more practical experience in deep learning, I can aim for a niche career path where I am in the middle of innovation and technological advancement.

Patrick Maier

My motivations for doing this project are to understand what neural networks are and to learn what types of problems are more efficiently solved through their use. "Machine Learning" and "Neural Network" are huge buzzwords in tech news, but also, within the industry, it doesn't seem as though many computer scientists know what the implications of machine learning are, or when/how neural networks are appropriately implemented. This project is an opportunity for these concepts to be demystified. Working with experts in this realm, namely, Dr. Sumit Kumar Jha, Sunny Raj, and Dwaipayan Chakraborty, and conducting my own research will hopefully leave me with a clear idea of

these concepts and their potential. This may even become an area of specialization that I would like to pursue further in Graduate School and my career.

The fact that we will be working with adversarial machine learning techniques is particularly interesting to me. With the modern proliferation of technology and information systems, cybersecurity and the integrity of software are incredibly important. It is becoming increasingly unrealistic to outsource security needs. We have reached a point in which software must be developed with vulnerabilities in mind, and the individual software engineer must understand not only how their systems fundamentally work, but also how others will be trying to exploit flaws in their design. This project will demonstrate vulnerabilities in recognition systems and light the fire for researchers and developers to confront these risks. The project will also allow me to become more aware of these issues, and thus, in theory, be better prepared to create software that isn't as easily fooled.

My interest in this project also stemmed from it appearing to be one of the most challenging options. The problem posed by the requirements document did not lend itself to a straightforward solution, and it borders on being a research project as opposed to a typical design project. Having the opportunity to dive into a complex field of computer science while receiving skilled mentorship was something that I could not easily pass up. I see this as a way to test the problem-solving skills and technical abilities that I have been developing throughout my time at UCF. This will better showcase my skills to potential employers than a simple database-backed website with a mobile app. More than that, I know that I will be able to look back with pride on what I am able to accomplish with this team and what obstacles I am able to surmount as an individual.

Emily Pelchat

My motivations in this project are in the neural networks and how to build and use them. At my work, I am being brought into a project using neural networks to find humans hidden in images, as well as determine whether they are in a form of cover that would stop a bullet. I expect that using adversarial neural networks to train a network that takes what few human features we can see can generate a system that can find where the whole human is.

My motivations also branch over into video game AI design. Currently, a lot of video game AI designs do little to no prediction for the player's actions, or they use perfect knowledge of the game environment to avoid having to implement a difficult reconnaissance algorithm. I see an antagonistic neural network as a method of allowing the computer to perform educated guesswork that can improve over time based on the player's actions.

Even more generally, if normal neural networks are a logical thought process, I see GANs (Generative Adversarial Networks) as an imaginative thought process. Rather than just computing data, now, the computer is creating data that you believe to be correct, or envisioning a solution to a problem, that we can't fully formulate. It is a process for finding the answers to questions, when we might not even know the question itself.

Using fingerprints as security measures is a universally bad idea for a number of reason as well. First off, your finger prints aren't really private. If you've ever thrown away a piece of paper you've touched, you've released your fingerprint into the public. As such it is very hard to keep them a secret. Second, if your security ever gets compromised, you need to be able to change your authorization method, like your password. Finger prints cannot be changed though, so now you have a problem that cannot be solved without some other method involved. Finally as we are going to show, it is impossible to make an automated system with our hardware to accurately determine if a finger print is your finger print. As such the closest we can get is a best estimate.

Wyatt Waterfield

My motivations for pursuing this particular project deal largely with the tech stack (Neural Networks) that is used to implement both our fingerprint authentication system and the feature which creates adversarial fingerprints. I originally became interested in this field when I was exposed to possible ethical issues with developing artificial intelligence, which then led into my research of the necessary precautions that AI developers need to take when making improvements to such technologies.

As the tech industry continues to make advancements in artificial intelligence and implement technologies such as machine learning and neural networks to make important decisions, the designers of these systems need to take responsibility for creating the necessary protections of these systems. I'm interested in this particular aspect of the field because I do believe that AI and learning based models will continue to provide substantial benefits to the tech industry and help solve some of the hardest problems that we may encounter.

However, if these systems are not protected and their weak points are not recognized, it could have serious consequences. Significant decisions such as those involved with driving, providing news content to users, or in our case, determining whether a certain set of fingerprints has the required authorization, are increasingly being made by trained systems. In choosing to pursue this project, I hope to gain insight into how AI based decisions like these could be exploited.

Along with the acquired insight into adversarial machine learning that comes along with this project, I also chose this project largely because it deals with several new technologies that I have not yet had the chance to explore on my own. I've gone through the process of creating a database, web application, and developing a backend to support it all in my other classes as well as my past internships. Though I do enjoy these software related tasks and I'm glad I get to partake in their implementation in this project as well, this more research-oriented project will allow me to explore various avenues of computer science which I have not been exposed to prior. Since this project involves quite a bit of research and conceptually learning several new topics, I hope that it will give me further insight into an area I could possible specialize in the future. Hopefully, this project will either further encourage my interest in AI systems or direct me down another path I could potentially investigate.

2.3 Broader Impacts

The system that we will create will demonstrate the weaknesses in fingerprint authentication systems, which are one of the most widely used authentication systems on mobile phones today. This type of authentication is not only used for simply unlocking the device, it is also used for signing into individual applications and services, as well as providing a method for two-factor authentication. Because our system will be lightweight and easily accessible, users will be able to test their own data with our system.

The ease of use that we are aiming for in this project is a key factor in its eventual, broader impacts. If the developers who implement this method of authentication and the consumers who use it regularly realize the ease of creating these adversarial fingerprints, it will shed some light on the vulnerabilities they face. Since many users heavily depend on this form of authentication for protecting their digital information, this hands-on approach to security will make a greater impression on customers.

This research may also become a part of the push toward better security by forcing tech companies to enforce their devices with better authentication systems. With the constant discussion of data privacy in the public discourse and the increased awareness of the potential consequences of data breaches, it would become imperative for companies to ensure the security of their consumers data. Through a combination of external attacks, or internal leaks, it becomes a matter of if not when the data regarding user information will be revealed.

Pursuing this project will also create a method for which security engineers could argue for more secure measures to be put into place in their respective companies by being able to demonstrate the ease in which the fingerprint authentication system may be attacked. Once the technical information is placed in the background and the simplicity of use is brought front and center, it will be truly apparent to everyone who views the site how someone with minimal technical

knowledge can create adversarial fingerprints. This will especially benefit those with little understanding of new technology, such as young kids, older citizens, and the poor, as these groups are often the most vulnerable to cyber-attacks.

3. Project Requirements

3.1 Base

3.1.1 Research

The team will study 15 or more research papers or academic articles on fingerprint authentication, adversarial machine learning, deep learning, and/or security in iOS and Android OS. The research topics are as follows:

- Databases for fingerprint matrices
- Image processing or converting to matrices
- Neural Network approaches
- Adversarial network approaches
- TensorFlow uses for Neural Networks
- Website Frontend/Backend

3.1.2 Fingerprint Authentication System

The team will create a fingerprint authentication system in Python similar to iOS and/or Android OS using TensorFlow. The authentication software must be able to perform the following actions:

- Pull fingerprints for a user from Database
- Recognize if a fingerprint matches a user's fingerprints in the database
- Guarantee device access with a 90% (± 5%) success rate on true authentication inputs

3.1.3 Generative Adversarial Network

The team will create an adversarial machine learning system in Python that is compatible with a web interface using TensorFlow. The adversarial network software must be able to perform the following actions:

- Pull fingerprints for all users from the Database
- Take common features from the fingerprints and create a master key set of prints to fool the recognition system
- Create fingerprint matrix to unlock at the fingerprint authentication system with a 50% (± 5%) success rate

3.1.4 Web Interface

The team will create a web interface with a user-friendly UI in JavaScript to feed in data and to receive data from the adversarial machine learning system. The web interface must be able to perform the following actions:

- Enter fingerprints via Web UI
- View fingerprints entered for a user in the Web UI
- Pull skeleton key prints from database in Web UI

3.1.5 Database

The team will set up a database in JDBC or Python DB API that stores the fingerprints in matrix format. The database must be able to perform the following actions:

- Needs to be able to provide all fingerprints (up to 10,000 fingerprints) in the database on request
- Needs to store master prints separately from user prints
- Needs to provide master prints by request in at most 5 minutes

3.1.6 Testing

The team will create:

- 15 or more test cases to test the fingerprint authentication system
- 3 tests to stress test different kinds of loads on the web interface

3.2 Stretch Goals

3.2.1 3D Printing

3D print master keys and successfully attack the mobile fingerprinting authentication system using the 3D-printed fingerprint. The synthetic fingerprint must be able to achieve the following specifications:

- Resemble an accurate representation of the master fingerprint
- Be recognized by fingerprint scanners

3.2.2 Commercial Hardware Penetration Testing

Implement a modified adversarial machine learning system that attacks a mobile fingerprinting authentication that is currently in use. The adversarial network system must be able to perform the following actions:

- Attack provided fingerprint sensor authentication system with a 50%(± 5%) success rate
- Authenticate with 3D-printed master fingerprint

4. Research and Investigations

4.1 Comparing Keras vs. TensorFlow

Keras and TensorFlow are the two most popular libraries in the tech world due to their robustness, flexibility, and large community. TensorFlow was originally developed by Google's GoogleBrain division but now continues to be developed by open-source contributors on the Source Code Management site, GitHub [1]. Keras is a library that was built on top of TensorFlow to provide developers an easier alternative to TensorFlow for more rapid prototyping and less code [2]. Keras provides many new methods in its high-level API that makes building networks easier.

Keras provides two key API calls that cut down on unnecessary code. The *Model* API encapsulates all input and output tensors and neatly packs all arguments with all the configurations--optimizer, loss, metrics, loss_weights, sample_weight_mode, weighted_metrics, and target tensors-- into one class [3]. The class features all the necessary methods--compile, train, predict, evaluate, and other optional methods. Additionally, the *Sequential* API handles the layers related to deep learning in one class, encapsulating all the layer convolutions. This ensures that the code is easy to read and all resources relating to writing a neural network are easy to find.

TensorFlow is less friendly to beginners. Many data set inputs, layer convolutions, training, prediction, training, and estimation are done manually. While TensorFlow does provide high-level API calls that encapsulate most functions under the Estimator and the Datasets API, most calls are unique to the function and require more research. However, because Keras is designed to hide its complexity, it lacks the greater control TensorFlow allows its users. TensorFlow is able to handle threading, queueing, debugging, and defining every bit of the operations on the neural network, while Keras falls short of this [3]. Additionally, TensorFlow has a larger user base with more threads, topics, and most importantly, solutions to questions.

GANs fall under the high complexity category of deep neural networks. They have only been in the research phase for the last four years, and these systems require a lot of tweaking to get working with low levels of loss. While Keras could help us get the network up and running faster, it will be easier to tweak and troubleshoot networks in TensorFlow due to its greater flexibility and more numerous user resources.

4.2 Using TensorFlow to Create Neural Networks

TensorFlow is a tool used to create deep neural network models or train existing models with new sets of data. It's supported on Windows 7 or later, MacOS X 10.11 or later, and Ubuntu 14.04 or later. There are two toolsets available--TensorFlow

with CPU support only and TensorFlow with GPU support. Because training machine learning models requires a high number of cycles, TensorFlow.org suggests installing TensorFlow with GPU support. The required dependencies for this option are CUDA® Toolkit 9.0, NVIDIA drivers associated with CUDA Toolkit 9.0, cuDNN v7.0, and an NVIDIA GPU card with CUDA Compute Capability 3.0 or higher. There are two mechanisms by which you can install TensorFlow. The first mechanism is the “native” pip, the official Python package manager, which installs TensorFlow directly to your system. The second mechanism is Anaconda, which creates a virtual environment to install and run TensorFlow.

To run from a GPU, within the code, the developer will specify with *tf.device("/gpu:0"):* before executing the code. If there is more than one GPU on your machine, 0 will be assigned to the GPU the computer uses, and the additional GPUs will be assigned to other numbers in ascending order. To run the program with multiple GPUs, a *for* command can be used to split the load [4]. For example:

```
for d in ['/device:GPU:2', '/device:GPU:3']:
  with tf.device(d):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0],shape=[3, 2])
    c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
  sum = tf.add_n(c)
```

Figure 4.2.1 Example Multiple GPU usage

As Figure 4.2.2 demonstrates, TensorFlow uses Estimators, an API that “provides methods to train the model, to judge the model's accuracy, and to generate predictions,” and Datasets, an API that provides methods to “load and manipulate data, and feed it into your model” [5]. The low-level API methods of the tool are mainly written in Python, so it supports TensorFlow project development in this language the best.

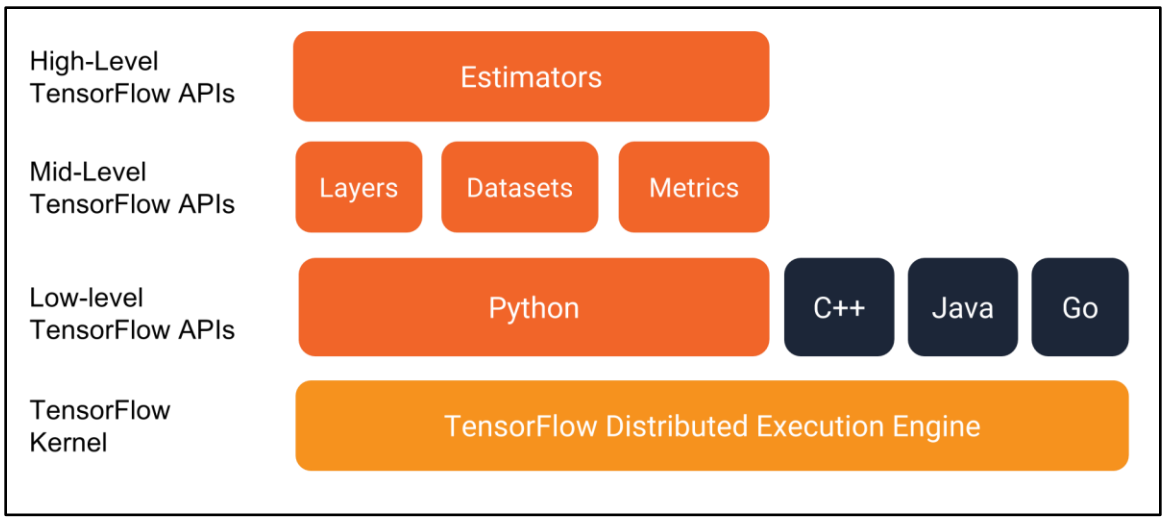


Figure 4.2.2 TensorFlow Programming Environment [5]

Estimators encapsulate the initialization, logging, saving, and prediction of a model. There are two options to choose from when starting a new project: using a pre-made Estimator or creating your own Estimator. To use a pre-made Estimator, the developer will "create one or more input functions, define the model's feature column, instantiate an Estimator, specifying the feature columns and various hyperparameters and call one or more methods on the Estimator object, passing the appropriate input function as the source of the data" [5].

```
def input_evaluation_set():
    features = {'SepalLength': np.array([6.4, 5.0]),
                'SepalWidth':  np.array([2.8, 2.3]),
                'PetalLength': np.array([5.6, 3.3]),
                'PetalWidth':  np.array([2.2, 1.0])}
    labels = np.array([2, 1])
    return features, labels
```

Figure 4.2.3 TensorFlow Input Function [5]

To create an input function, a two-element tuple containing a Python dictionary of *features,* with names and values, and a *label* array. The *label* is the outcome from a set of features. These datasets can be created manually or organized with the TensorFlow Dataset API. Figure 4.2.3 demonstrates how the function can be set-up manually. The feature column is the method by which TensorFlow accepts feature names as numerical values for use in its algorithms. If the features are already numerical, this section will act as a *tf.placeholder*. Next, the Estimator class will be instantiated. There are many pre-made classifiers, each that handle different uses better, including:

- tf.estimator.DNNClassifier—for deep models that perform multi-class classification [1]
- tf.estimator.DNNLinearCombinedClassifier—for wide-n-deep models [5]
- tf.estimator.LinearClassifier— for classifiers based on linear models [5]

Each Estimator has its own *train* class. This is where the maximum number of steps while training taken is defined. The deep learning system will train for this many iterations. The *evaluate* class can then be used to return the accuracy compared to test data with a number between 0 and 1. With the *predict* class, the system will return a dictionary with a prediction result from the data. These methods round out the machine learning approach by creating an algorithm based on data to answer a "question" with a predefined "answer".

TensorFlow can also handle neural networks and deep learning with its high-level API. To build a Convoluted Neural Network (CNN), TensorFlow provides a module called *layers* in the high-level API [6]. This module performs the three components that are part of every neural network:

- **Convolutional layers**, which apply a specified number of convolution filters to the image. For each sub region, the layer performs a set of mathematical operations to produce a single value in the output feature map. [6]
- **Pooling layers**, which downsamples the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. [6]
- **Dense (fully connected) layers**, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer. [6]

This is achieved by the methods *conv2d*(), *max_pooling2d*(), and *dense*() respectively. Each receive the tensor (matrix) as an argument and return a new tensor as an output. The output is then sent to an Estimator, which can train, evaluate its accuracy, and return a prediction.

To further the neural network into deep learning, TensorFlow implements the use of embeddings. An embedding is a matrix that is fed into the deep learning system along with the other feature columns. The embedding matrix is filled with random values each time the training is iterated to extract features that were not defined by the data set. A higher dimension, passing in as an argument, gives the network more freedom to recognize features in data sets the network hasn't seen before.

To apply these neural network methods to GANs, two networks are defined: a generator() and a discriminator(). The generator takes in a vector and returns a feature vector, while the discriminator takes the feature matrix and returns the probability on how acceptable the vector is.

While the discriminator applies the *conv2d*() method multiple times to pick out features that it uses to distinguish between data, the generator does the opposite. It takes data and uses the *conv2d_transpose*() to take the transpose of the matrix, which deconvolutes the data [7].

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Figure 4.2.4 GAN Loss Function [8]

Defining a loss function is an important part of training that TensorFlow also handles. In Figure 4.2.4, the loss function used for GANs in TensorFlow is mathematically defined. The loss function is how far the generator is from an acceptable input during training.

```
G_sample = generator(Z)
D_real, D_logit_real = discriminator(X)
D_fake, D_logit_fake = discriminator(G_sample)

D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
G_loss = -tf.reduce_mean(tf.log(D_fake))
```

Figure 4.2.5 TensorFlow Loss Code [8]

Since the function to minimize loss is already implemented in TensorFlow, we simply define the loss for the generator and the discriminator. The reduce_mean function returns the mean of all the elements in the tensor, which is used to calculate loss. D_loss and G_loss will then be used in the train class with the minimize method. There are two methods of approach to minimizing loss. The first method minimizes log(1-D(G(x))). However, this method saturates the gradient, leading to an inefficient training system [7]. The second method maximizes log(D(G(x))). This method creates stronger gradients early in the learning phase [7]. The example in Figure 4.2.5 applies the second method in TensorFlow by taking the reduced mean of the formula by assigning it to a negative value to maximize it [4]. There are, however, other losses that can be applied, such as the sigmoid_cross_entropy_with_logits() method, depending on which performs better on the data set. Both G_loss and D_loss are inserted into the minimize method to effectively maximize positive outputs.

4.3 Neural Network Approaches

There are 3 kinds of fingerprint scanners that are on the market:

- Optical scanners: These look at the print and creates a monochrome matrix of values
    - Often used in corporate or security settings
- Capacitive scanners: Reads the change of electrical charge in a bank of capacitors to create a monochrome matrix of values
    - Often used with cell phones
    - Considered secure against prosthetics, or fake fingerprints as if the material changes, the fingerprint looks different to the scanner

- Ultrasonic scanners: Reads a 3D image of the print, and uses that to create a monochrome matrix of values
    - New technology
    - May be able to detect prosthetics when worn over a finger

Each of these approaches would require a different approach at 3D printing a fake fingerprint. To start out, however, we will be using an already created database of monochrome fingerprint matrices as test data. If we get to the point of using a real scanner though, we will have to develop a method of reading a print and normalizing it to a similar matrix of values.

4.3.1 FingerNet

Our first proposed approach would be to use the FingerNet [9] implementation. In this implementation, they train a neural network with a series of prints and associated labels to output a label when given the associated print. The neural network attempts to avoid problems with orientation and extracts segmentation, enhanced features, and minutiae. Our research has shown that recognition approaches that use minutiae are more easily fooled than methods that utilize the entire fingerprint. This makes it a good starting point for our project.

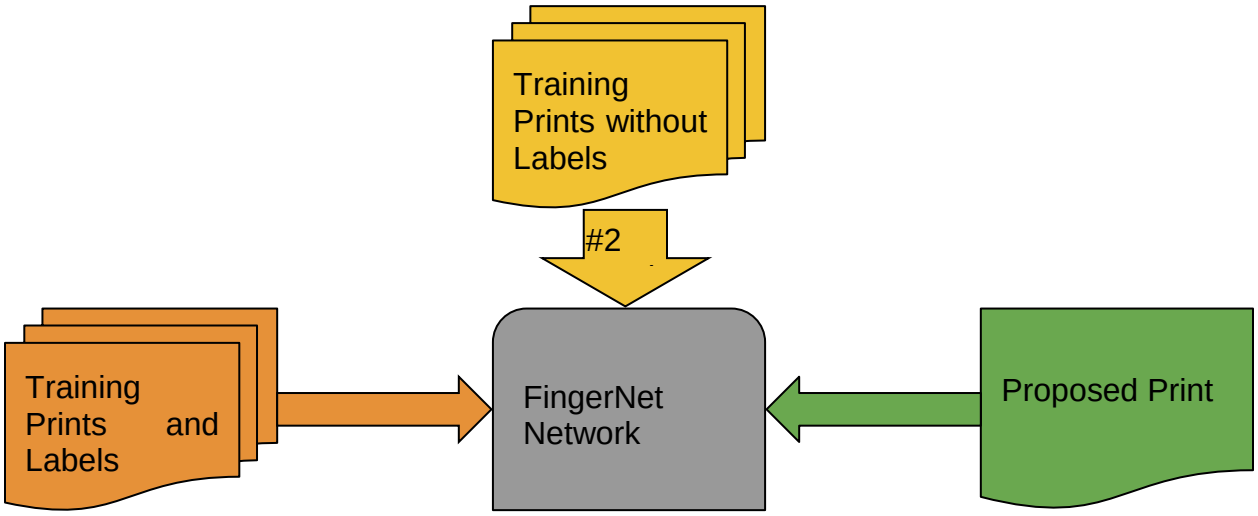


Figure 4.3.1.1 The Training, testing, and deployment model for FingerNet

FingerNet uses a 3-stage from training, to testing, to deployment strategy. First, the network is trained on the input data and associated labels. It learns the prints as well as it can. Then, it is tested on the prints without labels. This repeats with more training and testing, until 100 epochs or it has converged. Finally, it is deployed, where you give it one print and it should tell you from whom it came from to a reasonable accuracy.

Going into this, none of us really have an idea on how to build a neural network to identify a fingerprint, so FingerNet can provide a good starting point. Furthermore,

it is available under the MIT License, so we can use it so long as we provide a copy of the MIT License with our code. Finally, this is open source, so we can learn how the recognition is done by tinkering with it and alter it to improve it for our needs.

4.3.2 Microprints

Another model of network training is to use micro prints, similar to how cell phones are accessed. In this model, we may be able to use the FingerNet code to implement our network, but instead of a single stored fingerprint for each user, we have a collection of partial prints that the network will need to learn as the same person. This method may have the potential to be very easily fooled, since it potentially has no knowledge of how the prints are connected. It may see a test print as stitched together prints in the wrong order.

We would want to test this model because of the chance that a generalized set of minutiae may incorrectly match more people in the data set when the trained network can stitch together prints from a person anyway it wants. As such, it may hold the greatest success for us in creating a master fingerprint for the system or even a fingerprint general enough that it would work on any system for any user.

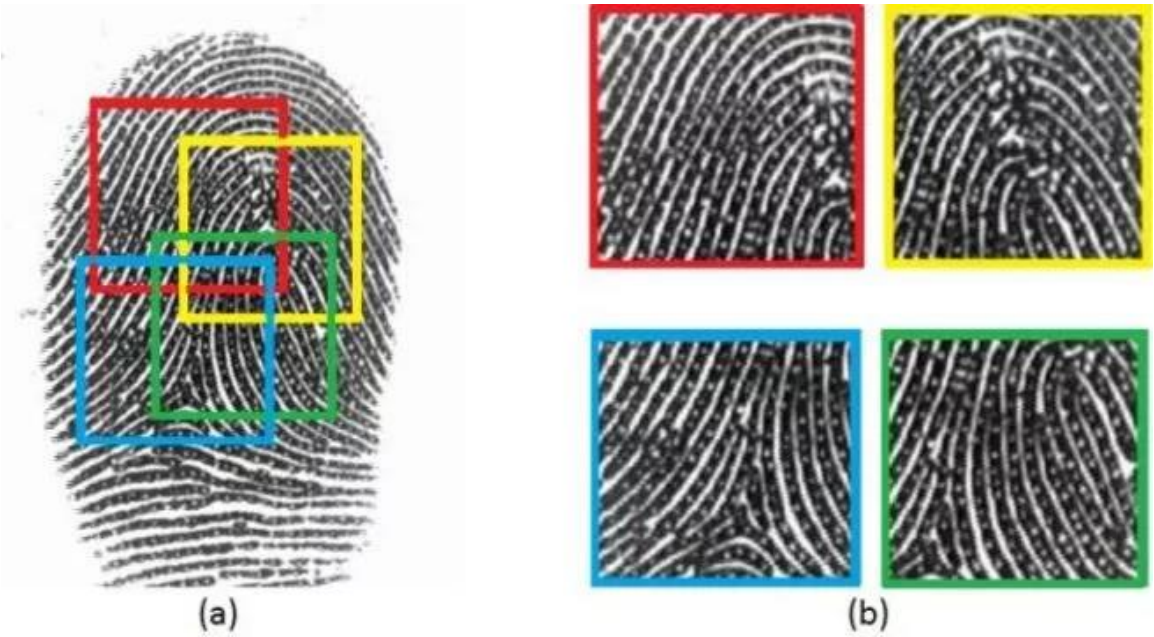


Figure 4.3.2.1 Microprint examples courtesy of Popular Science [47]

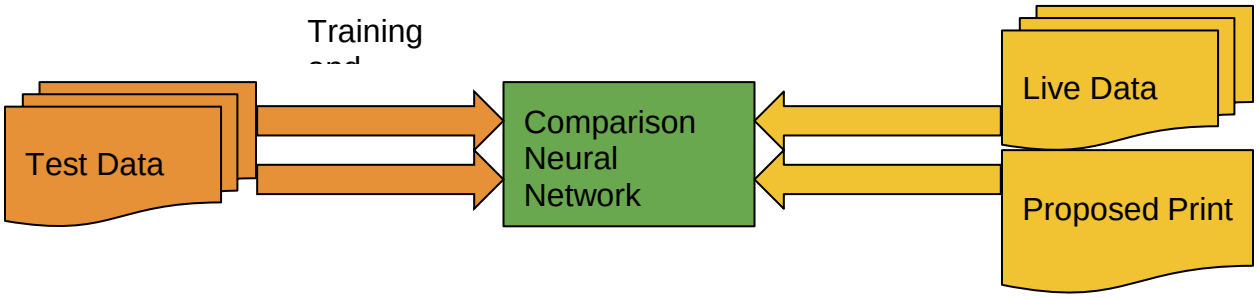
The problem that we would face in practice in this model is many cell phones use capacitive scanners to read a print. These scanners will read a print made of different materials in a different way. As such the same fingerprint made of plastic, as one made of skin, will look different to the scanner. We will have to try to choose a material for 3D printing that mimics skin's natural ability to accept charge. There is a small chance we can train a network to accommodate the differences in

material, but this would have to be late in the project, after we implement 3D printing.

4.3.3 Matching Networks

Another approach we have considered is one where a Neural Network is given 2 sets of prints: one from a known individual and the other one proposed to be from that individual. It will then look at both prints and determine if the proposed print belongs to that individual.

This approach likely will need a complete reworking of the FingerNet model, as it takes in two separate images, not one. It's very likely that we will also want to compare the entire print, rather than just the minutiae extracted from them. Rather than learning a set of prints and who they belong to, the neural network will learn how to compare two prints together and report if they match.



Again, we will have 3 stages for the network: Training, Testing, and Deployment. However, both training and testing use the same model. A control entry is picked, and another random entry is picked. If the CNN judges correctly whether they were the same print, then it passes; otherwise, it fails. If the test entry is the Control Entry, some acceptable amount of noise or shifting may be done to the print to encourage non-exactness in matching. After all, no two fingerprints are identical even if taken one after another.

This model will benefit from having multiple prints for the same person from the same finger. This data may not be available in the database available to us, so we may have to simulate it until we implement a real scanner. Simulating rolled print from a non-rolled print can be done with selective stretching. The reverse could be simulated with selective compression. Rotated prints are simulated through rotation. Shifted prints can be done by shifting the print a bit, adding noise or random values along the edge, or even just cutting off the print at that point.

4.3.4 Matching Network with Microprints

A variation on the above approach would be to implement it with microprints. Instead of just matching one print to another print, it would be matching a set of microprints to another print. This would require another redesign of FingerNet, but

it would be along the same lines of the above matching network. In this method, the network would have to learn that it simply needs to match some of the microprints to the print. The more accurate the matchup is, the stronger the confidence in the match becomes.

This approach is once again to model the way cell phones store prints, possibly with a harder to break neural network. The problem with this approach is that a cell phone may not be able to simulate such a large neural network quickly enough for production. However, this might change in the future as cell phone processors get stronger.

There is still the risk that a microprint set may be stitched together incorrectly by the CNN when matching, causing false positives just as with the FingerNet network. This is because the CNN has no idea how the prints fit together and no way to tell. The idea is that the more microprints of similar areas you have, the less likely this will happen, as you must match more of the prints to get a confident match.

4.3.5 Finger Print Hashes

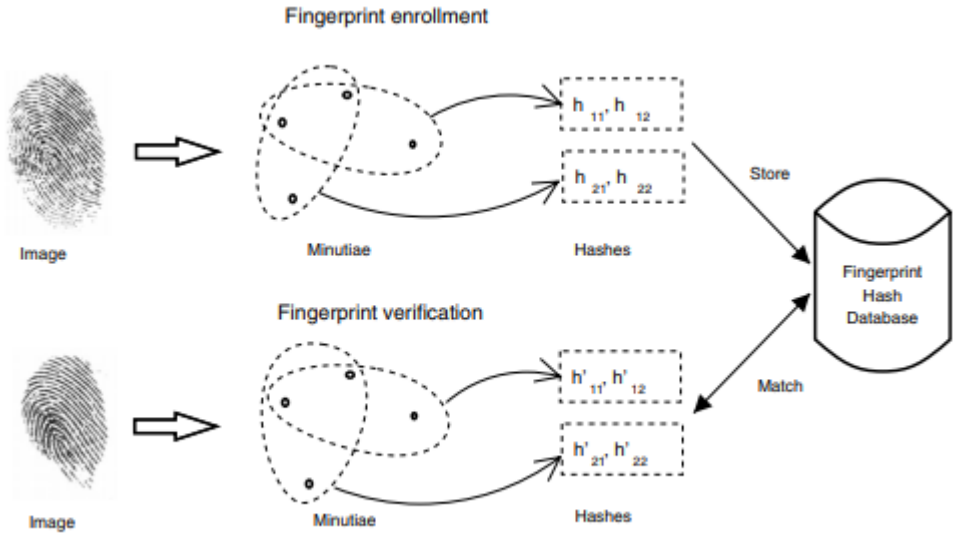


Fig. 1. Securing fingerprint information.

Figure 4.3.5.1 An Example Problem with Minutiae Hashing [10]

It is insecure to keep fingerprints on a central computer, just as it is insecure to keep the passwords for users on a computer. Passwords are stored as hashes. Perhaps we can make a method of hashing the minutiae of fingerprints in such a way that we can check against a hash. If we can write a matching neural network that matches to a fingerprint, we can easily have it match against a hash of a fingerprint as well.

A good hashing system needs to be immutable on rotation, stretching, compression, or shifting. It may be enough to hash the found minutiae only. If it isn't enough, perhaps hashing the relative positions of the minutiae from each other or a central point can also work. More experimentation would need to be done on how to do a proper hashing of fingerprint minutiae.

One source suggests using minutiae in all order triplet pairs [10]. Doing this can create a reversible hash of the print. Then, by adding a salt along the lines of a secret phrase or passcode, one could randomize which hashing algorithms were used and in what order for each triplet.

The reason why we did not explore this method is that having an external password is outside the boundary of the model we are working with. Furthermore, by adding the password, the security of the print is the password which becomes the security of the system and defeats the whole point of using fingerprinting for security.

4.3.6 Capsulized Neural Networks

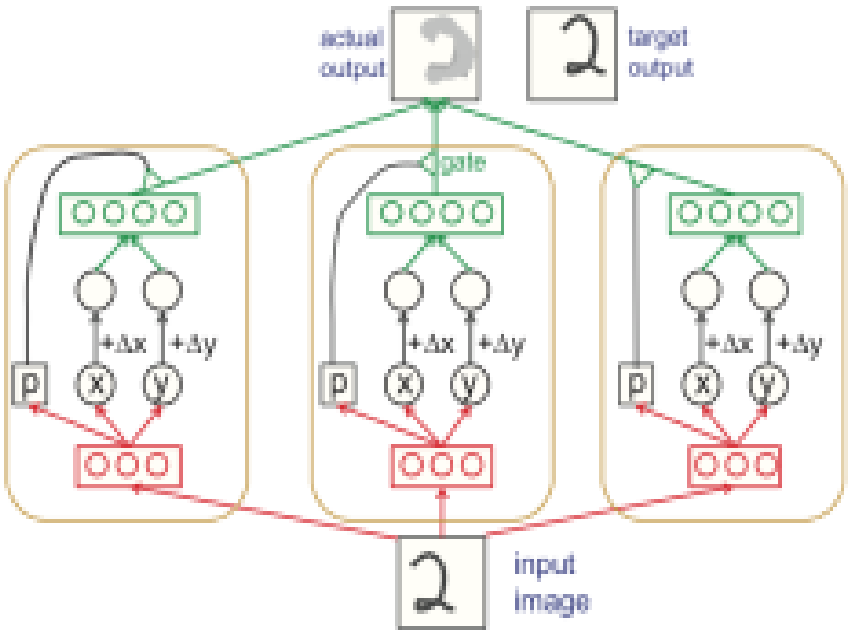


Figure 4.3.6.1 Example Capsule Network Model for Neural Networks [11]

Serial neural networks have been shown to have trouble with adversarial attacks, but it has been hypothesized that smaller networks that specialize in one recognition task may not have this problem. Capsulized neural networks are networks where the output of one network goes in as the input to networks related to that output. In short, the networks run often in parallel.

For use in our system, we would have one neural network that fixes the rotation angle, any stretching or compression, shifting of the print on the scanner, and feeds that in parallel to an array of networks. Each one looks for individual features specific to that network. Finally, each of those passes those features to a final network that will determine which person, if any, the fingerprint belongs to. This can also be used with matching networks as well.

4.4 Approaches for Backend Operations

4.4.1 Host Models with TensorFlow Serving

Once the TensorFlow models, which would include the authentication and adversarial networks, have been properly trained and configured, the next challenge arises when attempting to provide clients easy access to those models. The team at TensorFlow anticipated this particular issue and developed TensorFlow Serving. TensorFlow Serving works by implementing a server which can take in input from a client (in the case of this project, the web-client) and pass that data off to a running model for either classification or prediction purposes.

To understand how TensorFlow Serving works on a high-level, several key concepts and terms must be acknowledged:

- Model
    - In TensorFlow, a model refers to the artifact that is created in the process of algorithmic training on a dataset. In our case, the models are the trained authentication and adversarial networks. There would be multiple different models for each different implementation of these networks.
- Servables
    - Servables are the general layer of abstraction in TensorFlow Serving. They represent the element that the clients use to perform computation. In most cases, servables refer to the TensorFlow models that need to be used. In our case, the servables would be represented by our neural network models.
- Sources
    - Sources are modules utilized to locate servables. It's built-upon a plugin based architecture, so one can point services to servables located in file systems, outside servers, along with other locations that the servables may be found.
- Loaders
    - Loaders handle the infrastructure that is independent of the models that are represented by the servables. Loaders essentially gather the necessary metadata that is required to instantiate the servable once it comes time to host it.
- Aspired Versions

    - Aspired Versions represent the collection of servables (that have come from the source and loader) which are available and ready to be hosted. These aspired versions are passed off to the manager, which then takes over.
- Dynamic Manager
    - The manager handles the serving, loading, and unloading of servables from the server. The manager takes the observer approach, watching the sources for changes in model versions and handing the uptime (depending on the policy set) for the exchange of the old and new servable.

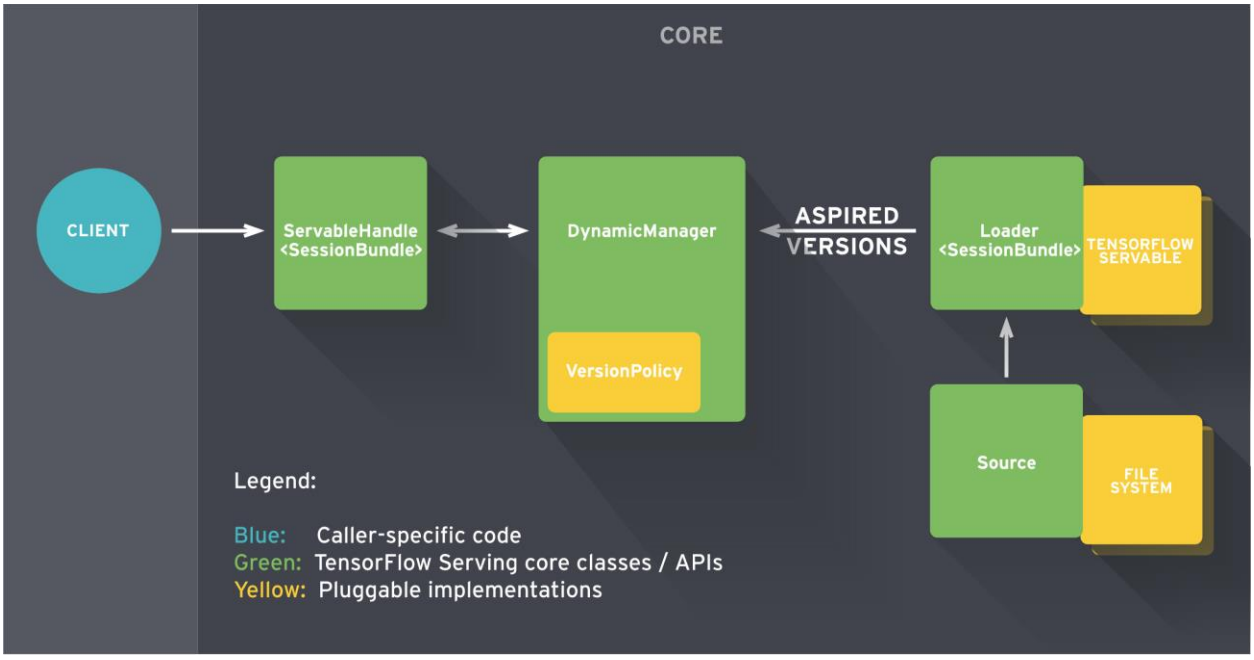


Figure 4.4.1.1 TensorFlow Serving Architecture

4.4.2 Possible Web Frameworks for Web Application

Since our neural network models are being created in Python, using TensorFlow, it seemed like a reasonable step to consider making the backend for the web-hosting portion of the project in Python as well. This would allow interactions between the TensorFlow components and the web application components to be completed in a easier fashion and we are less likely to encounter compatibility issues. With this in mind, we have determined from our research that Python is a very suitable language for hosting web applications and is widely used in large scale web projects.

There are several commonly used Python web frameworks:

Figure 4.4.2.1 Django Web Framework Logo

Django is a free and open source Python Web Framework that works as an all in one solution for web development. It uses the model-view-template (MVT) architecture pattern and focuses on giving developers the ability to rapidly create database-focuses websites. In doing this, it provides several pieces of key functionality that can help speed up the development of the web app portion of this project and shift focus to the neural network and adversarial fingerprint side of the project. The following are some examples of the functionality that Django provides:

- Web server for hosting pages.
- Extensible authentication system for web app accounts.
- Form validation and serialization system which can easily transfer information into a connected database.
- User-friendly, dynamic administrative interface for handling administrative and maintenance tasks.
- Unit testing framework for web apps.
- Built in security measures such as protection against cross site scripting, cross site request forgery, and SQL injections.

In combination with these key features which would speed up the development process, since Django is written in Python, we are able to develop an API between the web backend and the TensorFlow training modules so the website will be able to initiate training on an uploaded fingerprint dataset. Django also has native support for connecting to popular SQL databases such as MySQL and PostgreSQL [12], so this process of uploading the fingerprint data becomes substantially easier with this integrated web framework.

Figure 4.4.2.2 Flask Web Framework Logo [13]

Another commonly used Python based web framework is Flask. Flask is considered a “micro” framework and aims at providing a simple core functionality for web development. Flask is quite different than Django as it’s not necessarily meant to be used to create large web applications. Instead of providing all of the

functionality that Django provides out of the box, Flask aims to provide a simple and fast way to develop a web application. While this may not be suitable for our final web application, Flask could prove itself rather useful when it comes to testing APIs for our TensorFlow models. Since there are a lot of moving parts between TensorFlow Serving, the website frontend, the website backend, and the individual TensorFlow models, Flasks simplicity can be used to isolate models and control the flow of data in a very easy manner. This will make debugging our models easier and scripts can be made to rapidly use a Flask created API to test our models.

4.4.3 Containerization of Backend Services

Since the backend portion of our project will have many moving parts, it would benefit our team greatly if we had a system that was effectively able to isolate our individual parts from each other while still providing the common dependencies. Software containers appear to address this issue. For this project, the container platform Docker will provide more than enough functionality.

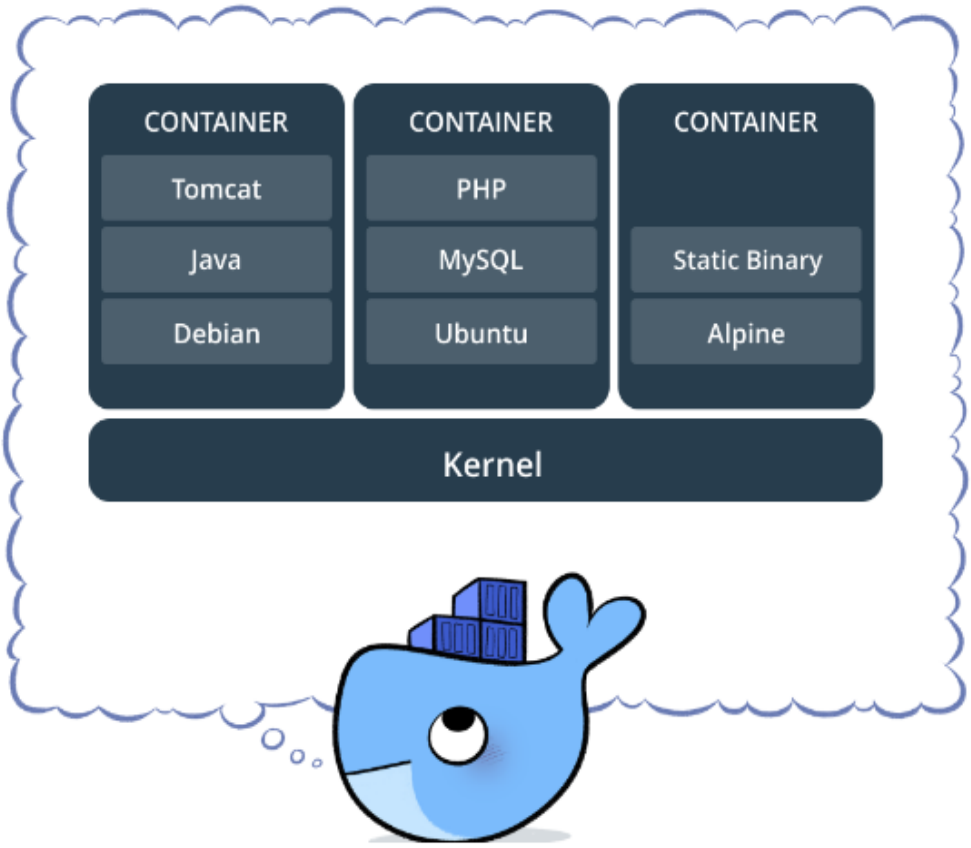


Figure 4.4.3.1 Infrastructure Diagram of Several Docker Containers [15]

In their documentation regarding the design of containers, Docker describes software containers as, “a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings” [15]. Docker containers run on top of the Docker kernel

and provide a system for embedding all the needed binaries and dependencies along with the software you are trying to run. This eliminates the need for the guest operating system in something like virtual machines, which will benefit our project since our core software will already be rather resource intensive.

In order to further understand how the Docker platform can fit into this project, we must address a few key concepts:

Need to get Docker glossary sources.

- Dockerfile
  - A Dockerfile is a text file which contains all the commands needed to build a Docker image.
- Docker Image
  - Docker images act as the blueprints for containers. Images contain the ordered collection of filesystem changes and the needed parameters to execute upon runtime. It is important to the note that the image isn't running, so it does not change state.
- Container
  - A container is simply a runtime instance of a docker image. Images are pulled from a repository and when the images are instantiated, they become a container.
- Docker Repository
  - A Docker repository is a collection of Docker images. Multiple versions of an image can be kept on a repository through the use of tagging.

Along with the high-level benefits that the Docker platform provides, its convenience is complemented by the fact that TensorFlow serving supports and actively recommends the use of Docker containers. As each team member is working on different approaches to both the fingerprint neural networks and ways to produce adversarial fingerprints, having everyone working on the same TensorFlow and TensorFlow serving installation would not be optimal. Rather, it would be preferable if each team member had the equivalent of their own clean environments with the two pieces of software installed on them. The process of providing TensorFlow within a container is relatively easy. The company TensorFlow has an actively maintained and supported container with TensorFlow and its needed dependencies installed. Getting such a container up and running involves simply executing the following commands [16]:

```
$ docker pull tensorflow/tensorflow
$ docker run -it -p 8888:8888 tensorflow/tensorflow
```

Provided that we have Docker properly installed on our production server supplied by our sponsor, the first command would download the container image from Docker Hub. Docker Hub is a resource which hosts a myriad of software images. The next command would instantiate a container based on the image that was just

pulled. Once this is completed, a container with all the necessary dependencies and TensorFlow will be up and running. However, this particular container will only have access to the CPU in order to train the TensorFlow models. This command can be expanded to provide each team member with their individual TensorFlow environment that can be accessed.

Using these tools and commands does solve the dependency conflict issue. However, these containers running TensorFlow would only be suitable for training smaller models. By default, Docker does not have the ability to leverage Nvidia GPUs for training. For our purposes, since we are training large neural networks on many fingerprints, we would need to leverage the GPU. Otherwise, using Docker would not be time effective. Fortunately, Nvidia developed a container runtime called nvidia-docker [17].

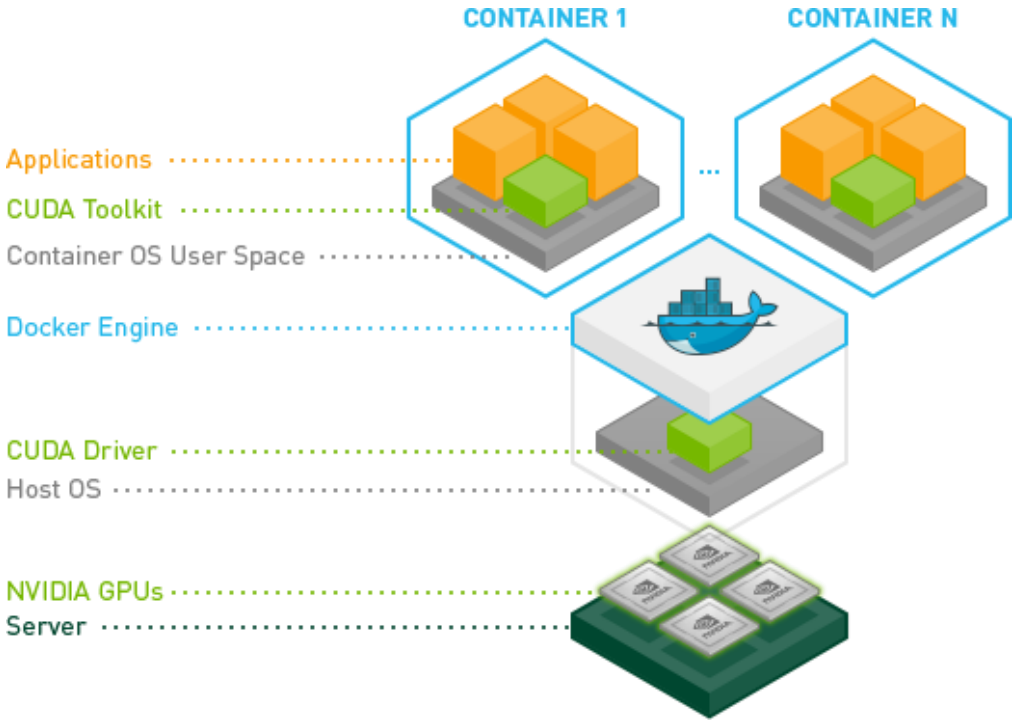


Figure 4.4.3.2 Nvidia-Docker Architecture [17]

Figure 4.4.3.2 demonstrates the new architecture of nvidia-docker. The original Docker platform performs as normal by encapsulating the needed software and dependencies. From there, instead of doing the needed processing using the CPU, the CUDA driver allows Docker to redirect its processes to the NVIDIA GPU on the Docker host. This will substantially decrease the amount of time needed to train our fingerprint neural networks. As an added benefit, utilizing the CUDA driver and the available GPUs does not added any additional complexity to getting TensorFlow up and running in a container. In order to start TensorFlow with GPU support, one would issue the follow command on the Docker host [16]:

```
$ nvidia-docker run -it -p 8888:8888 tensorflow/tensorflow:latest-gpu
```

Another major advantage of using Docker to manage our individual environments and backend resources is the fact that TensorFlow also actively supports a containerized version of TensorFlow Serving. Since TensorFlow Serving is a larger, more complex system, easing the effort required to set it up repeatedly would be immensely helpful. With the needed Dockerfile maintained by TensorFlow, creating a Serving container is just as easy as creating a TensorFlow container like before. One first starts by pulling the Dockerfile located on TensorFlow's website. Once the needed file has been obtained, you can simply build and run the container by executing:

```
$docker build --pull -t$USER/tensorflow-serving-devel -f Dockerfile.devel.
$docker run -it $USER/tensorflow-serving-devel
```

This will start one container of TensorFlow Serving, in which we can host our trained models and have a RESTful API available to access them (This process is described in detail in 4.4.1).

Along with the noted benefits of Docker that are specific to TensorFlow, the design of software containers also provides key benefits to this project. Docker containers have the ability to be tagged, and since they run on the Docker kernel, the containers can easily be moved between different hosts. This would allow the team to test out a new feature or new algorithm for recognizing fingerprints and easily pass that off to another member of the team. This uniformity of the environment in which Docker provides will help relieve headaches and reduce debugging time. All of our backend service could also be unified into one platform. The two Python-based web platforms described in 4.4.2 conveniently have Docker containers available. Alongside the potential Python based web platforms, there also exists premade containers for the MEAN (MongoDB, Express.js, AngularJS, Node.js) stack. Overall, choosing a backend platform like Docker, which supports all of our needed software, can help redirect the effort towards the core of our project (developing the neural networks) and reduce the time spent getting the infrastructure working in a way we would desire.

4.5 Web Front-End Frameworks

The team will be building an interactive web service that allows users access to the neural networks. The users should be able to download fingerprints, upload fingerprints, modify the neural networks, and view graphics based on the neural networks. The interface should be fast, user-friendly, and pleasing to look at. The code should be modular and adhere to web standards. Our data doesn't have high demands from the front-end services; however, it should still be flexible enough to add new modules without breaking the rest of the website. We're especially looking for something easy to set up so we can concentrate our time and efforts more on the neural networks.

4.5.1 ReactJS

React is one of the most popular web services, alongside Angular. While it is not technically a framework, it is a view oriented library. It's backed by Facebook and used by Netflix, Uber, Paypal and other tech giants. It's ideal for single-page creations, and it's popular for its flexibility and modularity [18]. It is typically bundled with other libraries and preferred for being able to work with them well [18]. It's supported by a big community, and it has the ability to make mobile applications with React Native.

```
var Greeting = React.createClass({
  render: function() {
    return (
      <p>Hello from React</p>
    )
  }
});
ReactDOM.render(
  <Greeting/>,
  document.getElementById('greeting')
);
```

Figure 4.5.1.1 Example React Code [18]

Unfortunately, React is also well-known for its steep learning curve. The code is ridden with long, verbose declarations, and developers typically have to learn multiple libraries to use with React to create a full website [18]. It is best for mid-sized projects for teams already familiar with React.

4.5.2 AngularJS

AngularJS is a traditional MVC framework backed by Google. It is known for providing superior structure to complex projects and simple implementation of two-way data binding. Two-way data binding allows updates to web interfaces from model-to-view and view-to-model, which benefits simple projects [18]. It combines HTML and JavaScript to create tags, which set it apart from React. It also features a big community that is slightly smaller than React's community. It handles mobile development elegantly with Angular 2. The main language used is Typescript, which is a new language built on top of JavaScript, so the team would have to be familiarized with a new language. It is generally meant to be used in enterprise-level web projects.

```
var app = angular.module('app', []);
app.directive('parentDir1', function () {
    return {
        restrict: 'EA',
        template: '<div>{{name1}}<child1></child1></div>',
        compile:function(tElem , tAttrs){
            console.log(': compile called');
        },
        link: {
            post: function(scope,elem,attr){
                console.log(': Pre called');
                scope.name1 = 'Parent directive Without pre link';
                scope.parentName1 = 'Parent 1';
            }
        }
    };
});
```

Figure 4.5.2.1 Example Angular Code [18]

Similar to React, AngularJS also has a steep learning curve. It's ideal for use in enterprise web interfaces, but it's not meant to be easy to get up and running. Many of its features, such as complex structuring and DOM management, are not useful for small projects. Because it's such a mammoth of a library, it can be on the slower side as well [19].

4.5.3 Vue.js

Vue.js is a lightweight framework that was released a year after React. It features concise syntax, a minimal learning curve, and a small package size that works well with third-party libraries [20]. The main selling point of the framework is how fast and easy it is to get it up and running with minimal prior knowledge. Vue.js is the highest performing framework out of the three frameworks due to its size, and it eliminates the bloated features that Angular comes prepackaged with that are rarely used. Instead of Typescript, pure JavaScript is used to cut down on the extra effort of learning Typescript [20].

```
<script>
  new Vue({
    el: '#app',
    data: {
      message: 'Hello from Vue'
    }
  });
</script>
```

Figure 4.5.3.1 Example Vue Code [18]

On the other hand, it is not backed by a large organization like React or Angular, which may mean that the amount of resources is less or of lesser quality. It has a healthy sized community, especially in China, and it is quickly starting to gain considerable traction in the West. However, the community is still smaller than React and Angular, and some third-party documentation is in Chinese due to a strong following in China [20].

4.6 Approaches for Generating Adversarial Fingerprints

4.6.1 What are Adversarial Examples?

Before diving into the various ways to generate synthetic adversarial fingerprints, it's worth exploring what adversarial examples are and how they really affect machine learning based algorithms. In general, adversarial examples are crafted inputs that can intentionally cause a misclassification in machine learning models. What is also worrisome about these misclassifications is that they are often produced with high confidence by algorithms. To demonstrate this process consider the following example [21]:

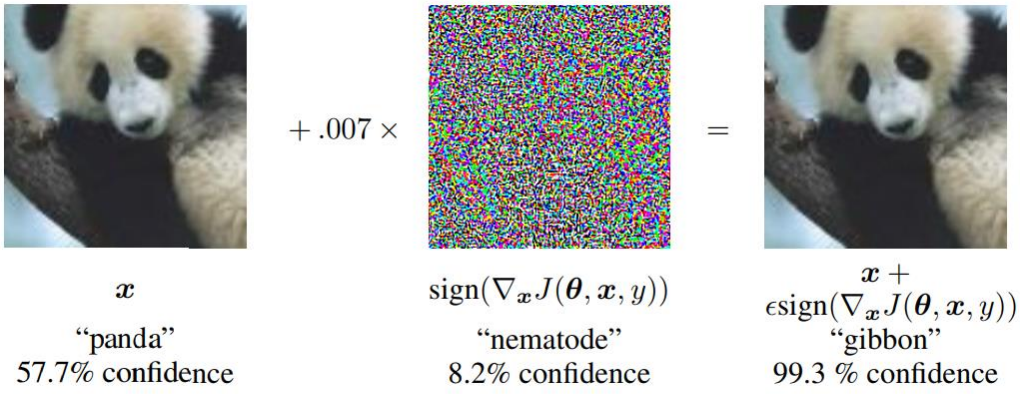


Figure 4.6.1.1 Adversarial input result on GoogLeNet [21]

Figure 4.6.1.1 shows an example adversarial input that was crafted and then applied to GoogLeNet, Google's image recognition neural network. In this example, a change that is imperceptible to humans can change the classification of the neural network from a panda to a gibbon with high confidence. This example exists within the context of computer vision; however, these types of examples exist across multiple mediums. For example, adversarial examples can be as simple as misspelling blacklisted words in spam emails. The words are written in a way that intentionally fools the spam filtering algorithm into classifying the email as a legitimate message. Another, more commonly referenced example of an adversarial example involves a team of researchers at the Massachusetts Institute of Technology (MIT) whom 3D printed a turtle with a particular pattern on its shell which fooled Google's Object Detection AI into classifying it as a rifle [22].

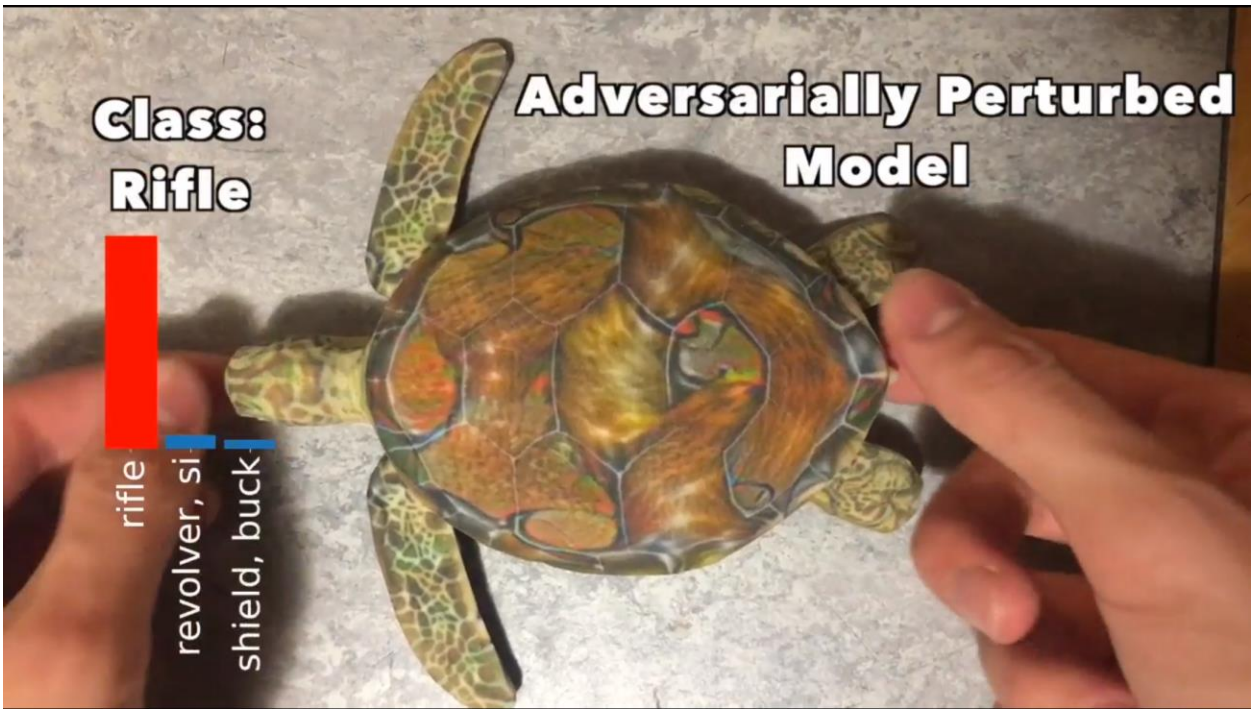


Figure 4.6.1.2 Video demonstrating Labsix's 3D printed turtle classified as a rifle. [22]

Due to how severely these adversarial examples can change the classification of an algorithm and how durable their details are (as demonstrated some crafted changes can be printed on to physical objects), a field known as adversarial machine learning is dedicated to studying these examples.

In general, there are two main types of attacks in which adversarial examples are used to exploit machine learning based algorithms: evasion attacks and poisoning attacks.

- Evasion Attacks
    - Evasion attacks work in the manner of the few examples described above. Several examples are crafted in a fashion such that their classification is considered legitimate. This is the most common form of attack.
- Poisoning Attacks
    - In some machine learning algorithms periodic retraining on the distribution of data is required. This is necessary because over time, the distribution of data that is fed into the algorithm can change. The retraining process is needed to ensure the algorithm continually provides the right classifications. In this context, adversarial examples are crafted in a way so when the algorithm retrains with the examples in their distribution, they will alter the algorithm's classification abilities. This comprises the system as a whole since the algorithm is no longer properly trained for the distribution it expects.

In the case of our project, we are more concerned with the evasion attacks. Adversarial images of fingerprints can be made such that they will be classified as a legitimate, authenticated user in a recognition system. As one can imagine, the ramifications of having a potential exploit like this are rather severe.

4.6.2 Using Generative Adversarial Networks to Create Adversarial Fingerprint Images

In the context of our project, Generative Adversarial Networks prove promising in generating adversarial fingerprints. From the current research available regarding creating adversarial fingerprints, there seems to be two main options: the adversarial fingerprints are made only using the fingerprint minutiae (known as the template level) or actual images are developed (known as the photograph level). Since we aspire to apply these adversarial fingerprints to widely used authentication systems, our group is aiming to create photograph level images of fingerprints.

As Bontrager et al. describe in their paper “DeepMasterPrint: Fingerprint Spoofing via Latent Variable Evolution”, Fully Visible Belief Networks (FBVN), Variational Autoencoders (VAE), and Generative Adversarial Networks (GAN) are all popularly used for image generation. However, as they mention, FVBN’s tend to produce images one pixel at a time, which can lead images to become rather noisy, and VAE’s produce very smooth images, which often lack the needed detail to distinguish between the fingerprint minutiae. Generative adversarial networks tend to take the best of these two networks and produce images that contain less noise than FBVN’s and are sharper than those produced by VAE’s. [23]

Generative adversarial networks are comprised of two main components: the generator and the discriminator. These two components are both neural networks which are trained in an unsupervised manner. These components work and are trained together. The generator works by taking in random noise as input and then outputs an image from that. The discriminator holds the responsibility of classifying various images as “real” or “generated”.

The three main steps to training the generative adversarial network consist of:

1. Take images of real fingerprints and use them to train the discriminator. This needs to be a varied dataset from which the discriminator will be able to learn what are considered “real” fingerprints.
2. Initiate image generation on the generator to generate a new set of “non-real” images. This will be the generated set which you will mark as such and then train the discriminator on. Training on the generated set will allow the discriminator to distinguish these images from the set of real images.

3. Gather the gradients produced by the discriminator and pass those features to the generator. This will allow the generator to train and eventually, produce images that the discriminator will classify as real.

This three step process is repeated until the adversarial fingerprints are able to authenticate on the recognition system.

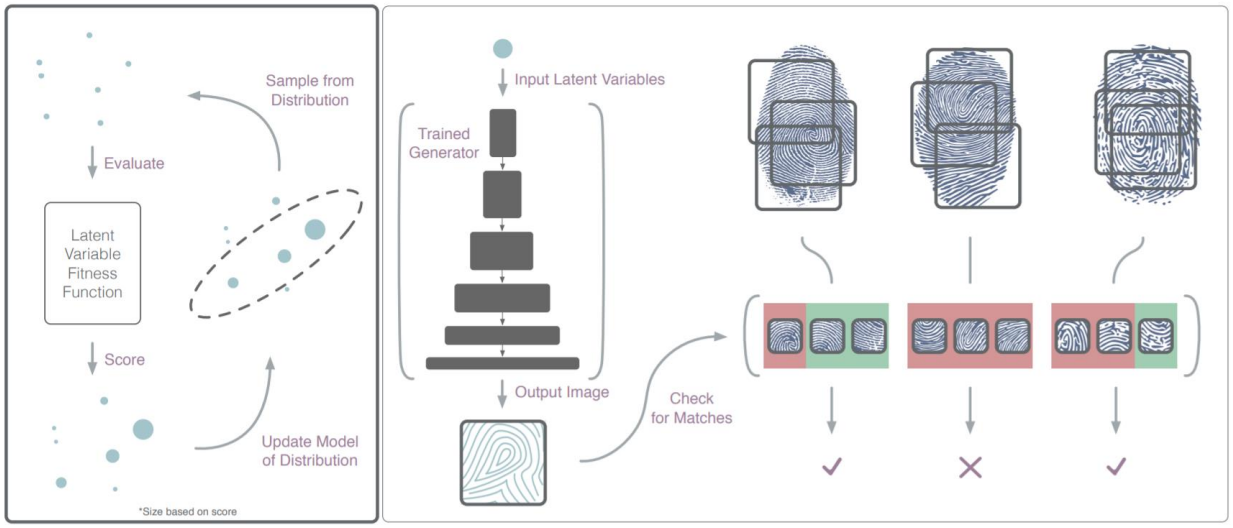


Figure 4.6.2.1 DeepMasterPrint's Implementation of GANs using Latent Variable Evolution [23]

One of the main difficulties encountered in this training process involves keeping the generator and discriminator training at the same pace. If one part of the generative adversarial network becomes much better than the other, it causes issues in training the underperforming component and the resulting GAN won't be representative of the needed dataset. A large portion of the research performed on generative adversarial networks are directly related to rectifying this stability issue. An algorithm that shows promise (and that Bontrager et al. recommends) is the Wasserstein process for creating generative adversarial networks. By using the Wasserstein distance function to calculate to difference between the distributions consisting of the real images and the generated images provides better gradients for the generator than the standard method for constructing GANs.

4.6.3 Architecture of GANs

Generative adversarial networks seem to be our most promising approach for generating adversarial fingerprints. Their versatility and customizability will allow us to test our various possible implementations and see which works best for producing accurate fingerprint images. As mentioned in 4.6.2, generative adversarial networks have two main components: the generator and the discriminator. In order to understand how GANs work in a more thorough manner, we need to explore these two components in depth.

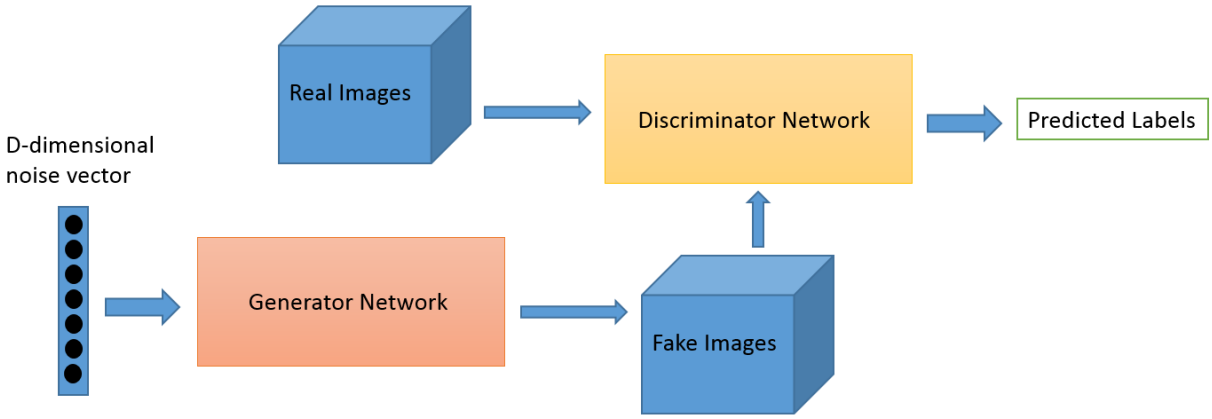


Figure 4.6.3.1 Generative Adversarial Network Architecture [24]

The generator and discriminator, in a sense, work as inverses of each other. The discriminator works by taking in an image as input and through multiple convolutional layers, outputs a single probability indicating whether an image is real or fake. The generator works in the reverse direction by taking in a D-dimensional vector with random values and upsampling that into an image. Using backpropagation, the discriminator supplies information to the generator and the generator's weights and biases are adjusted accordingly. This process is repeated until the generated images converge on the distribution of real images.

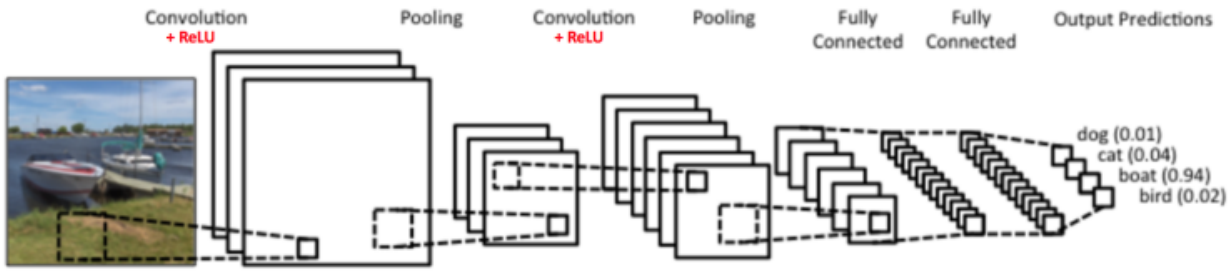


Figure 4.6.3.2 Convolutional Neural Network Architecture [24]

The diagram above shows the process that an input goes through, layer by layer in a convolutional neural network. This is a good representation of how the discriminator in our project's generative adversarial network will work. In the standard implementations of convolutional neural networks, there are four main components:

- The Convolutional Stage
- Introduction of nonlinearity through ReLU (Rectified Linear Unit)
- Pooling
- Classification (Standard Multilayer Perceptron)

At a high level, the convolution stage works by extracting features from the image (also known as applying a “filter” to the input). Since images can be represented by a matrix of integer values (0 to 255 * Number of Color Channels), particular matrix operations can be performed on these images. In our case, since we are using gray-scale images of fingerprints, our images would only have one color channel. Each convolutional layer represents a different filter is used to gather additional features from the image. Common features that are identified include edge detection, gaussian blurs, and curve detection [25]. The end result gives us several feature layers that can be used later in the process. As expected based on name, these filter layers are produced by performing the convolution operation on the original image with the desired filter. The exact size of the resulting convolution is determined by three distinct parameters we specify prior to the computation:

- Depth
    - The depth or number of feature maps that will be produced is determined by how many filters we apply or more intuitively, how many features we want to extract from a given image.
- Stride
    - The stride determines how many pixels we move in between convolutions with different sections of the image. Generally, this means that a higher stride leads to smaller features maps, but less features are gathered in return.
- Zero-Padding
    - The input can be padded with a layer of zeros in order to effectively isolate and convolve the edge values. Depending on the size of the original image and the size of the filter, this can change the size of the convolution.

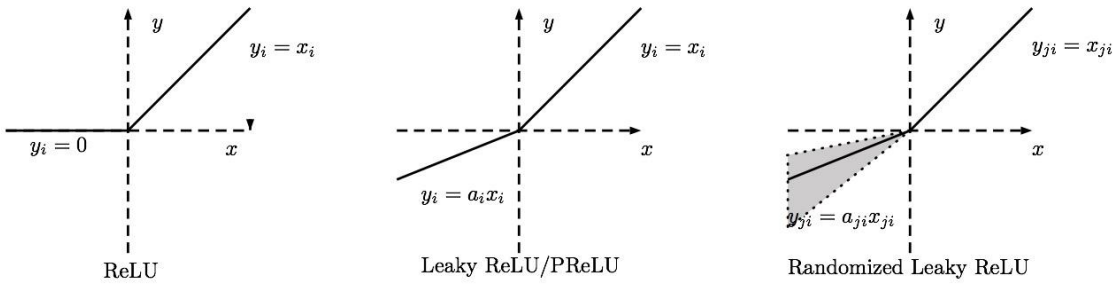


Figure 4.6.3.3 Various ReLU Operations [26]

Up to this point in the network, we have been performing mappings from one set of values to another set of values, essentially only performing linear operations. Since most data found in practice is nonlinear and at this point, our computed values are unbounded, we need a way to introduce nonlinearity into our neural network and bound our results in order to prevent extremely large values later in the network. Once the feature layers have been created, an operation called ReLU (Rectified Linear Unit) is set as the activation function to introduce nonlinearity. ReLU is a simple function which conforms negative values to zero. Other functions like tanh and sigmoid can be used for this particular step. However, interestingly

enough, it has been found that this rather simplistic function tends to perform better in practice (Research is being conducted to explain this phenomenon). Some CNN developers, depending on their individual datasets, found mixed or slightly improved results through using variations of the ReLU operation including Leaky ReLU and Randomized Leaky ReLU. These operations simply let some negative values through some of the time. Adjustments like these will be determined more concretely as we test various implementations of the GAN and figure out which hyperparameters and activation functions serve our datasets bests.

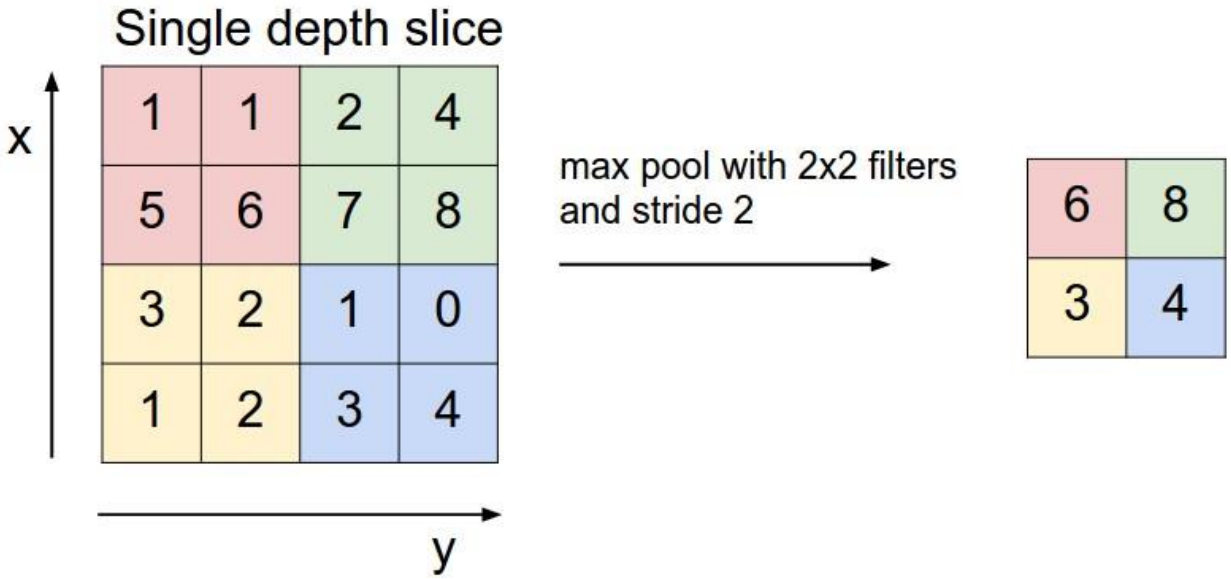


Figure 4.6.3.4 Example of Max Pooling [27]

The next step in developing the convolution neural network for our discriminator involves pooling the data from the previous layers. Keeping the initial dimensionality in the layers produced by convolution and applying the activation function can be rather costly computation wise. Pooling is a way of taking the matrices computed in the previous layers and reconstructing them in a smaller form while still keeping the key information. Retaining the key information and discarding other values in this manner also helps to reduce the chance of overfitting, since the model no longer has access to each value associated with the original image. Multiple pooling operations are used in research; however, it has been found that max pooling, where the maximum value is extracted from each NxN subsection, has the best performance in practice. Regarding generative adversarial networks, there are several good arguments for why pooling may not be optimal. Some researchers suggest that pooling should be removed altogether in favor of exclusively convolutional layers with some using larger strides to downsample the data. We will have to evaluate both options while conducting our research to see how the authentication rate of our adversarial fingerprints are affected.

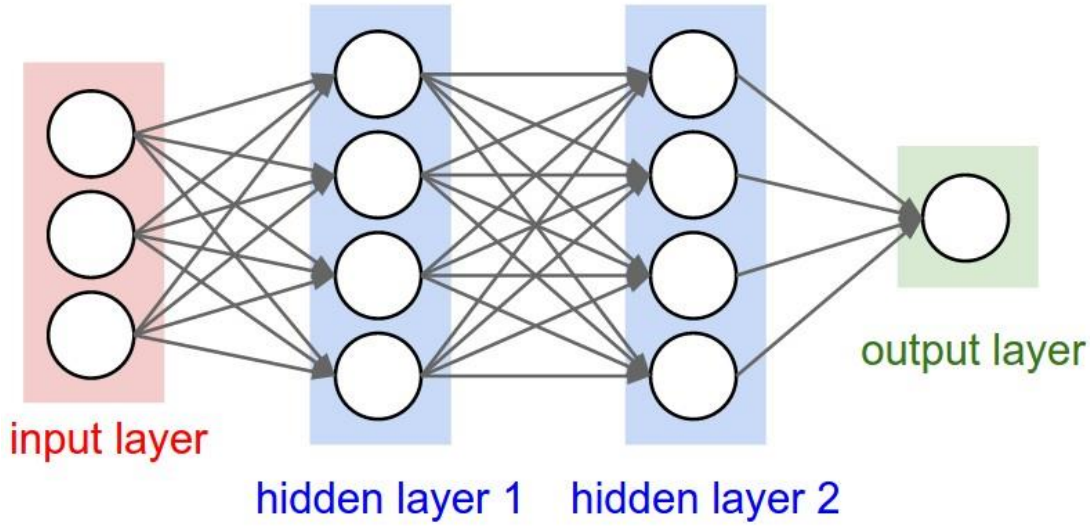


Figure 4.6.3.5 Fully-Connected Layers [27]

The final portion in our discriminator will consist of the fully connected layer. Up to this point, the combination of multiple convolutional layers, as well as the application of the activation functions, have supplied this fully connected layer with a variety of high-level nonlinear features associated with the original input image. The fully connected layer learns the high-level nonlinear combinations associated with the outputs of the previous layers. The values that are produced through the hidden layers in the multilayer perceptron then get run through a softmax activation function in the final layer. The softmax activation function compresses all of the outputs from the previous layers into values between 0 and 1 (ensuring that the sum of all results is 1). The result in the output layer is a probability that a given input falls into a particular category. In the case of our project, we would have two categories, one representing that the particular fingerprint is authorized and in the database of fingerprints, and one representing a fingerprint which is not authorized.

Once the fully connected layer supplies an output of probabilities for each category, the error rate is calculated and using gradient descent, the individual weights used in the previous layers are adjusted. This process is repeated until the discriminator can accurately categorize images.

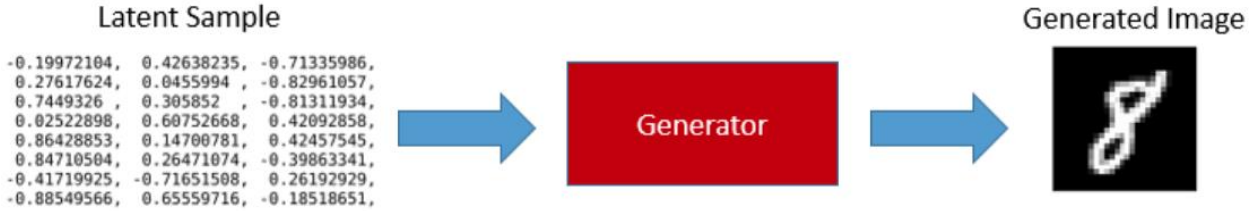


Figure 4.5.3.6 High-Level Generator Procedure [28]

At a high level, the generator works as an inverse of the discriminator. Instead of taking in an image as its input and then applying various convolutional and fully connected layers to generate a numerical output, the generator takes in an initially

randomized input and produces an image. The input to the generator is referred to as the “latent sample.” The latent sample is a set of randomized values that are adjusted based on the results of the discriminator. Initially, the generator produces images nothing like the desired output. It is through passing the generated images to the discriminator, calculating the loss, then adjusting the weights associated with the latent sample which allows the generator to learn. This process is repeated until the generator is able to produce images that resemble the desired output.

4.6.4 Various Optimizations for GAN Training

Since generative adversarial networks are under constant research, there have been many small modifications that have been discovered to perform better in practice. Much like how ReLU was found to work better in practice, then its reason for better performance was discovered later, many of these modifications are being researched to justify their effectiveness.

Through our research we have gathered that some of the following modifications to our process may prove beneficial in generating higher quality adversarial fingerprints [29]:

- Utilize DCGAN or other hybrid models
    - Deep convolutional generative adversarial networks apply various architectural constraints when designing GANs. These type of GANs have been shown to be more stable during the training process.
- Use the Dropout method during training
    - The dropout method is a way of adding randomness to the system by giving each neuron a probability to be removed (value set to zero)
- Randomize the batch size
    - In practice, it has been found that adjusting the batch size during the training process (in both the generator and the discriminator) produces images with less noise.
- Consider Using a ReLU derivative
    - Generally, ReLU is the standard activation function which produces the best results. However, for more complex images, the Leaky ReLU activation function may work better.

These are a small subset of the possible modifications we could make to our generative adversarial network training process. As we implement our various versions of the GANs, we will adjust these parameters accordingly in order to improve the quality of our adversarial fingerprints.

4.7 Annotating Fingerprints

4.7.1 Problem

We recently discovered after getting access to the fingerprint database that there are hundreds of thousands of prints to use, but none of them have annotations on minutiae. FingerNet requires these annotations to function, as it doesn't find its own, so we will have to annotate a huge amount of prints in a short period of time so that we can train a network.

What are annotations? Annotations are markings on a fingerprint of specific areas of interest. Specifically, they are features that are hopefully unique enough to differentiate one print from another.

What are minutiae? Minutiae are recognizable features on a fingerprint. Specifically, they are features recognizable to a computer. These tend to be places where the lines split, where they merge together, where they begin, and where they end. To a computer these are locations where the derivative of the image is in more than just two directions. Another way to think of them is that they are corners.

Because of this problem we will have to find a solution soon to handle this problem. There are several approaches and I will discuss each of them here.

4.7.2 Manual Annotations

The first option is the most obvious. We could write a program to allow us to quickly annotate these fingerprints by hand. However, this is not optimal, as we would spend the better part of a month just getting a small working set of fingerprints to use. We would prefer a rather large set of fingerprints so that we can generate master prints that match many users.

Unfortunately, this will have to be a fall back option if we can't find another way.

4.7.3 Automatic Annotations Through Computer Vision

We could use computer vision to find those features of interest. After all, if a computer can see and differentiate those features, the same computer can perhaps use those features to tell prints apart. There are 2 main approaches we know of: Harris corner detection and S.U.S.A.N. corner detection.

4.7.3.1 Harris Corner Detection Algorithm

Harris corner detection works by first smoothing the entire image. A simple Gaussian filter is good enough for that. After that, you find the Hessian Matrix (H) for the given pixel I. This is given by:

$$H_1(p) = \begin{bmatrix} I_{xx}(p) & I_{xy}(p) \\ I_{xy}(p) & I_{yy}(p) \end{bmatrix},$$

In this matrix, Ixx(p) is the second partial derivative of the image at pixel I with respect to x. Iyy(p) is the same with regards to y. Ixy(p) are both the same and it's the second derivative, first with respect to x and then, with respect to y.

To take the partial derivative of an image, you simply subtract two pixels. This is because it is a discrete function of brightness. This can be done for the x direction by applying a mask of [-1, 0 , 1] over every pixel. For the edges, you can wrap around or choose to lose the edge information. I do not think the edge information is too important for our project, so that may be the better option.

Once you have gotten the Hessian of the smoothed image, you then check "cornerness," our confidence that this is a corner, of each pixel with the following function:

$$\mathbf{Cornerness}(p, \sigma, \alpha) = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)$$

In this formula, the lambdas are the eigenvalues of the hessian matrix at that point, and alpha is a constant we can adjust up or down as needed to get more or fewer corners. A good starting point for alpha is 1/25.

Once we have determined the cornerness of each point, we can suppress nearby corners that are less than another nearby corner, and then extract the strongest corners of the image. These would be our minutiae.

Pros of this method:

- This is a well understood method by the community, and we would have no problem finding help implementing this design.
- Most of the tools, filters, and like, are already coded into Open Computer Vision library, so we wouldn't be remaking much.

Cons of this method:

- While we had some success on my own with this method, we don't think we had that great a success. We're not confident in our skills at this technique.
- We had trouble understanding what the Hessian matrix represents, and we have another method we understand better.

More than likely, we will not use this method, but it is outlined here in case we have to fall back to it.

4.7.3.2 Smallest Univalue Segment Assimilating Nucleus (S.U.S.A.N)

For this method we are comparing the overall brightness of 37 nearby pixels to the center pixel, or nucleus. This is done with a circular mask that looks like this:

[0 0 1 1 1 0 0]
[0 1 1 1 1 1 0]
[1 1 1 1 1 1 1]
[1 1 1 1 1 1 1]
[1 1 1 1 1 1 1]
[0 1 1 1 1 1 0]
[0 0 1 1 1 0 0]

In this mask, the pixels marked 1 are the ones we are considering. All others are out of consideration for the nucleus or the pixel in the center. It is a 7x7 mask, unlike the 3x3 or 5x5 masks usually used when smoothing, but it is not much slower.

To apply the mask, Every pixel under the mask is used as input to calculate the following formula:

$$n(r_0) = \sum_{r \in M} e^{-\left(\frac{I(r)-I(r_0)}{t}\right)^6}$$

In this formula, I(r) is the brightness of the currently considered pixel, $I(r_0)$ is the nucleus pixel, and t is a threshold for difference of brightness, and can be played with for optimal results. This formula outputs a range of values from (0, 1] where it outputs 1 if the two-pixel's brightness match perfectly, and it drops off sharply as they don't match. $n(r_0)$ is the value of similar pixels in the region, a number from (1,37].

In short, to read this formula, the number of similar pixels in the region is equal to the sum of the formula for each pixel under the mask. Once you have this value, you apply a threshold to it:

$$R(r_0) = \begin{cases} g - n(r_0) \text{ for } n(r_0) \leq g \\ 0 \text{ for } n(r_0) > g, \end{cases}$$

Here, G is the threshold, a number that usually you want around or less than half of 37.

You may have noticed that this method requires a complicated formula calculated 37 times per pixel, but you can reduce the number of calculations dramatically through memorization. First, the center pixel always calculates to 1 regardless, so there is no need to do any calculation for that. Second, the brightness in an image varies between [0, 255]. It is also only integers, so, if you calculate once for each value and store those values in a table, you can look up the value you want by the difference between the two brightnesses. This results in much more optimized code. As a result, the number of actual complex calculations per pixel drops to less than 1, compared with at least one square root in Harris. This makes S.U.S.A.N. the faster method overall.

This method is robust to noise, but not without its flaws. What we have here is only the bare bones of the algorithm and will work on an image without noise. However, most images have some noise.

Minor salt and pepper noise can be easily removed with a median filter, but that is slow. The problem with salt and pepper noise is each pixel tends to look like a corner and will spam the detector.

A solution we found to this, however, is while evaluating a pixel for cornerness, we will accumulate the approximate center of mass of the object we believe belongs to the pixel. If the center of mass is too close to the center pixel, we disregard that pixel for cornerness as an object can't have a corner in the middle of it. This has the result of not only ignoring salt and pepper noise, but also the following situations:

```
(a)      (b)      (c)      (d)      (e)
  --111--          --111--          --010--          --000--          --000--
  -11111-          -10001-          -00100-          -00000-          -00000-
  0111111          1000001          0001000          0000000          0000000
  0111111          1001001          0001000          0001000          0011100
  0000000          1000001          0000000          0000000          1111111
  -00000-          -10001-          -00000-          -00000-          -11111-
  --000--  --111--  --000--  --000--  --111--
```

Key: 1 = r0 ~ r, 0 = r0 != r, - = border of mask

In all these situations, the bare algorithm will detect a corner at the pixel. In (a) we can solve that by suppression of nearby maxima. In (b), (c), (d), my explained method will solve the problems, with only (c) being a valid corner. In (e) we have a quandary. Is this a corner, or is it just a curvy line? If we set our threshold too high, it will ignore it even if it is a corner. If it's too low, it will place corners along every line. This is a problem that we couldn't work out in the project.

A possible idea that could be applied stems from the fact that (e) could be a corner, or it could be an edge, but we're looking at it through too small of a lens. If we

increase the size of the lens in cases where it is close, we might get a better picture of what we are dealing with though. It does mean more calculation, and slowing down the algorithm, but it would be on a selective basis, only on the corners that are barely corners.

How do we evaluate the new data? One method would be to see if it becomes more cornery, or more like a strong corner, than it previously was when we zoom out. If it does get more cornery, then we accept it as a corner, while if it becomes less cornery, we pass it up as a corner. We will have to experiment on the ones that remain the same.

Another method would be to have a second threshold for that larger mask. If we determine it's a corner by a stricter criterion zoomed out, we will accept it.

We have tried neither of these methods, so we will have to see which will work better. It may come to be that a proper threshold will simply work best, as we haven't tried this method on fingerprints yet. Optimally, we want a few of the best corners after all.

Pros:

- We can conceptualize this very well, allowing us to handle problems in a reasonable manner without just taking shots in the dark.
- It may be the fastest method, which would mean something when we do several hundred thousand fingerprints one after another.
- We already have mostly working code we can adapt for this

Cons:

- It still means we must write a program to write out the annotations in the proper format.

There is one other method to consider. OpenCV has a built-in feature finder. We do not know how it selects features, however. It may be useful to use, or it might be a 100% bust. If all else fails, we can try this for detecting minutiae.

4.7.4 Open Source Auto-Annotation

Instead of developing our own process for automating minutiae annotation with computer vision techniques, we could possibly leverage existing software that is open source. This would save us time on the more tedious aspects of the project, so we can focus on the core of the system, which is the pair of neural networks. FingerNet mentioned MINDTCT in their research paper. According to the paper, "MINDTCT is an open source minutiae extractor from NIST Biometric Image Software".
The National Institute of Standards and Technology developed biometric image software to be distributed to the FBI and the Department of Homeland Security [30]. Their system is named the "Integrated Automated Fingerprint Identification

System (IAFIS)". They provide thorough documentation of their work. The documentation describes their system, where to access their code, how to install it, and how to use it. Certain sections of their source code is export controlled and only available upon request, but fortunately most sections are not export controlled. The specific package from the software distribution that we are interested in is MINDTCT – Minutiae Detection.

MINDTCT:
The links provided in the documentation for the source code of MINDTCT lead to pages that no longer exist. Fortunately, the source code has been preserved on GitHub, and the documentation still has the complete description of the algorithm and processes.
Input:
The input to MINDTCT is a fingerprint image. According to the documentation, this can come from ANSI/NIST, WSQ, baseline JPEG, lossless JPEG file, or IHead formatted file.
Output:
After the fingerprint image is processed, the output contains all minutiae that could be determined from the image, the location of each minutia, the orientation of the minutia, the type, and the quality. The output file format depends on the input format. ANSI/NIST files have outputs of ANSI/NIST files. Any other input format has the results output as a formatted text file.
Drawbacks:
The documentation acknowledges that their automated process of identifying minutiae is ineffective based on the quality of the fingerprint image provided as input. They specifically mention that latent fingerprint images present a problem for the minutiae extractor. In the case of latent fingerprints, they used expert examiners to manually mark the minutiae. This may or may not be a problem for us depending on the quality of fingerprint images that we have access to or that are uploaded when our system is connected through our web interface.

Figure 5. Latent fingerprint (left) with matching tenprint (right).

Example from the NIST documentation [30] of a latent fingerprint image. Their process would not extract quality minutiae information form the image on the left.

4.8 Generative Adversarial Network Attack Styles

There are two main styles of attacks on neural networks. Directed and undirected. Each have its own strengths and flaws. Undirected attacks are easy to set up and train, but tend to have a higher failure rate, and the training takes longer. Directed attacks are harder to build and train but work more consistently.

No method of attack has a 100% chance of success, nor does any method have a 100% chance of failure on any system. Success or failure is determined by the accuracy of what the attacker wanted to do versus what he was able to accomplish. For us, this means that if we want to make a print that matches every person in the system, but it only matches 48% of the prints in the system, it is a huge success for us but carries 52% room for improvement under that model. Current models claim to have achieved 74% accuracy on this task on modern fingerprint databases, but no information was given about what kind of discriminator, GAN, or even what protections were in place to defend against adversarial attacks. As such, we should not be judged on the effectiveness of others but on our own merit.

We will next outline each of these styles.

4.8.1 Undirected Attack Method

An undirected attack is where you build a GAN, and then ask it to generate output with no input. Then you provide feedback to it on whether it worked through a loss function that calculates the difference between the actual and expected values. It is very analogous to having a child draw a picture of a tiger without knowing what one is, and then criticizing the defects until the child draws the right thing.

Building such a neural network is easy, and such a network can be quite small, but training it takes a long time, as there is no direction given to the network. It only knows it was wrong, and how much it was wrong, but not what was wrong. Many of the earliest GANs use this approach.

An attack such as this would be a good early model to learn how to build a GAN, and a good initial test against our discriminator to see if it is robust. If we can easily fool the discriminator using an undirected attack, there would be no need for a stronger system until we find a way to strengthen our discriminator.

4.8.2 Directed Attack Methods

There are not one but 3 directed attack methods. They consist of black box attacks where the attacker knows nothing of the system they are attacking, white box

where the attacker knows all of the details of the system, and grey box which is found somewhere in between.

Considering the varying degrees of difficulty in protecting against these methods, if we fail a black box attack, there is no reason to try a grey or white box attack. Likewise, if we can defend against a white box attack, there is no reason to worry about grey or black box attacks.

The loss function for a directed attack is a little different. Rather than calculating the difference between the actual and expected values, you calculate how far from the initial value (or the ground truth of a print) and attempt to minimize the amount of noise needed in order to generate an image which gets seen incorrectly by the discriminator network. The closer the value to the initial value (signifying success), the stronger the positive feedback, and the further the value from the initial value (signifying failure), the stronger the negative feedback. Defining such a loss function has been discussed in several papers.

Next, we will examine each of the directed attack methods in detail.

4.8.3 Black Box Attacks

In a black box directed attack, the attacking GAN knows it wants to generate a fingerprint that looks like fingerprints in the discriminator network, but it does not know what they look like or what protections if any are installed to thwart adversarial attacks. In practice, such attacks have been very successful even against what was thought to be strongly protected network, showing that a basic black box attack has a good success rate even against the best discriminators around. However, there are a few methods of protection that were developed that can defend against black box attacks.

A black box directed attack is like the undirected attack where the network does not know what fingerprints are in the system and is given no information upon creating a print. However, it differs in that the loss function requires information about what print we were trying to match and how close we came so we still direct it in the right direction, even though it does not know precisely what prints the system has on file.

If we cannot achieve a network that can defend against a black box attack, there is no point in moving onto even a grey box attack. However, a success against a black box attack may be as much as reducing its accuracy by as little as 30%. For example, if initially it has a 90% effectiveness and then later we can improve the discriminator to only have a 60% effectiveness, this would be a good move onto our stretch goal of devising a defense.

4.8.4 White Box Attacks

A white box attack is a stark contrast to a black box attack. In the attack, the GAN knows everything, from the prints used to train a network, what minutiae were used, what kind of defenses are involved to thwart attacks, and even the logits output of the discriminator network are up for grabs. Virtually all data is at the disposal of the attacker.

Such an attack is typically viewed as impossible to fully, or even partially defend by several papers we have reviewed, because every step, in development, that has been taken to prevent an attack can be undone by the attacker. We still need to try this if we succeed in black box testing, simply because while it's not believed to be possible, no one has yet to prove it so.

These attacks are quick to train, as they are highly targeted on the chinks in the armor of the discriminator. Every possible attack vector is pinpointed and focused on. These attacks are the most likely to produce the best results, as should be expected. Because of that, if we can find a way to make a network impervious to such an attack, it would be the strongest possible network. Even if we can reduce efficacy of attacks by a small margin, this would be a major victory.

4.8.5 Grey Box Attacks

The grey box attack is, as you would expect, somewhat in between white box and black box attack. As such, its parameters are a bit unclear. It has some, but not all of the advantages of white box testing in that it might know some of the fingerprints being used, but maybe not the minutiae. It wouldn't know something as in depth as the logits of the discriminator but it might know what defenses have been erected. In truth, there is a whole rainbow of grey box attacks that we will have to annotate exactly what we are allowing and not allowing for each one.

Most defenses that have been attempted only impact at most 20% of grey box attacks. As such, even a small improvement in a defense against these attacks would indicate a new defense method we can explore and possibly improve. In one paper, however, they tested 10 different defenses that claimed grey box improvements, and only two provided any improvements against grey box attacks, while 5 didn't even protect against black box attacks. Through this, we know two things: defending even against grey box attacks is not very easy, and there are eight defenses we can avoid because they don't work.

If we can use existing defenses and improve on them to make attacks even 5% less effective would be a great success for our stretch goals and show that grey box attacks are possibly defensible soon. Furthermore, since white box testing requires very intimate knowledge of the system under attack, most attacks will be black box or grey box in nature, and thus, this improvement will greatly increase the security of said systems. While a white box victory would be truly amazing, our goal is improving grey box defense. After all, if an attacker knows a system that well, more than likely he doesn't need to fool the system in the first place.

4.8.6 Adversarial Attacks and Why They Work

Previous work in neural networks tends to give the impression that we have a deep understanding in how they function, and we can tell why they succeed and why they fail. Recent research has shown though that this is a fabrication. Just as the human brain is far more complex than we humans can understand, so have deep neural networks become, even to the point that we really have no clue what goes on in one. This false belief that we can understand them is in a large part simply hubris.

Neural networks are in no small part a very large loss function played out across many steps. Each step has many inputs and many outputs, so tracing success or failure to any one fact is nearly impossible. Everything gets tangled and weaved together in such a way that it boggles the mind. Therefore, many deep neural networks require main frame hardware to run the loss functions efficiently. As a result, and because of the actual limitation on the number of neurons even in the largest networks, when a computer scientist tells you that this network decided this photo was taken outside because it saw blue sky, do not believe him. In truth, we do not know what the network sees in the photograph, or what parts it used to make the decision. In fact, it might not have looked at the sky at all.

Adversarial attacks work in a way to learn the details that a neural network has learned to trick that neural network into thinking the what looks to the human as the same input is in fact something else entirely. In some AI research involving Facebook and the Bar-Ilan University in Israel [31], they discovered that very small, but specific, changes were needed, to cause a neural network to jump to a completely incorrect result in computer vision and computer hearing tasks. Such attacks had a human perceptibility rating of under 0.25% in most cases, and they attacked key features that the neural network was latching onto. The result was a new set of images with slight noise overlaid on them that simply looked the same to humans, or audio recordings with a slight amount of static that cause speech recognition to fail dramatically.

The reason why these attack work is the same as the reason why we don't understand how the networks work in the first place. They've grown so complicated that we can't comprehend what they do, so when we train them, we train them only on what is right. We never train them on the myriad of things that are wrong. After all, we don't know what wrong things we would have to add in the mix to train them well, as we don't even understand what they are learning in the first place. This also means that we humans are poor defenses against adversarial attacks, as we might never recognize them before it is too late.

In short, it may be that our best bet to defend our neural networks against adversarial attacks will come from other neural networks with their own faults. Perhaps in time they will become large enough, and with a large enough training

set, that their faults diminish on their own as has been the trend. For now, it is unlikely that we will ever have a 100% secure defense against attacks.

4.9 Databases

One of the main requirements of this project is to store fingerprint data that we can use to train the neural network. The database will need to be able to store data types such as images, matrices, and labels. Since there will be multiple versions of our neural networks, each which will most probably require different labels and data types, a database that is flexible in of utmost importance. Additionally, since no one in the team has worked in a database before, a database that is easy to use is a plus.

4.9.1 PostgreSQL

```
#!sql
SELECT locale,
count(*) AS amount,
(count(*) / sum(count(*)) OVER ()) * 100.0 AS percentage

FROM users

GROUP BY locale
ORDER BY percentage DESC;
```

```
 locale | amount |        percentage
--------+--------+--------------------------
 en     |   2779 | 85.193133047210300429000
 nl     |    386 | 11.833231146535867566000
 it     |     40 |  1.226241569589209074000
 de     |     25 |  0.766400980993255671000
 ru     |     17 |  0.521152667075413857000
        |      7 |  0.214592274678111588000
 fr     |      4 |  0.122624156958920907000
 ja     |      1 |  0.030656039239730227000
 ar-AE  |      1 |  0.030656039239730227000
 eng    |      1 |  0.030656039239730227000
 zh-CN  |      1 |  0.030656039239730227000
(11 rows)
```

Figure 4.9.1.1 Example SQL query

PostgreSQL is a relational database that is free and open-source. Relational databases store and represent data in rows and tables. These databases are known for complying with ACID (Atomicity, Consistency, Isolation, Durability), which are a set of properties that ensure no data is lost or miscommunicated in case of failure [32]. While this may be important for financial transactions, it's not a priority for our project. Performance is where PostgreSQL shines. It is widely used in large systems where calls and validations need to be made at a fast speed [32]. It can handle execution of complex queries due to a variety of optimizations, such as concurrency without read locks, that are special to the database [32]. In

addition, it features native SSL support for connections and roles to maintain permission, which is important for our dataset, which will often contain personal data [32]. The large following that PostgreSQL has means it has an active community with plenty of documentation.

On the other hand, PostgreSQL is a relational database, which means that this database is “much more concerned with standards and compliance and extensibility than with giving you freedom over how you store data” [33]. However, it does boast a fully-capable NoSQL system within its own relational system that the user can utilize at will. The database allows you to route data to a JSON column, similar to a non-relational database, and allows you to model it later [33]. Essentially, you can use PostgreSQL as a non-relational database. If the project were to later on require a relational approach, the transition is easy and painless.

4.9.2 MongoDB

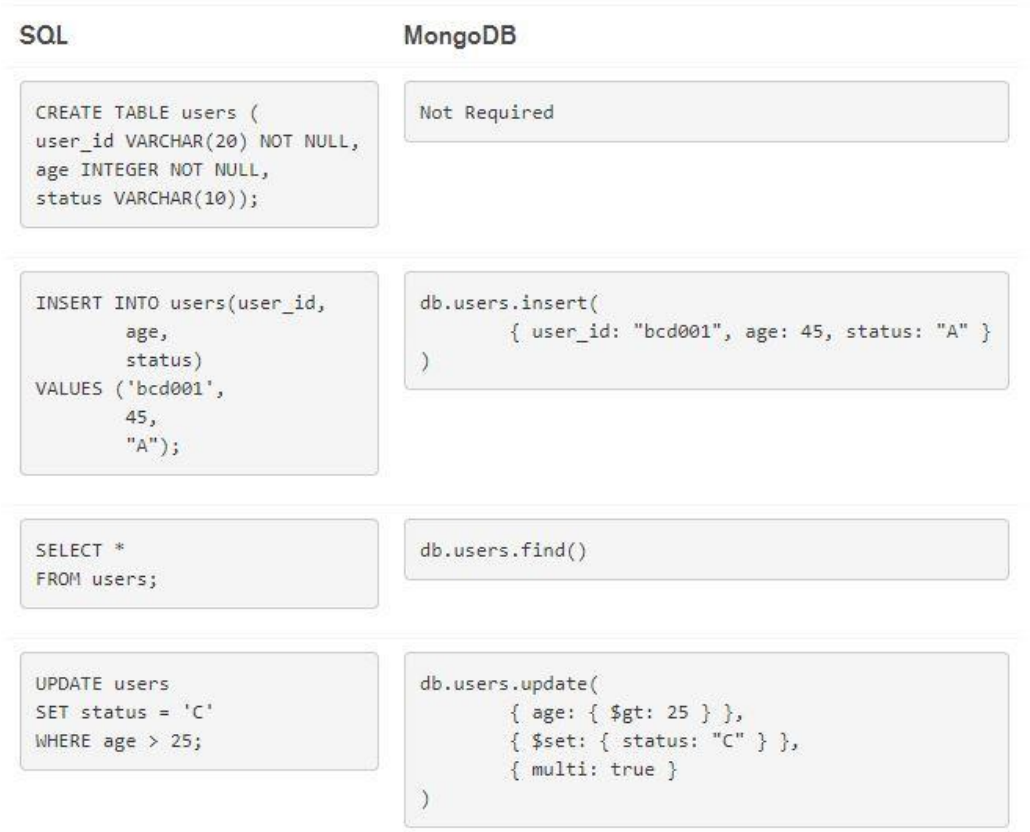


Figure 4.9.2.1 MongoDB Syntax vs SQL Syntax

MongoDB is a non-relational database that is also free and open-source. Non-relational databases store data in documents within collections. It is popular for its concise syntax, speed, and flexibility. Similar to PostgreSQL, MongoDB has a very large following with plenty of documentation. With MongoDB, any type of data type

can be stored in any collection, removing the need for rigid formats [34]. This method works best for projects that need data dumps and simple relations.

The format supports big data well, with the database being used most frequently for quick prototyping and platforms where data is constantly changing and being modified [34]. It's easy to learn and get up and running due to the simple syntax and modern command-line set up. However, due to the nature of non-relational databases, it can be difficult to keep data controlled and organized. MongoDB does not have input validation, meaning that any input validation will need to be done client-side [33].

4.9.3 Image Storage (Database or File System)

There are two main ways that image information is stored in databases. Usually, you either store paths which tell you the location of the images within a separate file system, or you store the images in the database itself as BLOBs (Binary Large Objects). Along with the path to the image, you would also store the metadata about the image within the database when using the former method.
The metadata could include information about the image such as:

- Resolution (pixel width X pixel height)
- Orientation
- Description of the image (in our case, could be thumbprint, index print, etc.)
- Minutia Location and orientation information (in our project's case)

There are advantages and disadvantages to storing the images on the database or the filesystem that we will have to consider. Below are the general pros and cons:
Store on File System:
Pros:

- Can be quicker in serving images to website
- Simplifies the database and reduces storage space
- Cuts down on expense of database processes
- Can easily utilize blob storage cloud services

Figure 4.9.3.1 Example Linux filesystem commonly paired with MongoDB [35]

Cons:

- More complicated to implement and maintain
- Potentially less secure if access to filesystem is not controlled
- Backups need to be done on the database and the filesystem

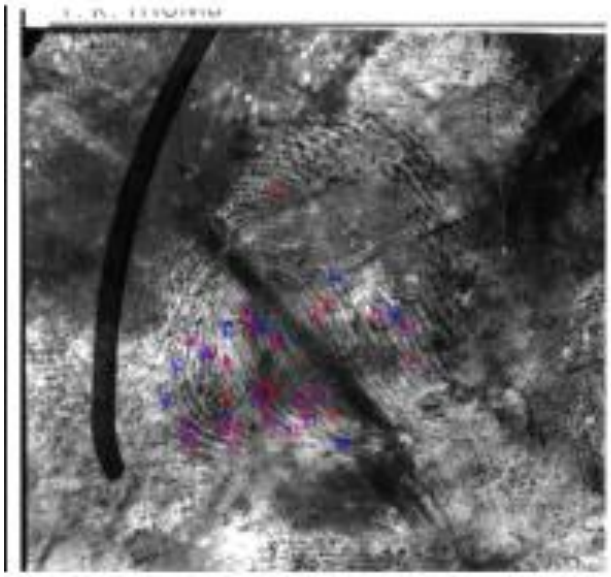
Store on Database:
Pros:

- Simpler implementation for our inexperience
- Files can't get lost as they could within a filesystem
- Any backups automatically are synced between the image and its metadata

Cons:

- Operations can become very expensive with large amount of data
- Backups of the metadata will backup the images as well

4.10 Methods for Minutiae Matching



(d) minutiae map

Figure 4.10.0.1 This image from FingerNet [5] is an example of extracted and manually marked minutiae. The red squares indicate extracted minutiae while the blue square indicate marked minutiae. It is from this information that the comparisons would be made.

There are many different options for verifying a match between the neural network's extracted minutiae and the minutiae provided for validation. Some allow for less error than others. This can affect the success of our project, as more lenient minutiae comparisons would likely allow the generative adversarial network to have an easier job of fooling the authenticator.

In the case of FingerNet [5], they used a pretty simple method of deciding whether extracted minutiae were true. They looked at two characteristics, minutiae location and orientation. If the extracted minutiae were within 15 pixels of the location of the annotated minutiae, and if the angle between them was less than 30 degrees, they were considered a match. In this case, the matching process only considered pairs of minutiae which differs from other methods.

4.10.1 Complex Matching Algorithms

FingerNet also tested whether fingerprint matching benefited from their extraction methods. The algorithm they used is based on minutiae as opposed to the whole fingerprint. When exploring identification methods for our own project, we may want to consider work on extended clique models [36].
Minutiae Tensor Matrix:
The method for fingerprint matching proposed in the paper “Minutia Tensor Matrix: A New Strategy for Fingerprint Matching” [36] is notable for being effective and efficient for minutiae matching while also robustly handling noise and nonlinearity in the minutiae information.
There are clear reasons for using or adapting this method of minutiae matching to be used in our own authentication system. Reasons are as follows:

- The authors have provided this information and methodology as open access as long as the original authors and source are credited. This means we are free to use and modify it for our needs.
- This method of matching is more effective than most other methods I could find through my research.
- FingerNet, which was provided as an example to possibly model our neural network after, has proven to be highly effective in utilizing this method of minutiae matching.
- Lastly, the creators of this method intentionally made a computationally inexpensive algorithm, which is important in the short time frame for which we have to train and build our neural networks.

Two Minutiae Tensor Matrices (MTM’s) are constructed to analyze the finger print. One is a local MTM and observes characteristics of local rigidity. The other is a global MTM and compares sets of minutiae. The combination of these two tensor matrices is what makes the matching process so effective.
Big Ideas:

1. The Minutiae Tensor Matrix proposed by this paper looks at both similarities between minutiae pairs and compatibility of pairs of minutiae pairs.
2. Dense sub-blocks within the Minutiae Tensor Matrices are recovered through spectral matching, which will be discussed.
3. At the local and global level, Minutiae Tensor matrices are constructed for minutia structure pairs and entire minutiae sets respectively. The recovery of the dense sub-blocks within both of these MTM’s is how similarities are calculated.

The minutia sets are organized into their own respective attributed graphs. These graphs are labeled as $G_P = (V,E,A)$ for minutia set P and $G_{P'} = (V',E',A')$ for minutia set P'. Within the graph, the values of the edges represent the distance vector between minutiae. These attributes of each graph are compared in order to determine a similarity score between sets of minutiae pairs. This similarity score is defined as $S_A(ii', jj')$, which would represent the similarity between A_{ij} and $A'_{i'j,}$ which are the distance vectors described earlier between minutiae i and j, as well as

minutiae i' and j'. These pairs are from the two separate sets of minutiae. Doing this comparison within our own project would not be difficult, as we expect to extract the location and orientation information for the minutiae of the fingerprints. The goal is to find a set of minutiae pairs that maximizes the similarity scores. This done with the formula in Figure 4.10.1.1.

$$\tilde{M} = \underset{M}{\arg\max} \sum_{i\,j \in M} S_A(i\,j)$$

Figure 4.10.1.1 Formula for Finding Set of Minutiae Pairs to Maximize Similarity Scores

This approach allows us to analyze the relationships of minutiae and of minutiae pairs on separate fingerprints as opposed to only looking at whether minutiae is within a certain proximity to marked minutiae. This approach is a more exact way to establish correct minutiae pairs.

The Minutia Tensor Matrix proposed in this method is produced by a fusion of a First-order tensor matrix and a Second- order tensor matrix. This is the innovation of this specific method, which incorporates the local and global similarities. The author also notes that the orders of the tensor matrices is intentionally kept below three in order to prevent unacceptable levels of computation. As mentioned earlier, this is an important reason why we should consider this methodology for our own minutiae matching system.

Fused High-Order Tensor Matrix:
The first-order tensor matrix contains the information of the similarities between just minutia pairs. This is calculated locally with the formula below:

$$LT_1(i,j) = S_{ij} = \alpha \cdot e^{-\alpha' \cdot D(v_{ai}, v_{bj})}$$

Figure 4.10.1.2 Minutiae Pair Function 2

The figure above calculates the similarity score of a minutiae pair. In this formula, α, α' are positive parameters that are to be set manually. v_{ai} is the distance vector from a minutia a to a minutia i. The meaning of similar symbols can be inferred from that fact. This is simple enough to implement for our own uses. The locations of minutiae are something that we will have access to. Finding the distance vectors between them should not be a challenge.

The second-order tensor matrix is needed next. This tensor matrix is used to describe similarities between the minutiae pairs. This MTM is what assess local compatibility between pairs of minutia from the first-order tensor matrix. This is calculated using a very similar formula as the one utilized for the local first-order tensor matrix. The formula is below:

$$LT_2(ii',jj') = C_{ii',jj'} = \beta \cdot e^{-\beta' \cdot D(v_{ij}, v_{i'j'})}$$

Figure 4.10.1.3 Minutiae Pair Function 3

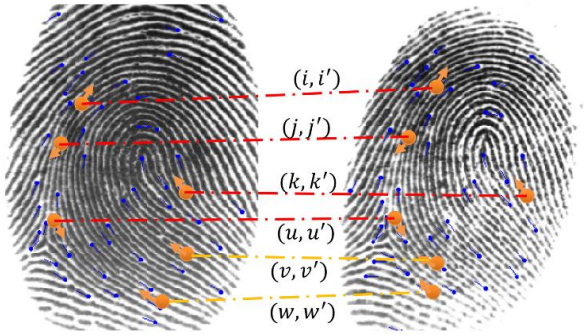
Figure 4.10.1.3 is similar concept to the function shown in figure 4.10.1.2, except now the 1-norm distance is of vectors from different pairs of minutia from each graph instead of from the central minutia, a, to its neighboring minutia within the local minutia topologic structures. As with the previous case, β, β' are still manually set, positive parameters. The fused high-order tensor matrix can then be calculated using these first and second-order tensor matrices. The formula for this fusion is below:

$$T_F(ii',jj') = \begin{cases} 0 & \text{if } (i = j \text{ and } i' \neq j') \text{ or } (i \neq j \text{ and } i' = j') \\ T_1(i,i') & \text{if } i = i' \text{ and } j = j' \\ \lambda \cdot T_2(ii',jj') & \text{else} \end{cases}$$

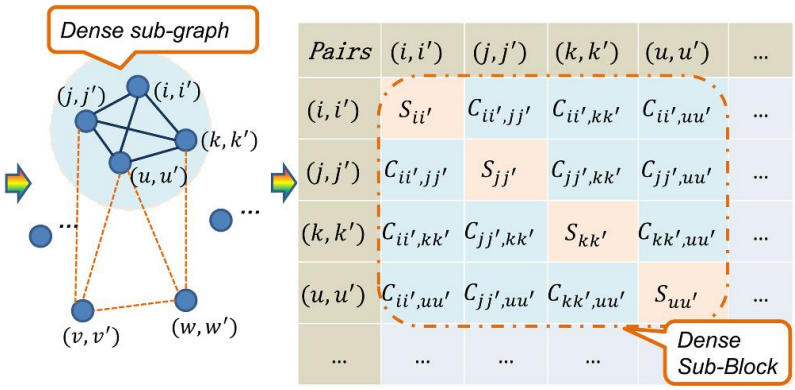
Figure 4.10.1.4 Minutiae Pair Function 4

The notation in the formula above should be understood at this point. The exception is the newly seen lambda symbol, which is meant to be set manually. This is due to the fact that T1 and T2 have different dimensions.The decision making in this formula needs to be explained. The value of 0 is given when (*i = j and i' ≠ j'*) or (*i ≠ j and i' = j'*) due to the fact that T_2 (*ii'*, *jj'*) is 0. The relationship between i and j is guaranteed to not be similar to the relationship between i' and j' in these instances because one comparison is between the same minutia and the other is between different minutiae. T_1 (*i*, *i'*) is the value when (*i = i' and j = j'*). This makes sense because you would be comparing the same minutia pair to itself, which is trivial.

An illustration of this process helps in the understanding. Below is a representation of the fused tensor matrix.



(a) A genuine match example
(Note: fingerprint images are from FVC2004 DB1 27-3 and 27-5)



(b) The correspondence graph (c) The minutia tensor matrix (*MTM*) T_F

Figure 4.10.1.5 Minutiae Tensor Matrix Example

The Tf minutiae tensor matrix is an (Np x Np') x (Np xNp') 2-dimensional matrix as each index for the rows and columns is a pair of minutia between minutiae sets p and p'. Np and Np' are simply the number of minutiae in each set. The Tf MTM is symmetric over the diagonal highlighted, which contains the T1 values from 4.10.1.4 formula. Again, the Dense Sub-Block is mentioned in this image. The idea behind this is that strongly connected clusters within the graph or Tf MTM will form where minutia pairs are correctly made. The similarity score between correctly paired minutia will definitely be high, and it is likely that pairs around them will have strong connections as well. The original optimization function 4.10.1.1 is adjusted to maximize the score for the new fused minutiae tensor matrix. The new version of the formula is below:

$$\tilde{M} = \underset{M}{argmax} \sum_{ii' \in M, jj' \in M} T_F(ii', jj')$$

Figure 4.10.1.6 Minutiae Pair Function 5

The notation of the formula above should be clear. The goal is to find the set of pairs that maximizes the similarity scores in the Tf MTM. The authors solve this

optimization problem with Spectral matching methods. The formula for this optimal solution is below:

$$\tilde{m} = \arg\max_{m} (m^T T_F m), m \in \{0,1\}^{|P|}, s.t.\ m1 \leq 1 \text{ and } m^T 1 \leq 1$$

Figure 4.10.1.7 Binary Vector Function

The figure above demonstrates the solution is a binary vector maximizing $m^T T_F m$. The paper contains a description of pruning and normalization of the first-order tensor matrix T1 that can be examined during implementation. The main steps of the process of finding the solution to the formula above are contained within algorithms that they provide. These algorithms are an essential part of any code writing that will need to be done if we would like to replicate or adapt their approach. The previous formulas and descriptions were essential to understanding the process proposed by the authors, but I am not here to summarize all contents of the paper. Below are the algorithms provided.

Algorithm 1 Spectral relaxation iterations.
Input: minutia tensor matrix *T*F
Output: *m**, the main eigenvector of *T*F
1 initialize $m^* = m_0 = \bar{T}_1$
2 repeat
3 ↓*m** ← *TFm**;
4↓ $m^* \leftarrow \frac{1}{\|m^*\|_2} m^*$;
5 until convergence;

The authors stated that this algorithm generally only needed to repeat around a dozen times before convergence occurred. Also, they gave it a complexity rating of O (z), with z being the number of non-zero elements in the matrix. Previous pruning means that this will not be very expensive.

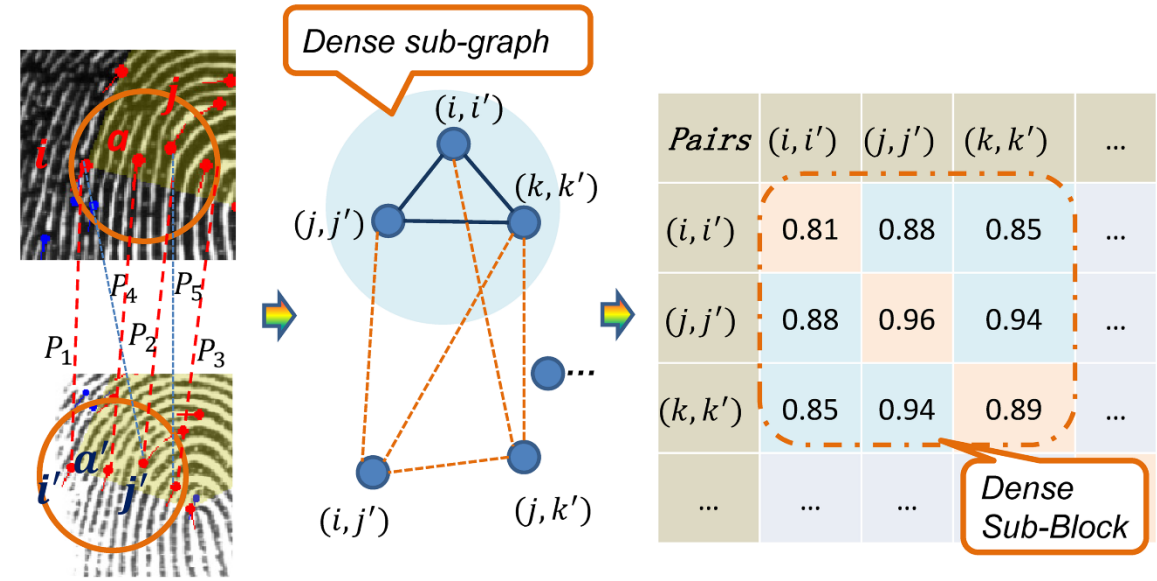
Algorithm 2 Greedy approach for discretization.
Input: continuous matrix *m**
Output: binary matrix $\tilde{m}$
1 initialize $\tilde{m} = 0$
2 set *m** (*i*,*j*) = 0, ∀i, ∀j, m*(*i*,*j*)< δ
3 repeat
4 ↓ find the maximal element m*(*i*,*j*)
5 ↓ set$\tilde{m}(i,j) = 1$, ∀k, set *m**(*i*,*k*) = 0,*m**(*k*,*j*) = 0
6 until *m** = **0**;

At this point the final similarity score between the local minutiae sets can be calculated. For minutiae sets P and P' they provide the following formula:

$$S_{pp'} = \tilde{M} \cdot N_m^2 / (N_p \cdot N_{p'})$$

Figure 4.10.1.8 Minutiae Pair Function 6

In this formula, Nm is the number of matched pairs, and Np and Np' are the numbers of minutiae in sets P and P' respectively. Practically the same concept and process is applied to generate the global minutiae tensor matrix. Some of the formulas need to be modified, but the formulas are close enough to those used for the local MTM that there is not much explanation necessary. Referencing the original paper would be more appropriate to see the differences.
The following images are illustrations of this process for local and global MTMs. The similarities should be clear. Basically, the global MTM is doing the same process, but comparing pairs of local minutia topological structures as opposed to just minutia pairs.
It should also be noted that the global minutiae tensor matrix cannot be constructed prior to the creation of the local minutiae tensor matrix. The global minutiae tensor matrix depends on the LMTS for its similarity scoring, and the LMTS are determined through the scoring system used at the local level.



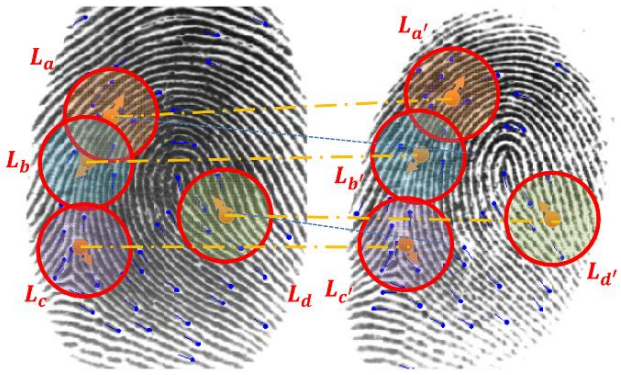
(a) A local topologic structure pair (b) The local correspondence graph (c) The local minutia tensor matrix LT_F

(Note: fingerprint images are captured from FVC2004 DB1 6-2 and 6-5)

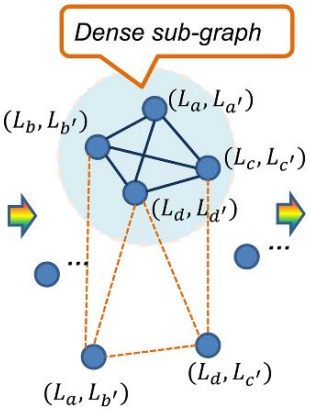
Figure 4.10.1.9 Minutiae Extraction Process

Above is the demonstration of the extraction of correct minutiae pairs [36]. The local minutia topological structures [36] are represented graphically and then in the local MTM. As the original paper states, the dense subgraph and dense block from the MTM represent correct minutiae pairs. Local minutia topological structures are basically just smaller sets of fingerprint minutiae within a fingerprint image. The set is focused around a center minutia and the minutiae surrounding it in a small area are grouped together.

Global Minutia Tensor Matrix:



(a) A genuine match example
(Note: fingerprint images from FVC2004 DB1 27-3 and 27-5)



Pairs	$(L_a, L_{a'})$	$(L_b, L_{b'})$	$(L_c, L_{c'})$	$(L_d, L_{d'})$	...
$(L_a, L_{a'})$	0.89	0.92	0.81	0.85	...
$(L_b, L_{b'})$	0.92	0.78	0.87	0.96	...
$(L_c, L_{c'})$	0.81	0.87	0.94	0.88	...
$(L_d, L_{d'})$	0.85	0.96	0.88	0.91	...
...	...	...	...	...	

Dense Sub-Block

(b) The global correspondence graph (c) The global minutia tensor matrix GT_F

Figure 4.10.1.10 Topological Structure

The illustration above should look familiar. Same concept as with the local MTM, but now applied on pairs of the local minutia topological structures. Now with the whole methodology for minutiae matching in mind, we have the final unifying algorithm. This was provided by the authors and is essential to implementation in code.

Algorithm 3 High-order tensors matching algorithm.
Input: Two fingerprint minutia sets *P* and *P'*, with $\boldsymbol{N}_p$ and $\boldsymbol{N}_{p'}$ minutiae, respectively.
Output: The entire similarity $\boldsymbol{S}_{PP'}$ of minutia sets ***P*** and ***P'***
1 Initialization:
constructing local minutia topologic structures:
$\boldsymbol{N}_p$ and $\boldsymbol{N}_{p'}$structures, respectively.
calculating similarities and compatibilities of all minutia pairs:
$\boldsymbol{N}_p \times \boldsymbol{N}_{p'}$ similarities and $(\boldsymbol{N}_p \times \boldsymbol{N}_{p'}) \times (\boldsymbol{N}_p \times \boldsymbol{N}_{p'})$ compatibilities.
2 Local matching: calculating *LMTS* similarities

repeat: from 1 to $\boldsymbol{N}_p \times \boldsymbol{N}_{p'}$
calculating similarity of *LMTS* pairs for *LMTS* pairs (L_a,$L_{a'}$)
building the local first-order tensor $\boldsymbol{LT}_1$ and the local second-order tensor $\boldsymbol{LT}_2$
constructing the local minutia tensor matrix $\boldsymbol{LT}_F$
spectral matching for $\tilde{LM}$ and matched pairs
calculating the similarity $S_{L_a L_{a'}}$ of *LMTS* pair (L_a,$L_{a'}$)
until all candidate *LMTS* pairs are calculated.

3 Global matching: calculating entire similarity
building the global first-order tensor $\boldsymbol{GT}_1$and the global second-order tensor $\boldsymbol{GT}_2$
constructing the global minutia tensor matrix $\boldsymbol{GT}_F$
spectral matching for $\tilde{GM}$ and matched pairs
regaining lost genuine minutia pairs using *MLS* model
constructing the convex hulls of minutia sets ***P*** and ***P'***.
calculating the entire similarity $\boldsymbol{S}_{PP'}$ of minutia sets ***P*** and ***P'***

4.10.2 Conclusion and Considerations

If we really want an effective system for matching minutiae after extraction with our neural network, this seems like the best option so far. The drawback is that we could not really implement this whole algorithm when training our recognizer. It would take a huge amount of time, and the neural network would find little success to learn from. This could potentially be put in place near the end of the project to have a very accurate and robust authentication process.

There is also the potential to scale down the algorithm to incorporate it into the recognizer early on. Once the recognizer can extract minutiae within somewhat lenient areas of proximity, maybe we could then take the next step of training it to accurately extract the local minutiae topologic structures described above.

At the very least, we could use it as final test to demonstrate that our neural network extracts minutiae effectively. FingerNet used this algorithm similarly to demonstrate how their extraction methods compared to existing ones.

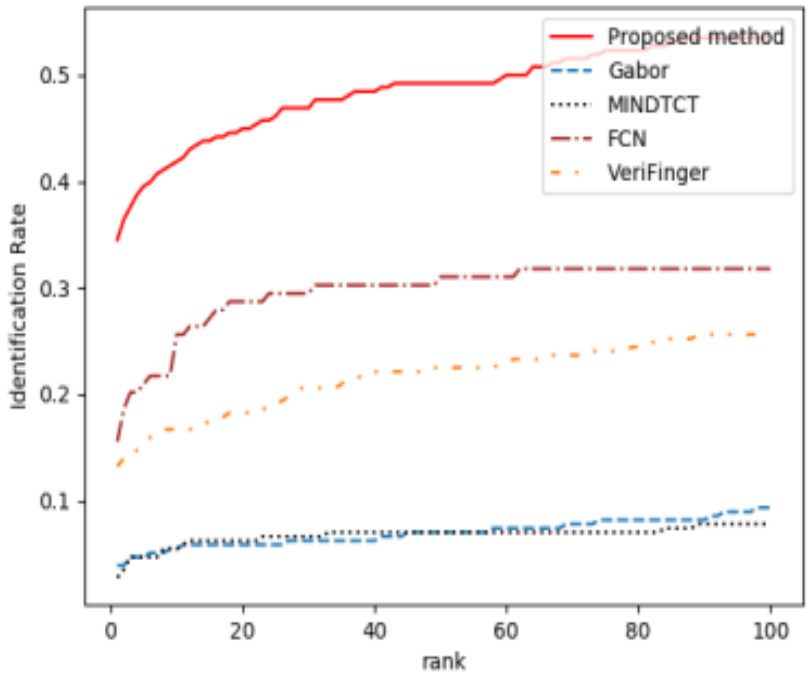


Figure 7. Identification performance (CMC curves) of different minutiae extraction algorithms on NIST SD27.

Figure 4.10.2.1 FingerNet Minutiae Extraction vs Other Existing

4.11 N.E.A.T. for GANs

NEAT for CNNs is a process for where both the accuracy and topology of the network is left up to training. In a normal CNN, the design of the neural network is left up to the programmer, and a different topology can have dramatically different outcomes after training. NEAT, however, allows the training algorithm to add and remove neurons as it feels is necessary to assist further accuracy of the model.

There are a number of papers on NEAT, but not one applies it to Generative Adversarial Networks. One avenue of approach to an attack could be to start with a relatively simple network and allow training to add a neuron whenever training seems to stall with a low accuracy. Would such a method allow even an undirected attack to bypass security measures that an undirected attack usually fails at? If we have time, we may attempt this approach.

So what is NEAT? It stands for the Neural Evolution of Augmented Topologies. Normally, when a neural network is trained, the topology is set at the start, and the network will eventually reach a peak accuracy. NEAT allows the training algorithm

to detect when the network is starting to stagnate, and track which connections are failing to learn the data well. Then, it breaks those connections by adding a new interconnected neuron between them in order to hopefully improve the accuracy. Neurons which do not contribute well to the final output may also be removed or reconnected in new ways to improve their contribution and overall network accuracy. [37]

Not only does NEAT build up its structure over time, but it will build separate competing structures to see which one performs the best at the end. This is very similar to Genetic Algorithms approach, but differs by keeping these species separate, and not crossing them to hopefully get a new one. As a result, you may have hundreds of possible networks by the end, but you keep the one that performed the best out of all of them. [37]

Unfortunately, NEAT can be difficult to implement, but thanks to Kenneth Stanley, Ph.D. at UCF, there is a library with NEAT already implemented for very large networks. Furthermore, he may be available to advise us in use of this library, or at least point us to graduate students to do it in his stead. This makes attempting NEAT a definite possibility.

5. High Level Design

Figure 5.1 outlines the current state of our project's workflow. Due to the nature of this project, there are many different paths to take for each particular portion. Due to this, much of the project is split equally between the team members, so each team member can take a different approach.

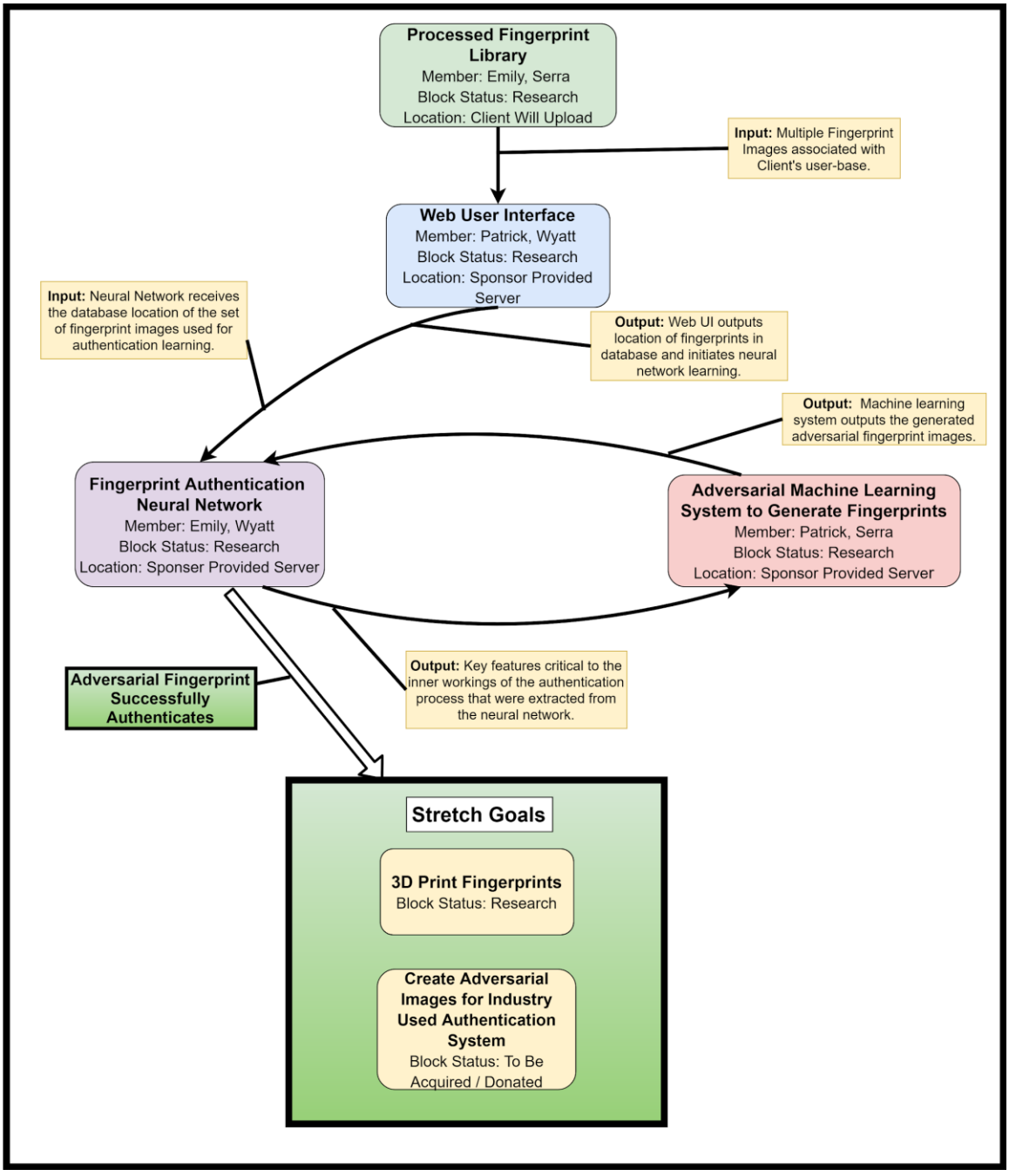


Figure 5.1 High Level Architecture Overview

The project initially starts with the fingerprint library, which the user will upload. This fingerprint library will consist of fingerprint images that the user wants a synthetic print based off of. This data needs to be processed into a form that will be usable in both the authentication neural network and the generative adversarial networks. In Section 4. Research and Investigations, we explored various ways to conform these images into the needed format. As we design the authentication network in Senior Design II, the specific format will be formalized. From there, the website (website architecture can be found below) will store the fingerprints into the database and initiate the learning process on the fingerprint authentication neural network. Depending on the authentication network we use, the fingerprint images will either be directly passed to the neural network, or the minutiae (ridges, edges, etc.) of the fingerprint will be extracted and sent to the authentication neural network. Once the NN has learned the data set provided by the client, the process of training the generative adversarial network will begin.

ecuOnce the generator can effectively create images that the discriminator classifies as genuine, the adversarial fingerprints are tested on the authentication system and regenerated if they are declined. At the end of this authentication-regeneration process, we intend to have a skeleton key that will be able to validate itself as one or many actual user. The rate of success for this particular part of the process will largely be dependent on the accepted rate of error allowed in the fingerprint authentication system. Various common error rates will be provided in the web user interface.

Depending on the point in time at which we successfully and consistently create a skeleton key for a given authentication system, we can choose to pursue our stretch goals. As mentioned previously in the document, these stretch goals consist of 3D printing our master keys and testing our adversarial fingerprint generator on an industry standard fingerprint authentication system.

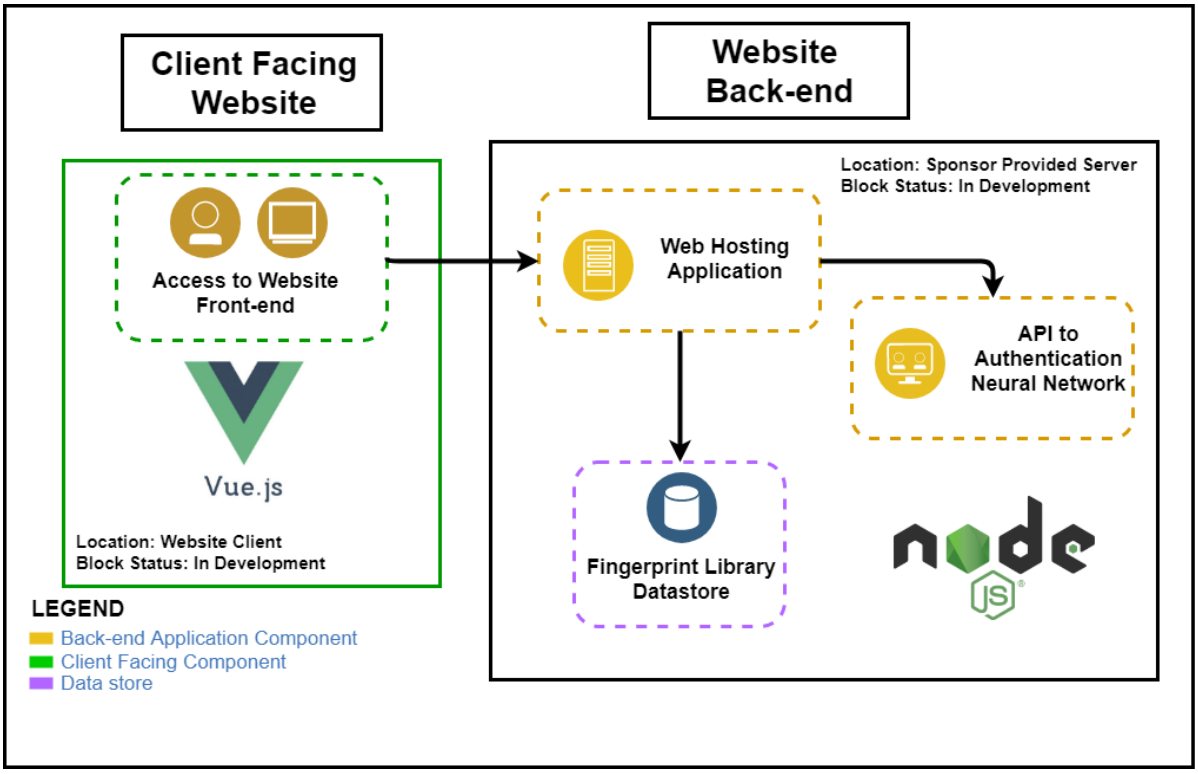


Figure 5.2 Web User Interface Architecture

Figure 5.2 indicates the plan four our website architecture. The front end must allow the user of the system to login to their associated account and upload their collection of fingerprint images. The user authentication is absolutely necessary due to the nature of this project. Each fingerprint will be associated with a specific user and the id of that user. Since the user is providing data that our neural networks will be trained on, this operation has to be asynchronous. This is due simply to the amount of time it takes to train these networks. Our neural networks will have to run in the background and users will have to be notified at a later time that their data has finished processing.

Again, in the client space, they will also need to have the ability to retrieve adversarial fingerprint images as well. Having user authentication would allow each user to have private access to their newly generated fingerprints. This also means the web user-interface will mainly consist of a portal where the users can manage and view their uploaded fingerprints, as well as view their current running jobs.

Once the client initiates the upload of their fingerprint library, the flow of control redirects to the website backend. The website backend is responsible for handling the distribution of the user's fingerprint images, as well as initiating the training process. The fingerprints will be stored on the server, and the appropriate entries for the image locations will be made in the database.

Once the data is successfully mapped in the database, the backend API will trigger the learning process. At this point, the web user-interface will notify the user that the job has started. Since this particular part of the process takes a significant amount of time, we will have functionality which will email or message the user once the fingerprints have been successfully created.

6. Detailed Design

6.1 Generative Adversarial Network

The GAN will be coded in Python 3.5 with TensorFlow and Keras in the Anaconda environment. It will use the Wasserstein GAN and loss function as a template, with a generator neural network and a discriminator neural network. The generator neural network will feature 12 layers. These layers will be applied in order as follows:

1. Convolution
2. ReLu
3. Max Pooling
4. Convolution
5. ReLu
6. Max Pooling
7. Convolution
8. ReLu
9. Max Pooling
10. Convolution
11. ReLu
12. Max Pooling

The discriminator will apply these layers in the opposite order. Visualization plots will be handled by the pyplot library to import graphs into the web interface. The TensorFlow model will be exported as a Protobuf (Protocol Buffer) file to be uploaded onto the server. This file will be used to access the generator and discriminator functions.

6.1.1 Minutiae Detection for Discriminator Training

Before starting training on the CNN or GAN, fingerprints will be deployed to the appropriate directory, and a python script will be run to extract the minutiae. This script will extract the minutiae using SUSAN corner detection, and it will extract the best 20 corners from the smallest area of the fingerprint possible. It will output those minutiae into a minutiae directory under the same base file name with the extension ".mnt". It will output the fingerprint image into the orientation directory. Finally, then it will post a binary image of the fingerprint in the segmentation labels directory where the white is the area that the minutiae were taken from.

To find the best minutiae we will find all the minutiae for the whole image and do a mean shift search for the highest concentration of good corners. We will start with a window of 512x512 pixels, and then then after it finds the area that best fits, it will use a binary search to determine if growing or shrinking the window produces better results. Finally, it will select the best 20 corners from this area, as those are

most likely to be the ones which fit actual minutiae of the print and are most easily detected by a computer.

It doesn't matter if the algorithm used has false positives for minutiae. The reason behind this line of thought is that we aren't using all of the corners found. The weaker corners are more likely to be false positives than the stronger ones, and even if we get a false positive or two, we may be learning a part of the fingerprint that could result in better security. If we find that the minutiae need to be improved, we can further refine the algorithm as we go.

The same algorithm can be used for both the CNN and the GAN because mean shift often can return a different area depending on the starting point, so in a graybox test where the GAN doesn't know the precise minutiae found, it might receive a different result.

In the situation of the white box testing the minutiae finder can be run once and then it's results can be copied verbatim to the GNN.

6.1.2 Fingerprint Datasets

Initially when we started researching this project, we looked into using the fingerprint datasets from the National Institute of Standards and Technology. Prior to the start of this project, NIST provided a comprehensive selection of fingerprint databases. However, unfortunately, these databases have been removed since the datasets "lack the documentation required by NIST for distribution." We have engaged in a email correspondence with NIST inquiring about the eventual availability of these datasets. According to Patricia Flanagan, who is responsible for these datasets at NIST, she expects that these datasets will potentially become available again within two months time. Once we start implementing this project in Senior Design II during the fall, we will evaluate the availability of these datasets and use them accordingly.

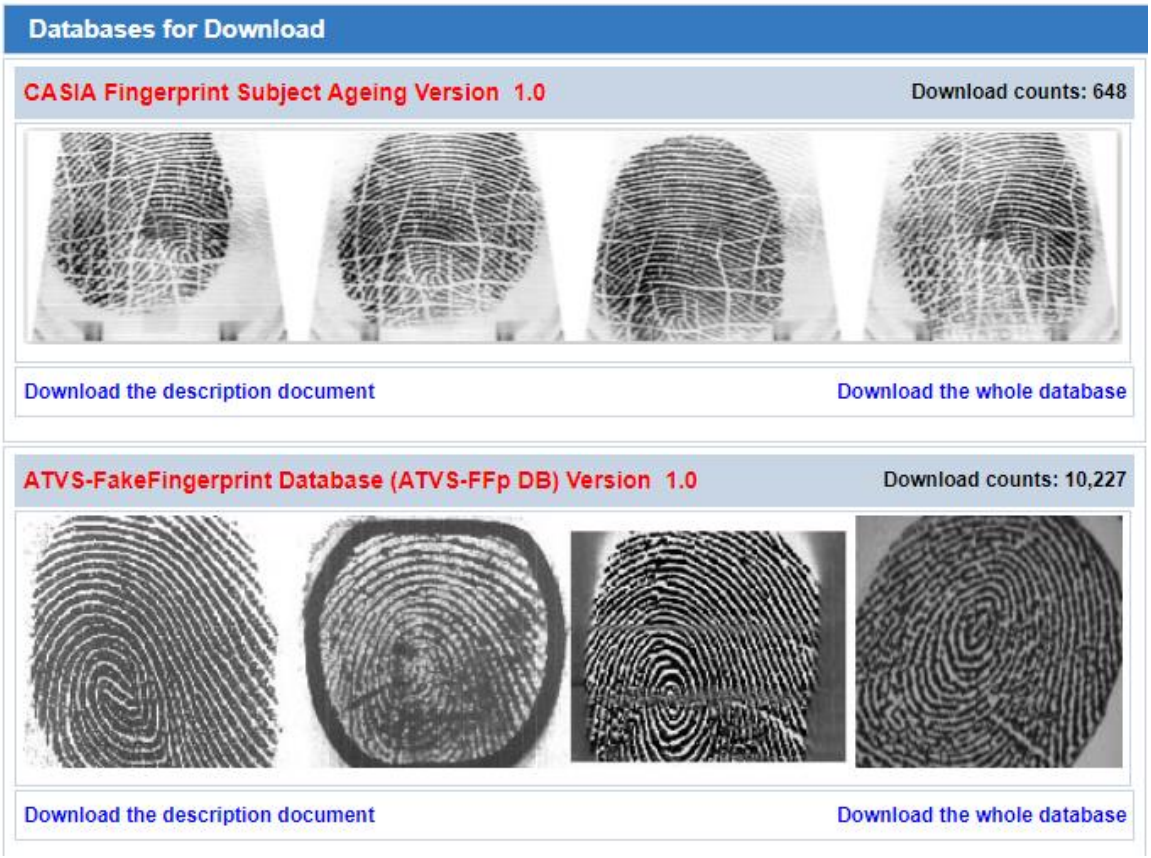


Figure 6.1.2.1 Sample Images from Biometrics Ideal Test Dataset

For the time being, we have found several different datasets that will work for our purposes. Our sponsor, Dr. Jha recommended we look into the datasets provided by the Biometrics Ideal Test. The Biometrics Ideal Test is sponsored by the National Basic Research Program of China. They provide several fingerprint datasets, some consisting of fingerprints from users at different points in their life, while other provide fake fingerprints for use in general testing. These datasets also contain fingerprints scanned from multiple different sensors with some being capacitive and the others are standard rolled fingerprints. Having the multiple formats will be useful as we test our neural networks' effectiveness.

	Sensor Type	Image Size	Set A (wxd)	Set B (wxd)	Resolution
DB1	Optical Sensor	388x374 (142 Kpixels)	100x8	10x8	500 dpi
DB2	Optical Sensor	296x560 (162 Kpixels)	100x8	10x8	569 dpi
DB3	Capacitive Sensor	300x300 (88 Kpixels)	100x8	10x8	500 dpi
DB4	SFinGe v2.51	288x384 (108 Kpixels)	100x8	10x8	about 500 dpi

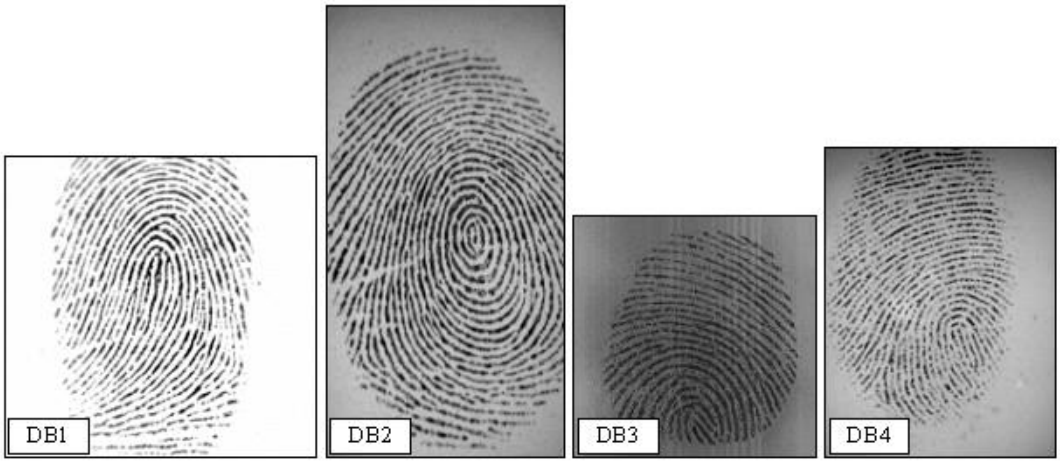


Figure 6.1.2.2 FVC2002 Dataset Examples [38]

We have also found the datasets provided by the Fingerprint Verification Competition. The Fingerprint Verification Competition is a competition based on implementing the techniques discovered in the various advances of fingerprint authentication research. This dataset is fairly comprehensive. Two of the datasets utilize optical sensors for scanning the fingerprints, while the other dataset uses a capacitive sensor. There are 110 unique fingerprints with 8 impressions per finger. Compared to some of the other datasets we have looked at, this particular dataset seems to provide a quality collection of fingerprints, with minimal artifacts in the images. Some of the other datasets we were considering were not as complete as this one, or the images were not of the same quality.

6.2 API calls

The API calls that the web interface will make to the GAN are as follows:

- startGeneration()
  - Argument(s): file array of JPG images
  - Return: None
- checkProgress()
  - Argument(s): int GAN identification number
  - Return: bool true/false
- updateFingerprintData()

- Argument(s): int GAN identification number
- Return: image file

- fetchPlotData()
  - Argument(s): int GAN identification number
  - Return: string JSON serialization

The call startGeneration() will send the image data to start the GAN in the background. It will not return any data, since this will be fetched by checkProgress(). The checkProgress() will fire when a user accesses the web interface and checks whether the GAN with the matching identification number has completed its process. The call getFingerprint() will fire if checkProgress() returns true and update the screen with the master fingerprint image. When the user accesses the “Details,” if checkProgress() returns true, the fetchPlotData() will fire and update the screen.

6.3 Server Structure

The server is provided by Dr. Sumit Kumar Jha. It is running Apache 2.4.18 with Ubuntu, and it’s hosted on Amazon AWS.

The server will host all the files for the website and the corresponding files for the GAN. It receive requests through the web interface and run the GAN with the correct input from the database. It will host the Protobuf file that will be used to access the classes of GAN (gan_model.pb). The structure of the server is as follows:

Note: this is only an example. Not every file is included in the list.

```
./GAN_deploy_container
        > gan_model.pb
        ./GAN_deplot_container/variables
                > data_matrix_1.npy
                > data_matrix_2.npy
./graphics
        > banner.jpg
        > footer.jpg
        > etc...
./web
        > robots.txt
        > index.html
./web/css
                > style.css
        ./web/javascript
                ./web/javascript/anchor-js
                        > anchor.js
```

6.4 Website User-Interface and Backend

6.4.1 Web User-Interface

Throughout our research this semester, we evaluated many possible options for the web user-interface. In regards to the front-end aspect, this is no shortage of possible frameworks we could pursue. After much deliberation, we have decided to go with Vue.js as our frontend framework.

Figure 6.4.1.1 Vue.js Logo

Vue.js provides several key features which are extremely beneficial for this project. For one, when compared to Angular and React the learning curve for Vue.js is rather minimal. Since no one on our team has previous experience with web development, Vue.js will allow us to rapidly prototype our website. Vue.js's ease of use also allows for the team to focus its efforts more on the core of this project (neural networks). The documentation associated with Vue.js is also more than sufficient for our purposes.

Our user interface will act as a dashboard so the user can see information about their current running jobs, the data that they have uploaded, as well as any progress pictures and analytics associated with the training process. Since the training process does take some time, we would like to incorporate progress checks. These progress checks would reveal information such as the current rate of error within the generative adversarial networks, as well as current samples from the generated image set. Since Vue.js is a reactive framework, the pages associated with these results will automatically update as the data on the server changes.

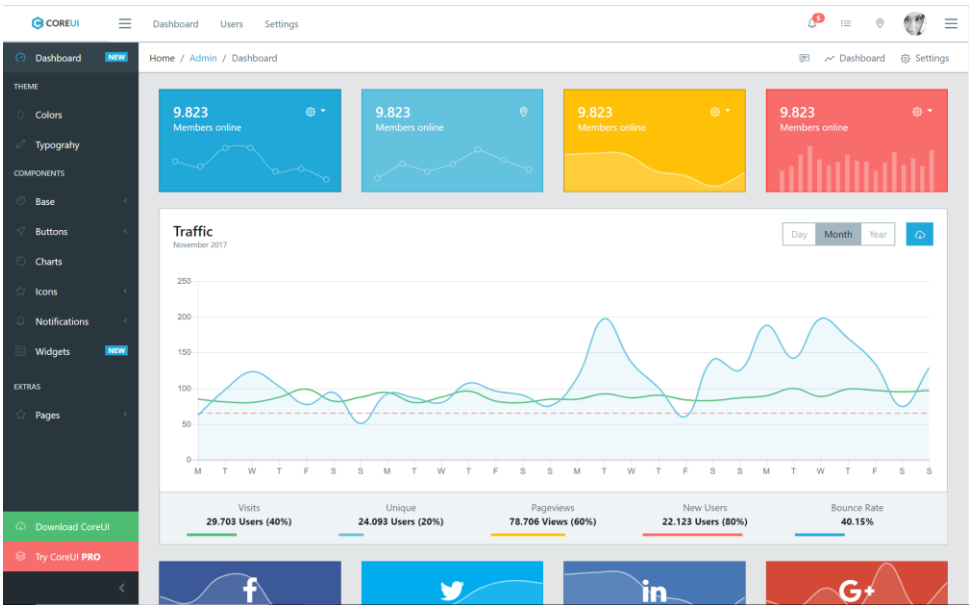


Figure 6.4.1.2 CoreUI Live Demo [39]

We also have decided upon using the CoreUI template provided by Lukasz Holeczek. Core UI is an admin template licensed under the MIT license and based in Bootstrap and Vue.js. The CoreUI Vue.js template provides a very modularized shell that our team can build off of. Each element is designed in a responsive manner, and it includes various utilities for providing items such as charts for data analytics. CoreUI also provides functionality for notifications to the user within the administrative portal, and since it's using Node.js, we can call our needed training modules easily from the web application.

Based on the design provided by CoreUI, we anticipate the web user-interface to resemble the following mockups:

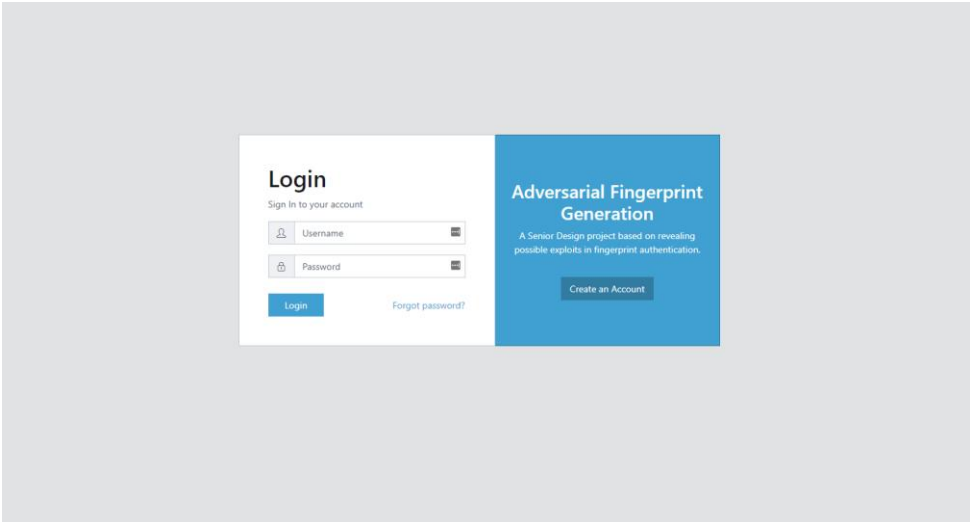


Figure 6.4.1.3 Login Page using CoreUI

There will be two main pages that the user will encounter once they visit our site. One will be the homepage, which will describe the project as well as the team members. As depicted above, the users will also be brought to this login portal. From there, the users will have access to their individual dashboard which will allow them to upload new data, start training jobs, and view the results and analytics associated with their completed jobs.

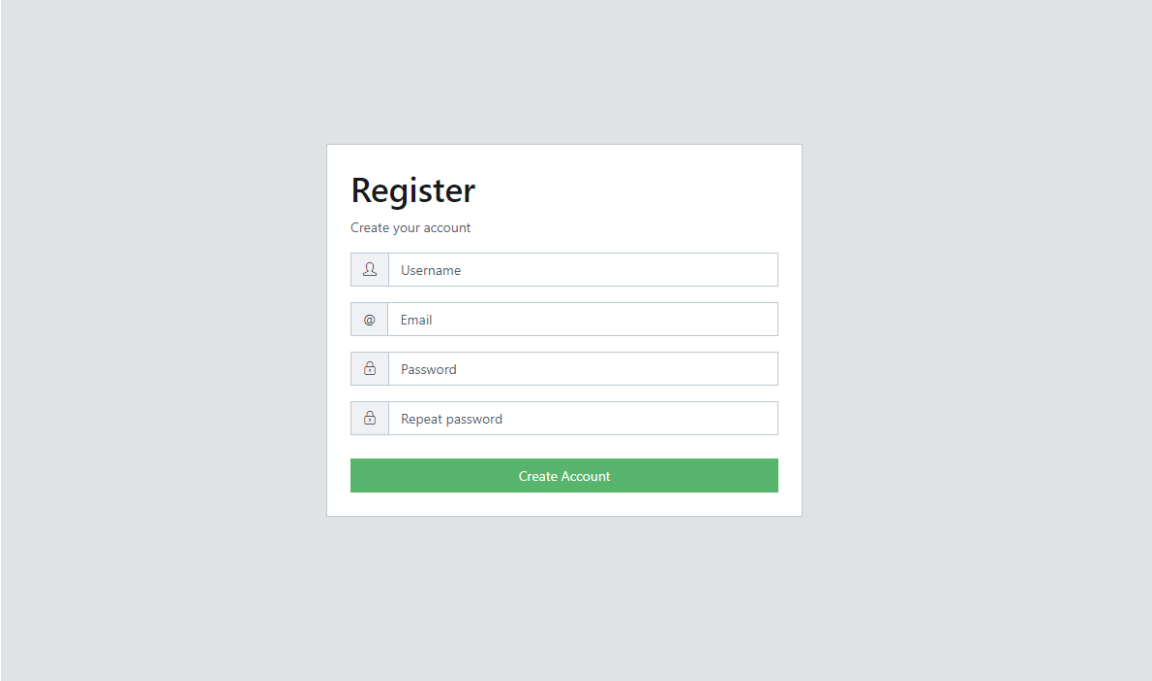


Figure 6.4.1.3 Sample Registration Page

Similarly, the users will be able to register directly through the web user interface.

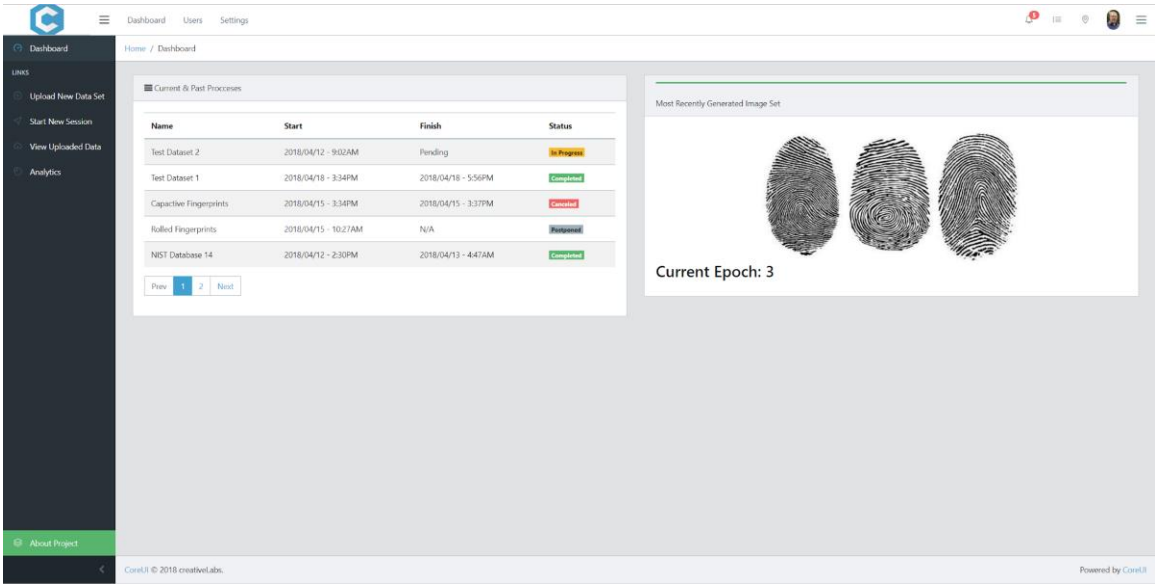


Figure 6.4.1.4 Dashboard That User Arrives Upon Login

Pictured above is the outline for our dashboard. Once the user has successfully created an account and logged in, they will be brought to this particular page. The dashboard acts as the main hub of information for the user. On this page, the website will have information regarding the user's current running processes, as well as their past processes. This will show useful information such as the processes' current status, its start and end times, as well as the name of the dataset associated with the process. In another view on the dashboard, the user will see the latest image created by our generative adversarial networks. In this section it will give progress information like the current epoch, as well as data that is specific to the stage of the training process that the dataset currently is in. On the left-hand side of the page, the user can go to different parts of the website through the navigation bar. Here they can access pages where they will be able to upload a new dataset, view previously uploaded datasets, and go over the analytics associated with their various finished jobs. Along with the analytics associated with each job, the user will be able to access and download the synthetic fingerprints created for each dataset.

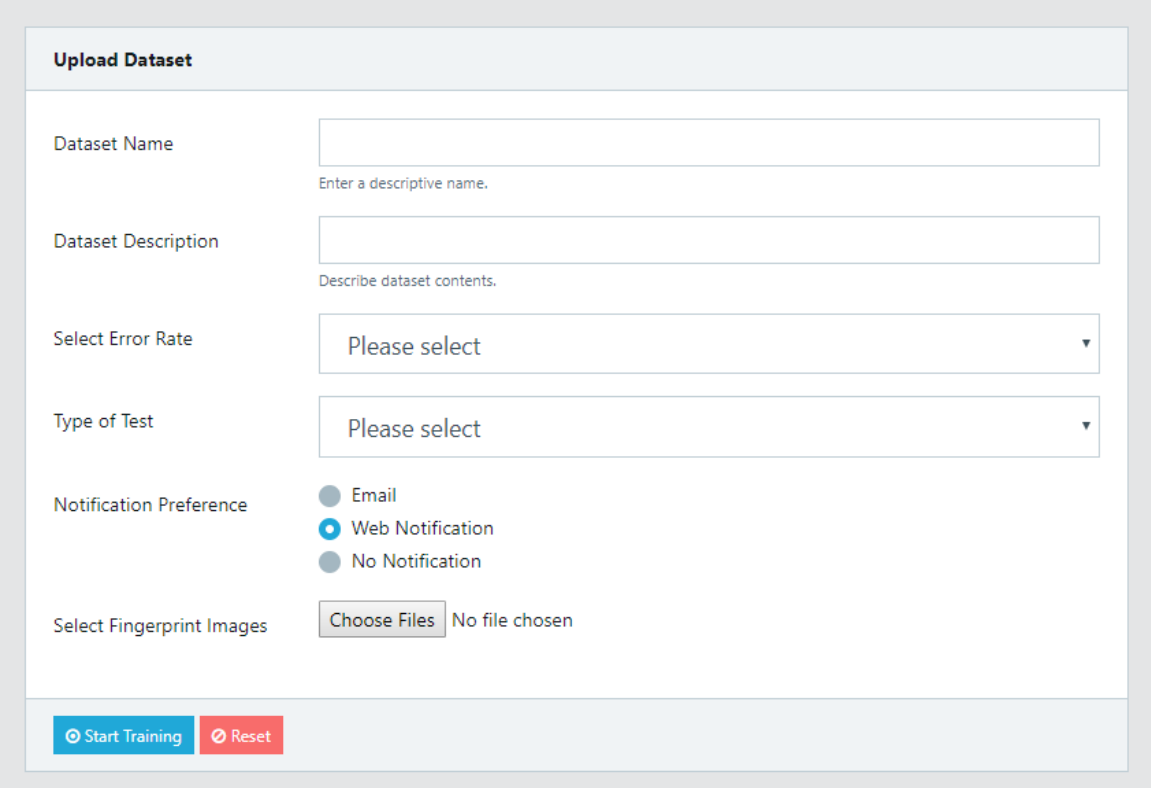


Figure 6.4.1.5 User Dataset Upload

The above image represents the page the user will be directed to once they want to upload a dataset. CoreUI conveniently provides various form templates that are customizable. This form will allow the user to input the necessary information regarding their dataset (name, description, authorization algorithms). It will also allow the user to make choices. For instance, the user might want to select a different error rate for testing the authorization neural network or they may want to decide exactly how much data they want to feed the generative adversarial networks. Once we start implementing the project in Senior Design II, the particular file protocol which will be used to transfer the dataset will be determined. For now, however, the above form allows the user to submit a folder of images and all will be transferred based on the mechanism defined in our website backend.

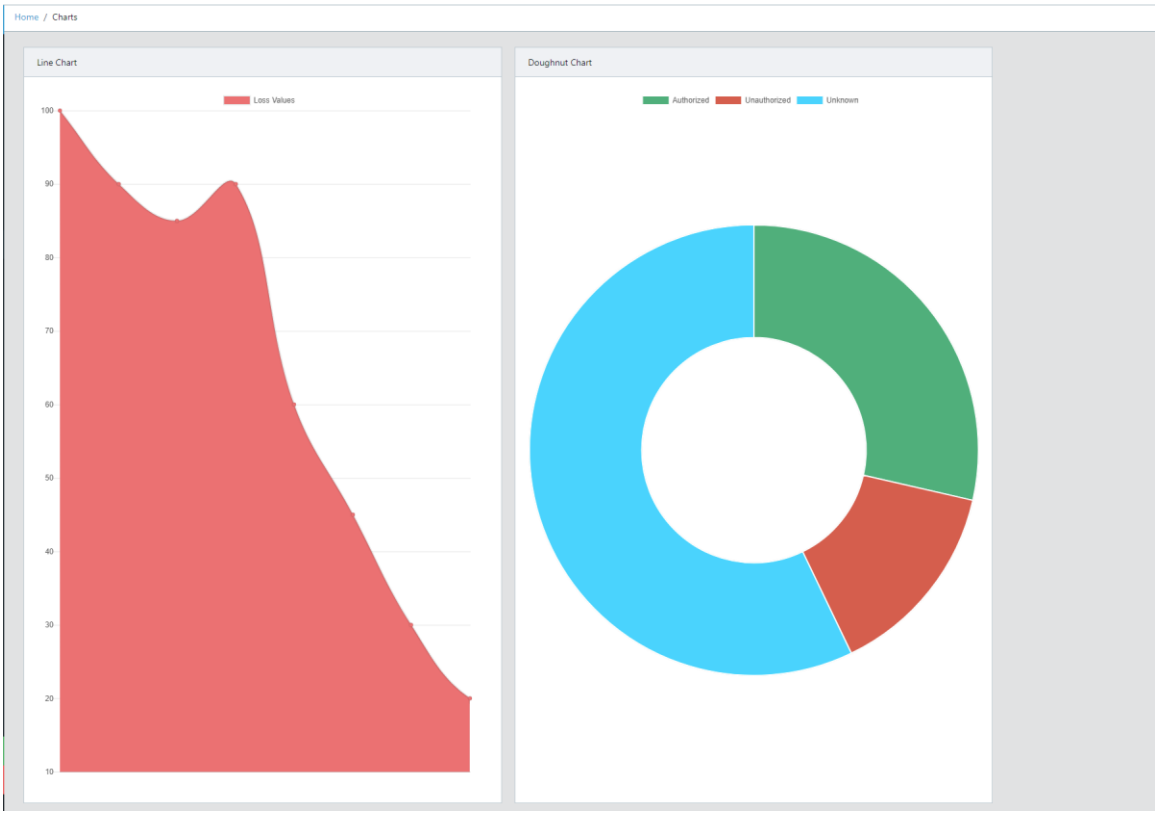


Figure 6.4.1.6 Analytics Page using CoreUI

The above diagram is a possible layout for the analytics and statistics portion of our web portal. CoreUI, along with Vue.js, come with several premade scripts that can be used to show data in various forms. We will be able to pass data in between our Python and Tensorflow based training process and our Vue.js based web frontend. This will allow the user to see the progress of the job as it is completed on the server end.

6.4.2 Web Backend

For the backend portion of the website, we have made a few key decisions that we believe will provide many benefits and conveniences to this project.

Figure 6.4.2.1 Node.js Logo [40]

We decided that the web server for our project should be Node.js. Especially considering that everyone in the group is new to web development, Node.js provides a few key features that make it very convenient to use. For instance:

- Node.js uses the same language as our frontend framework: Javascript. This means code can be shared between the frontend and the backend.
- Using Node.js allows us to easily keep a persistent connection between the backend and the client, supplying updates as necessary in real time. This is done without creating a sizeable load on the server.
- Node.js allows us to asynchronously create a child process. This will be rather convenient when we have to spin up our various Python based scripts depending on what the user requests.
- Has convenient dependencies and plugins for supporting a large variety of databases

Figure 6.4.2.2 Express JS Logo [41]

Complementing Node.js, we will also be using Express to manage a variety of common tasks associated with our website. Express works to simplify actions like creating RESTful APIs, managing sessions, handling errors, and parsing payloads in Node.js. This functionality can be completed using Node.js alone, however, it would take significantly more effort to do so.

Both Node.js and the associated framework Express will be installed on our sponsor provided server. On the server, the NPM package manager will also be installed. The NPM package manager allows us to easily gather the dependencies needed both for our frontend frameworks, as well as those needed for the backend software listed above.

6.4.3 Database

In making decisions for the final backend design we had to choose which database we would be implementing. Our first consideration was to use a non-relational database such as MongoDB. The reasoning behind this was that we would working with varying amounts of data in various formats without a very clear structure. The users of the web interface would be able to upload fingerprint data that could be capacitive or complete, and with missing annotations depending on the source. We ultimately decided that we wanted a database that was well documented and

could also easily be used for relational data. This was because for our own training purposes, we would likely be working with complete information with already generated annotations on minutiae, orientation, and segmentation. The clear choice to satisfy all these needs was PostgreSQL.

Figure 6.4.3.1 PostgreSQL logo [42]

30.19. Example Programs

These examples and others can be found in the directory `src/test/examples` in the source code distribution.

Example 30-1. libpq Example Program 1

```
/*
 * testlibpq.c
 *
 *      Test the C version of libpq, the PostgreSQL frontend library.
 */
#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,
                j;

    /*
     * If the user supplies a parameter on the command line, use it as the
     * conninfo string; otherwise default to setting dbname=postgres and using
     * environment variables or defaults for all other connection parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname = postgres";

    /* Make a connection to the database */
    conn = PQconnectdb(conninfo);

    /* Check to see that the backend connection was successfully made */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
```

Figure 6.4.3.2 Example PostgreSQL code from their documentation [42]

The above image is taken from the PostgreSQL documentation site. Example programs and information like this are extensive and easily accessible. The above code demonstrates how to connect to the database based on information from the command line. This particular code segment is written in C.

We will be using Python to connect with the database. There are multiple options for python drivers to connect with PostgreSQL but not all of them are still supported. Below is a chart showing our options for drivers with information about their release dates, platforms, versions, notes about support, and some other important info.

Software	License	Platforms	Python versions	DB API 2.0	Native (uses libpq)	Last Release	Notes
Psycopg2	LGPL	Unix, Win32	2.6-3.6	yes	yes	2017	Most popular python driver, required for most Python+Postgres frameworks
pg8000	BSD	any (pure Python)	2.7+ / 3.3+	yes	no	2017	Used by Web2Py. current updated official site
py-postgresql	BSD	any (pure Python)	3.0+	yes	no	2012	pure Python with optional C accelerator modules, extensive custom API. Python 3 only.
PyGreSQL	BSD	Unix, Win32	2.6 thru 3.6	yes	yes	2017	The first PostgreSQL adapter for Python. Still actively maintained.
ocpgdb	BSD	Unix	2.3-2.6	yes	yes	2010	PG8.1+
bpgsql	LGPL	any (pure Python)	2.3-2.6	yes	no	2009	labeled alpha

Figure 6.4.3.3 Chart with different options for python drivers to connect to PostgreSQL database [43]

Out of the options for Python drivers, Psycopg2 seems to be our best bet. Psycopg2 works on a Unix or Win32 platform. Psycopg2 seems to be the most popular choice for a Python driver. It is required for most Python+Postgres frameworks according to the chart. It is still widely used and supported. The following is an example of Python code to connect using the Psycopg2 driver.

```
#!/usr/bin/python2.4
#
# Small script to show PostgreSQL and Pyscopg together
#

import psycopg2

try:
    conn = psycopg2.connect("dbname='template1' user='dbuser' host='localhost' password='dbpass'")
except:
    print "I am unable to connect to the database"
```

Figure 6.4.3.4 Example code for connected to PostgreSQL with python [44]

Psycopg2 is actually very user and beginner friendly in terms of connecting to and communicating with the database. Psycopg2 inherently has protections and security features built into the driver. For example, you cannot drop a database from within a transaction.

```
var pg = require('pg');

var connectionToDb = "postgres://username:password@localhost:5432/Database1";

var dbClient = new pg.Client(connectionToDb);

var query = dbClient.query("SELECT * from fingerprints");

query.on('row', function(row) {
    console.log(row);
});
```

Figure 6.4.3.5 Example Connecting and Query [44]

The above code shows how to connect to a PostgreSQL database using Node.js. This is based upon the pg package that can be located using the following command:

npm install pg

This package creates the connection between Node.js and the database while providing all of the needed API routes to sufficiently retrieve all the data we need from the database.

7. Build and Prototype

7.1 Preliminary Ideas

7.1.1 Fingerprint Authentication Neural Network

- When selecting a fingerprint library to start from, I suggest that we look into datasets that contain capacitive fingerprint images. Capacitive sensors are used on modern smartphones (Ex: Latest iPhone), and since the sensors are smaller than one's actual finger, it takes multiple images of your finger. This can lead to easier exploitation since an adversarial fingerprint only has to match one of the multiple images it captures. (Wyatt)
- We should also evaluate what rate of error most fingerprint security systems allow for when scanning fingerprints. Depending on this rate of error, we can recognize that some types of fingerprint authentication systems can more easily be exploited than others. This could help direct what standard our authentication system should set. (Wyatt)
- There are 2 potential ways to develop a recognizer. The first is to have a recognizer that will recognize any print in the database and tell you who it is. The second is to have a recognizer who is told who the print belongs to, and it simply replies whether the print matches. The advantages of the former is that it would be easier to fool, but the latter would be more secure in real world settings and more likely what we would face. (Mel)
- Coding of the machine learning components can be done in virtually any language. While the TensorFlow libraries were written in C++, they can be included in C++, C#, Python, or even Java. As a result, we can choose the language best suited for our task. (Mel)
- We can think about allowing multiple fingerprints to unlock a device, similar to the iOS and Android OS fingerprinting systems where you may register more than one fingerprint to unlock the device. This may make the device more vulnerable by allowing a greater amount of features to unlock the system. (Serra)
- We may also want to consider an authentication system that first determines whether a provided fingerprint is genuine or artificially generated before checking if there is a match in the system. In this type of system, the deceiver would have a harder time generating an effective masterprint as more features would be examined by the recognizer before any comparisons are made in the database. (Patrick)

7.1.2 Adversarial Neural Network

- Generative adversarial networks (GAN) could potentially be used to hone in our adversarial fingerprints. Having our fingerprint authentication system as the discriminator and us coming up with a generator could create a

process flow to rapidly generate images until they are recognized. Potentially, we could use the GPUs in the Senior Design Lab to speed up this process. (Wyatt)

- If the GANs we develop are not convergent, that is to say, they don't all end up producing identical products, then we can build several competing GANs to improve the recognizer further. (Mel)
- Training competing GANs can be done in parallel, each one against their own copy of the recognition software. We can do this once for each piece of hardware available to us. (Mel)
- A threshold for when our GANs are sufficiently trained should be determined in order to prevent harmful overtraining and wasted time. (Patrick)
- To make sure the machine is not simply memorizing the data, and it's actually finding the features in the fingerprint, we should save some of the fingerprint data until the very end of the training to test the accuracy of the model we created as a part of the evaluation process. (Serra)
- We can start with the adversarial method called Houdini to generate a print that belongs to "Bob" but the system recognizes as "John." From there, we can learn what common features or noise we can add to a print to make it fool the system into thinking it's multiple people's fingerprint. [15] (Mel)

7.1.3 Web Interface and UI

- To create the web application, we can use the AngularJS framework and Node.JS environment to create a visually appealing and flexible interface to interact with the machine learning system. Node.JS is lightweight and easy to set up, so it will be ideal for our situation. (Serra)
- For the sake of demonstration, we may want to implement features in the UI beyond uploading sets of fingerprints and downloading the sets of masterprints. Maybe the user could have an option to add or eliminate layers in the neural network to see how that affects the resulting masterprint production, or perhaps if we use multiple NN designs, the user could choose between them when uploading their fingerprint sets. (Patrick)

7.1.4 Database

- We can use the Java JDBC driver to connect to any database easily, and all we need to do in Java afterwards is build an adapter to construct SQL queries and commands for our other programs. (Mel)
- We have found a Python DB Driver that can connect to almost all common database. As a result, this portion could be written in Python. Notable exception to connectable databases is Oracle, but there is no need to use an expensive database. (Mel)
- An advantage of writing our Database adapter in Java is the ability to utilize MyBatis Mygrations to easily control database versioning. It allows us to build up a new database or roll back a database to any one specific version

that we had previously through a very easy scripting method. This way, we never need to worry about using the wrong version of our database, as it's managed for us. (Mel)
- Using PostgreSQL seems like a good option because it's widely used and stable. Since no one in the group has worked with large databases before, using something well-documented is a must to be able to finish on time. Working in the same language as our neural networks will also make it easier to link the two together without any quirks. (Serra)

7.1.5 Integration with Commercial Hardware

- We can attempt to break our own fingerprint authentication by registering our own fingerprints to the hardware that will be provided as a real life example of how real authentication can be attacked. A demonstration could be made on the fly, which will be more impressive when doing the presentation. (Serra)

7.1.6 3D Printing

- The fingerprints could be printed with a hard plastic and a soft, more skin-line material to test which material the fingerprint sensors prefer. Hard plastic will probably keep the details of the fingerprint more, but the ability to compress the softer material more might work in our favor for fooling the sensor into thinking it's a real finger. (Serra)

7.2 First Prototype

The first prototype features four sets of neural networks and a database. Each team member worked on one set of neural network pair. We had two main goals for the first prototype. The first goal is to set up the right type of environment to build and train on. The second goal is to familiarize ourselves with the process of designing, coding, and training neural networks. This way, we can better answer questions like how much time needs to be allotted to training and what is the best way to collaborate on the neural network.

The discriminative neural network will not try to match identities or authenticate a user. For the first prototype, we stick to setting up a discriminator that tries to tell if a fingerprint was real of fake. This means we won't have to worry about generating masterprints or creating an algorithm that allows and denies access. Instead, we can concentrate on just the first step, which is generating a believable fake fingerprint.

The neural networks is built in Anaconda using TensorFlow libraries. All the neural networks use the WGAN algorithm to generate the fingerprints. The first neural network trains with fingerprints captured from a capacitive sensor. It uses 3 layers of transformations to build the fingerprint on the generator. The second neural

network also trains with fingerprints captures from a capacitive sensor. It uses 6 layers to build the fingerprint. This way, we can compare whether more layers leads to better results. The third neural network trains with fingerprints captured with rolled ink; it uses 3 layers. The fourth neural network trains with rolled ink as well; it uses 6 layers. With this method, we can compare which database works the best in creating the most authentic fingerprints.

7.3 Second Prototype

The second prototype will feature a website, two databases, and a set of neural networks. For this prototype, the main goal is to get an idea how the different aspects of the project will fit together. The website will run with Node.JS to handle request and Vue.JS as a front-end framework. The two databases will run on PostgreSQL. The first one will be built with an unstructured, non-relational data organization using JSON. The second one will be built with structured, relational data organization using YAML. We will test which database type fits our data better and works faster.

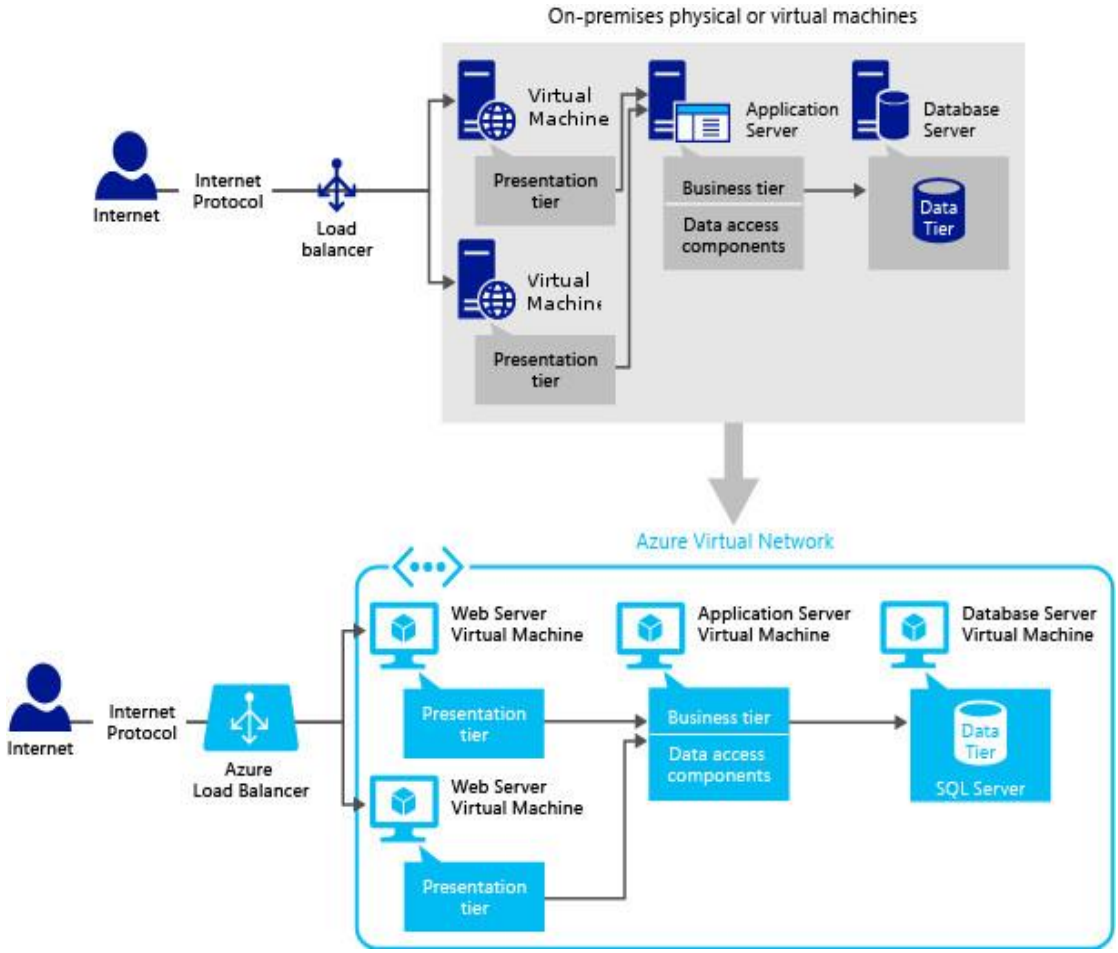


Figure 7.3.1 Local vs. Cloud Virtual Machine Network Diagram [45]

The neural networks will be built using WGAN as a template, using the loss function and number of layers that is decided through the first prototype. Training will be done with the NIST Special Database 9. The main goal of this part of the prototype is to test the best way to run multiple instances of the GAN concurrently with each request. We will set up multiple VMs locally that can run the GANs. We will also put the GAN on a cloud service. Each VM will be initialized with the Anaconda environment. We will test which of these methods runs the GAN the fastest and adjust our budget accordingly.

Since neural networks are usually not utilized due to the amount of time they take, it will be important for us to run the GAN in a reasonable time simultaneously. The cloud programs we will test out will be Amazon Web Services, Microsoft Azure Cloud, and Google Cloud. The figure above demonstrates the structure a Microsoft Azure Cloud service would be organized versus a local set-up.

The most critical part of this prototype will be to set up the API that will control the connection between the GAN and the website. It will be written in Python and hosted on the server. It will have two versions, one that makes calls to the non-relational PostgreSQL and another that makes calls to the relational PostgreSQL. It will handle storing and assigning IDs and users to each process. It will also manage formatting the data received from the user and checking and sanitizing input to fit the database fields, since special characters, spacing, and SQL injections are all things to consider when setting up API calls.

Securing the VMs and databases will also be done during this prototype together with setting them up. We will generate RSA tokens to sign in with. A lot of the recent hacking attempts have been due to insecure databases, VMs, and servers that were using default or easy-to-guess passwords.

7.4 Environment

The project is built using Anaconda in Windows. We chose Anaconda over Docker to set up the environment and share packages because Docker for Windows needs Windows 10 Pro (which is the only version that includes Hyper-V). Since this version is $199 and not a part of our budget, we opted to use Anaconda. The process for setting up the environment used for our first prototype is as follows:

Note: Do not install Cuda before Microsoft Visual Studio 2015. Do not install Visual Studio 2017. Cuda will not install correctly without Visual Studio, and it does not compile on Visual Studio 2017 since the latest update.

7.4.1 Installing Anaconda

1. Download the Anaconda 5.1 for Windows Python 3.6 Version Installer from the link https://www.anaconda.com/download/

2. Proceed through the installer until the Advanced Options dialog box. Check “Add Anaconda to my PATH environment variable.” Then, click Install.
3. When the installer finishes, run Anaconda Prompt from your Start Menu and type conda update -n base conda into the prompt. Enter “y” when prompted with “Proceed ([y]/n)?”.
4. Enter conda install python-dateutil. Enter “y” when prompted with “Proceed ([y]/n)?”.
5. Enter conda install python=3.5. Enter “y” when prompted with “Proceed ([y]/n)?”. Note: This step may take a while. I suggest moving onto the next step while you wait.
6. Check if your GPU is CUDA-compatible at https://developer.nvidia.com/cuda-gpus . If it is CUDA-compatible, proceed to “Installing GPU Support” below. Else, proceed straight to “Installing Tensorflow”

7.4.2 Installing GPU Support

1. Navigate to https://www.visualstudio.com/vs/older-downloads/ and click the download button next to “Visual Studio 2015 and other Products.” *Note: If you do not have a Dev account, you will need to make one first.*
2. Find and download Visual Studio Community 2015 from the list.
3. Proceed through the Visual Studio Community 2015 installer and finish the installation.
4. Nativate to http://www.nvidia.com/Download/index.aspx?lang=en-us.
5. From the dropdowns, specify your GPU type, series, product name, operating system, and language, and click Search. If you don’t know the name of your GPU, you can find it by going to Start Menu > Run, typing **dxdiag**, and clicking on the “Display” tab.

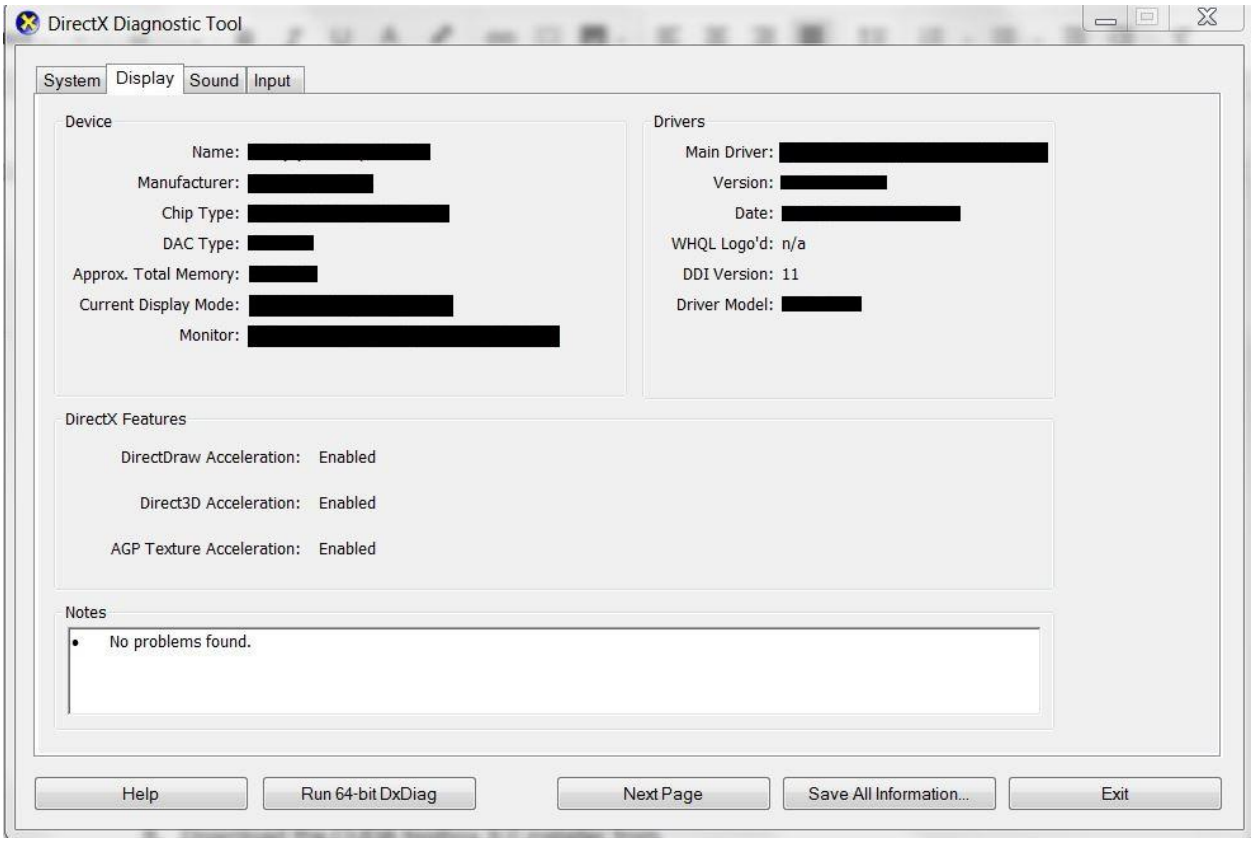


Figure 7.3.2.1 dxdiag Display tab

6. Click the Download button.
7. Run the installer.
8. Download the CUDA toolbox 9.0 installer from https://developer.nvidia.com/cuda-90-download-archive by selecting Windows, x86_64, version 10, and exe (local).
9. Install CUDA using the installer prompt.
10. In a file explorer window, navigate to the nbody directory of your CUDA installation. For our team, this was located in *C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.0\5_Simulations\nbody*
11. Open the file nbody_vs2015.sln in Visual Studio Community 2015
12. In the "Build" menu in Visual Studio, select "Build Solution."
13. In a file explorer window, navigate to the build directory. For our team, this was located in *C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.0\bin\win64\Debug*
14. Run nbody.exe
15. Navigate to https://developer.nvidia.com/cudnn and click the "DOWNLOAD cuDNN" button.
16. Make an NVIDIA Developer Program account if you don't already have one.
17. Agree to the Terms and Conditions, and click on "Download cuDNN v7.0.5 (Dec 5, 2017), for CUDA 9.0." Select cuDNN v7.0.5 for Windows 7 or cuDNN v.7.0.5 for Windows 10 depending on your version of Windows.

7.4.3 Installing Tensorflow

1. Open Anaconda Prompt from the Start Menu.
2. Type conda create -n tensorflow pip python into the prompt and enter.
3. Type activate tensorflow into the prompt and enter.
4. If you installed GPU support, type pip install --ignore-installed --upgrade tensorflow-gpu into the prompt and enter. Else, type pip install --ignore-installed --upgrade tensorflow into the prompt and enter.
5. Validate your installation by entering python, and then entering:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

If the console prints “Hello, TensorFlow!”, you’ve correctly installed the environment.

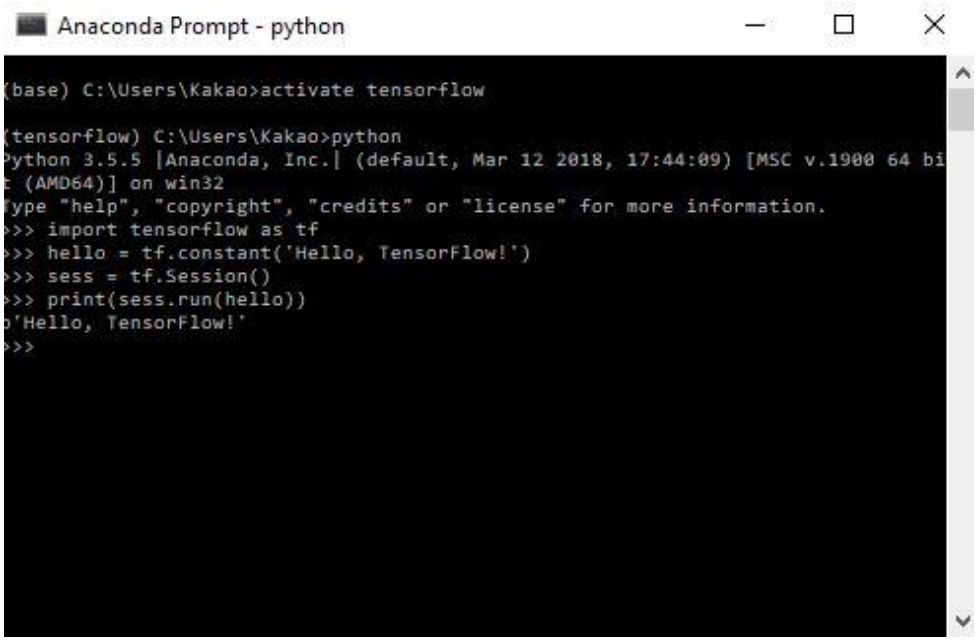


Figure 7.3.3.1 Example Anaconda Prompt

Note: Python will correctly continue to receive input until the last line, so if it’s throwing an error when you try to enter a line, the installation is incorrectly completed.

7.5 Security

Due to the nature of this project, sensitive material will be submitted to our databases by users with legitimate privacy concerns. Usernames, passwords,

emails, and most importantly, fingerprints will be stored in databases and transmitted through our services.

7.5.1 Sanitizing Input

SQL injection attacks are one of the main attack methods that hackers use to leak data from vulnerable websites. These injections work by entering a string that is rendered as code when it is stored in the server to run, essentially changing the code to display or modify the database. For example, the SQL statement SELECT * FROM product WHERE producttype = '__data_to_be_entered_here__'; , where the ___data_to_be_entered_here__ is expected to be a harmless string like shirts, could be exploited with an input of ';SELECT * FROM customer WHERE '' = '. This input would form the query SELECT * FROM product WHERE producttype = '';SELECT * FROM customer WHERE '' = '';, which changes the code completely and returns the list of all customers instead [46]. The only way to combat this is to sanitize the input. Sanitizing the input means that the website takes the data that the user enters and restructures it into a harmless format. The easiest way to do this is to use the "execute" method in psycopg2, the Python version of PostgreSQL, which sanitizes inputs automatically. For example, the line cur.execute("SELECT * FROM platforms WHERE language = '%s';" % platform) would automatically escape special characters by adding backslashes in front.

7.5.2 Server

Our data will be stored in the server provided to us, which is already protected with an SSH key. We will be under limited accounts, which will prevent any hackers from pivoting inside the system. Each file will be owned by a user with limited permissions, typically with no write permissions. We will update our server components at the end of the project to make sure we are presenting a secure software at launch.

7.5.3 Authentication

The login authentication will be handled by OAuth 2.0, which is the industry-standard protocol for authorization. The framework allows for secure login, which means that we will not have to implement our own login system. The different grant types that it provides include:

- Authorization Code: uses secure code to exchange access tokens through a redirect URL
- Implicit: uses a simplified flow that returns an access token without an extra authorization code exchange
- Password: uses a password to exchange the access token
- Client Credentials: relies on the client already having an access token to pull data about the client

- Device Code: uses a device identifier to exchange the access token for browserless set-ups
- Refresh Token: assigns a refresh token with access tokens for seamless web usage

We will use password, client credentials, and refresh token grant types in our website to ensure security. This way, the user will constantly be authenticated on every login, every refresh, and every request.

Here is how the service ensures security with tokens:

- User enters password into the service to receive the access token by exchanging the token with their password.
- The user clicks Enter to send the Request packet and the service checks to see if the token belongs to the password hash.
- The user clicks to allow the authorization.
- The server sends back a validated packet that grants the request, and the client exchange tokens with the server.
- Then the user has access, which is now stored in the session storage.

Storing the OAuth access token in the cookies can be a security risk if not done correctly. This is why the service stored the token in the session storage. However, Oauth falls back on cookies when this is not available.

8. Testing and Evaluation

8.1 Preliminary Ideas

8.1.1 Test Cases

- We can augment our unit tests with EvoSuite in any code that is written in Java. EvoSuite can often find edge cases and exceptions that we would not think of handling. (Mel)
- The AFL (American Fuzzy Lop) Fuzzer or other open-source fuzzers could be utilized to fuzz our system during the stress test. The fuzzer will send different inputs into the system with the main goal being to crash the system. This usually signifies a vulnerability or an input that a human could make when interacting with the system that leads to an ungraceful exit, which is not desirable. (Serra)

8.2 Test Cases

15 test cases will automate simulating general use, as well as testing for edge cases. These scripts will run the expected input in an expected load. 3 test cases will automate stress testing for the website and database. These scripts will test for unexpected loads and ensure that the website doesn't become unavailable when used simultaneously by multiple user. The details of each test case scripts are as follows:

General Use Test Cases

1. Runs the GAN with various rolled ink fingerprints

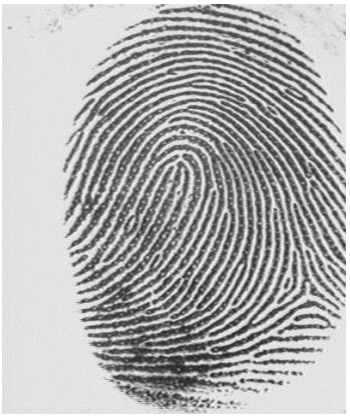


Figure 8.2.1 Example of Rolled Ink Fingerprint [47]

2. Runs the GAN with various capacitive fingerprints

Figure 8.2.2 Example of Capacitive Fingerprint [48]

3. Runs the GAN with various optical fingerprints

Figure 8.2.3 Example of Optical Fingerprint [49]

4. Runs the GAN with various unrelated images
5. Runs the GAN with various incorrect file types
    a. .exe
    b. .pdf
    c. .gif
    d. .zip
    e. No file extension
6. Runs the GAN with no input
7. Runs the GAN with multiple rolled ink fingerprints
8. Runs the GAN with multiple capacitive fingerprints
9. Runs the GAN with multiple optical fingerprints
10. Runs the GAN with a combination of fingerprints and unrelated images
11. Runs the GAN with a combination of fingerprints and incorrect file types
12. Creates user accounts on web interface with various username and password formats and SQL injection formats
    a. Special characters
    b. Too long strings
    c. " or ""="
    d. User OR 1=1

    e. 105; DROP TABLE Users
13. Uploads, then requests deletion of fingerprints in database through web interface
14. Logs in with correct user accounts on web interface, then logs out
15. Attempts false login attempts

Stress Test Cases

1. Sends fingerprint images with large sizes
2. Simulates DoS by sending continuous requests to server
3. Runs multiple general use test cases simultaneously. Test cases 1-3 from the General Use Test Cases will be executed simultaneously from different connections to simulate multiple users using the service at once.

9. Related Works

Many research has been done that is similar to our project. However, no research group has released code or detailed findings on their research, and all research has been done in the last few years.

9.1 DeepFool

DeepFool is an algorithm proposed by the authors Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard from École Polytechnique Fédérale de Lausanne. The algorithm addresses the growing needs to improve the robustness of deep neural network image classifiers by efficiently computing "perturbations that fool deep networks" [50]. The research paper proposes an algorithm that can compare how robust different classifiers are, performs extensive experimental comparison that shows "augmenting training data with adversarial examples significantly increases the robustness to adversarial perturbations," and show "imprecise approaches...could lead to different and sometimes misleading conclusions about robustness" [50]. This paper is essentially following the same experimental process we are following, except their goal is reverse ours. While our goal is to create a robust adversarial system to fool classifiers, the goal of the research team is to create a robust classifier.

9.1.1 Algorithm

The algorithm views the classifier as an "aggregation of binary classifiers" [50]. The formula is as follows:

Algorithm 1 DeepFool for binary classifiers

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.
Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\text{sign}(f(\boldsymbol{x}_i)) = \text{sign}(f(\boldsymbol{x}_0))$ **do**
    $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{\|\nabla f(\boldsymbol{x}_i)\|_2^2} \nabla f(\boldsymbol{x}_i)$,
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$,
    $i \leftarrow i + 1$.
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$.

Figure 9.1.1.1 Closed-form algorithm of minimal perturbation [50]

This formula descibes the "minimal perturbation to change the classifier's decision, with *x* representing the image generated by the adversarial neural network and

inputted into the classifier. In simpler terms, the formula minimizes the changes to an image required to change the classifier's answer to a question solved by machine learning (hence the loop until the sign change). The algorithmic result that is stored in r_i is the perturbation that is applied to the pixels of the image.

9.1.2 Experimental Results

To test the algorithm in practical cases, the research team attacks various deep convolutional neural network architectures with MNIST, CIFAR-10, and ILSVRC 2012 datasets [50]. MNIST is used to train a “two-layer fully connected network, and a two-layer LeNet convolutional neural network architecture”, CIFAR-10 is used to train a “three-layer LeNet architecture, as well as a Network In Network (NIN) architecture”, and ILSVRC 2012 is used to train “CaffeNet...and GoogLeNet” pre-trained models [50]. Using the perturbations that result from DeepFool, the team is able to judge which classifiers took the longest time to be fooled by the perturbations in the image.

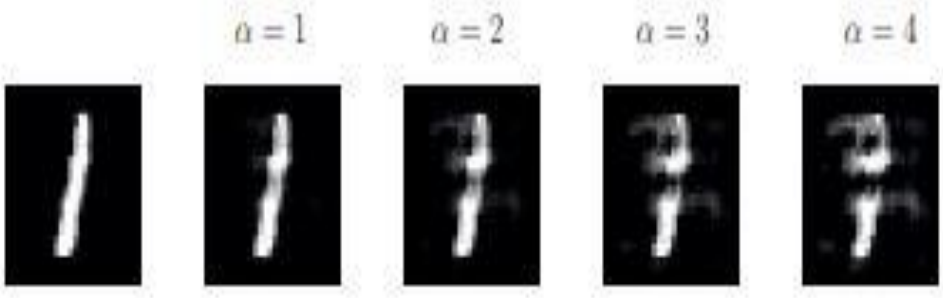


Figure 9.1.2.1 DeepFool Perturbation example [50]

In figure 9.1.2.1, the algorithm is shown in a real-life example. The original image is classified by the neural networks as a “1”, and the DeepFool perturbed images are classified as “7” after several iterations [50].

9.2 Minutiae Extraction

In their research paper [50] “Neural Network-based Minutiae Extraction for Fingerprint Verification Systems,” researchers Basem O. Alijla, Motaz Saad, and Sally F. Issawi tackle minutiae extraction, which is a large and critical part of our project that this research directly relates to. The approach the researchers take to extract minutia is called “Feed-Forward Neural Network (FNNN)...The method extracts the bifurcation minutia features and core points” [50].

The researchers split the phases of minutia-based fingerprint recognition method into 4 phases: acquiring high quality images, preprocessing images, extraction of the minutiae, and pattern matching based on distance. To get the best results, high quality images without excess or insufficient ink is necessary; the FVC2002 optical sensor database is used for this reason.

Image preprocessing entails noise removal, binarization, thinning, and adaptive filtering. To remove noise, a median filter is used, and this method reduces multiple types of noises, such as speckle, salt, and pepper noises. Median filters correct images “that contain falsely black lines with white points, which look like a sharp edge, which may classify incorrectly as a bifurcation” [50]. In specific, the medfilt2 function is performed. Binarization is achieved by converting the gray level fingerprint imagine to 0-bit and 1-bit representations for ridges and furrows of the fingerprint. This is done with an 8-bit gray and a im2bw filter with a 0.6 threshold. Thinning removes redundant pixels to remove unwanted edge points, which is done through the thin function in MATLAB.

The features that the researchers aim to extract from the fingerprint are represented in Figure 9.2.1. The paper limits its focus to extracting ridge endings and ridge bifurcations as they are the most informative features in recognition systems.

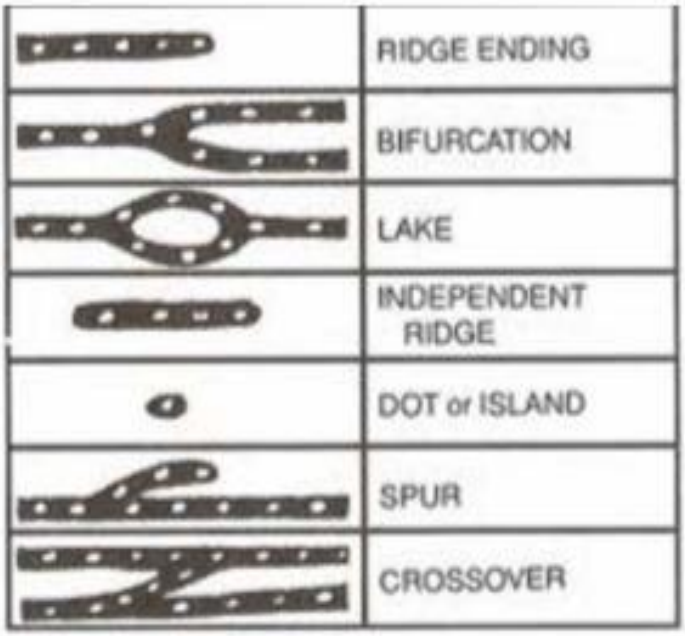


Figure 9.2.1 Ridge characteristics [51]

Core point detection is the third method used to extract minutiae from the image. The fingerprint core point is “the topmost point on the innermost upwardly curving ridge line... Core point is one of the most informative features in fingerprint recognition” [51]. The Poincare method used to facilitate the detection of this minutia. To extract the minutiae, every possible 3x3 block of the image is fed into the Feed-Forward Neural Network to classify the blocks. Then, the neural network is ran with various parameters to find the best setting that extracted all possible bifurcations, which is displayed in Figure 9.2.2.

Table 1: FFNN setting

Input layer	Nine neurons
Output Layer	One neuron, 0 non bifurcations, 1 bifurcations
Hidden layer	One layer of 5 neurons
Number of epochs	5000
Learning function	LEARNGDM
Learning rate and momentum	0.3, and 0.9 respectively

Figure 9.2.2 FFNN best parameters [51]

The output matrix is generated, with code “1” representing bifurcations and “0” representing non-bifurcations.

In our project, we aim to match fingerprints also through a neural network. However, the researchers of this extraction method instead utilized the Euclidean distance, which is the distance between any two points, to match two fingerprints to each other. The percentage of correct matches for this method is an average of 91.6%, with image quality and finger surface condition (i.e., wounds or injuries)” negatively influencing the extraction process and bringing down the average matching ratio. Especially in smartphones, this is something that could be taken advantage of because smartphones have to prioritize ease-of-use more than other devices. This could translate to worse image quality and finger surface condition, leading to a less secure matching system to accommodate this.

9.3 DeepMasterPrint

This project and research paper is the most closely related work to our own project. In “DeepMasterPrint: Fingerprint Spoofing via Latent Variable Evolution,” authors Philip Bontrager. Aditi Roy, Julian Togelius, and Nasir Memon present a method of attacking fingerprint recognition systems using “*MasterPrints,* synthetic fingerprints that are capable of spoofing multiple people’s fingerprints” [52].

With the Latent Variable Method, the researchers were able to match 23% of all users in a strict security setting and 77% of all users in a looser security setting. The strict security setting allows a 0.1% false match rate, and the loose security setting allows for a 1% false match rate.

The problem presented in the paper is that Generative Adversarial Networks were not controllable by design and would not be able to support both verifying and matching fingerprints to identities. They use the “Covariance Matrix Adaptation

Evolution Strategy" to search the input to the fingerprint authentication neural network for fingerprints that the neural network would identify as multiple people. Their approach to GANs was as follows:

1. Provide real images to the discriminator. Train the discriminator to classify them as real.
2. Provide generated images to the discriminator. Train it to classify them as generated.
3. Provide the generator with the discriminator's gradients. Train the generator to produce images that are classified as real [50]

They utilized the Wasserstein GAN (WGAN) and gradient penalties to keep the networks trained in a balanced manner so that one does not surpass the other. To match fingerprints, they adopted the evolutionary optimization method, which samples from a population and evaluates relative fitness to operate. In this project, the fitness function is the acceptability of the fingerprint.

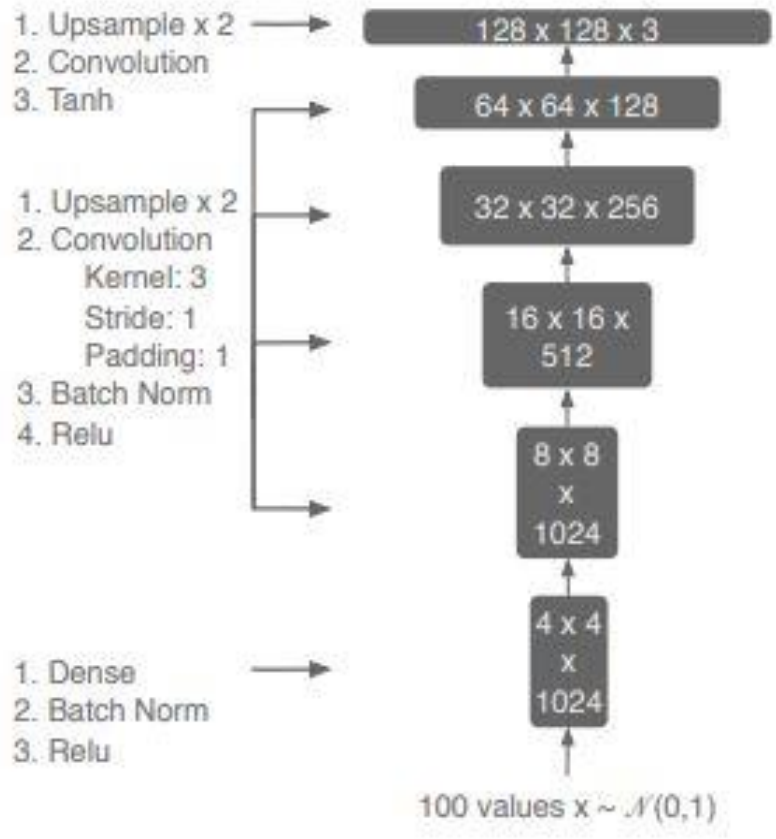


Figure 9.3.1 Generator Network Architecture [52]

The figure above illustrates the methods used to generate the fingerprint data for the generative model. The reverse of these methods would be the model for the discriminative neural network. The research team creates two generative neural networks. The first network is trained with a set of fingerprints captured by a

capacitive sensor; the second is trained with fingerprints in rolled ink. The generators are trained for 120,000 updates, and the discriminator is trained 5 times between each update.

Once the generators have been trained to create fingerprints, the fingerprint must evolve the latent variables to optimal values. This will allow the generators to generate MasterPrints that pass as multiple fingerprints, rather than generate any fingerprint. The algorithm is shown in the figure below.

Algorithm 1 Latent Variable Evolution
$fmr \leftarrow 1\%, .1\%, .01\%$ and $fingerprint \leftarrow 12\ partial$

```
1: G_θ ← trainGAN(data)
2: function MatchingScore(X)
3:     img ← G_θ(X)
4:     score ← 0
5:     for fingerprint in data do
6:         for partial in fingerprint do
7:             if matching(img, partial, fmr) then
8:                 score++
9:                 break
10:            end if
11:        end for
12:    end for
13:    return score
14: end function
15: MasterPrint ← CMAES(MatchingScore
```

Figure 9.3.2 MasterPrint Generative Algorithm [52]

The idea of the algorithm is to take an image with a hundred latent variables and finds an optimal solution--a point in the latent variable matrix. This algorithm is called the Latent Variable Evolution by the authors because it allows the fingerprints to evolve with their latent variables until the best group of latent variables arise. The fitness score associated with evolutionary algorithms is the number of identity matches of the fingerprint. It takes around 3 days for each fingerprint to evolve.

To test the fingerprints, VeriFinger fingerprint recognition technology, algorithm, and SDK system was used. VeriFinger is an industry-level software that is being used currently to identify fingerprints on stand-alone or web-based products. Over 1500 brands currently use it in their products. This system emulates the capacitive sensors on smartphone, which is the primary focus of attack for the research. The first neural network with a database of capacitive sensor fingerprints demonstrates

an attacker attacking with the same type of data as the sensor, whereas the second neural network with rolled ink fingerprints demonstrates robustness and ease of attack [52].

10. Facilities and Equipment

The training for the CNNs will be done using the computer lab provided by Dr. Sumit Kumar Jha in HEC 255 in the University of Central Florida. The lab contains supercomputers running with NVIDIA Tesla V100 GPUs [53]. These GPUs are specifically aimed toward developers working with Artificial Intelligence. The Tesla V100 GPUs boast 640 Tensor Cores, which are GPU cores specifically designed for deep learning to optimize matrix computation [53].

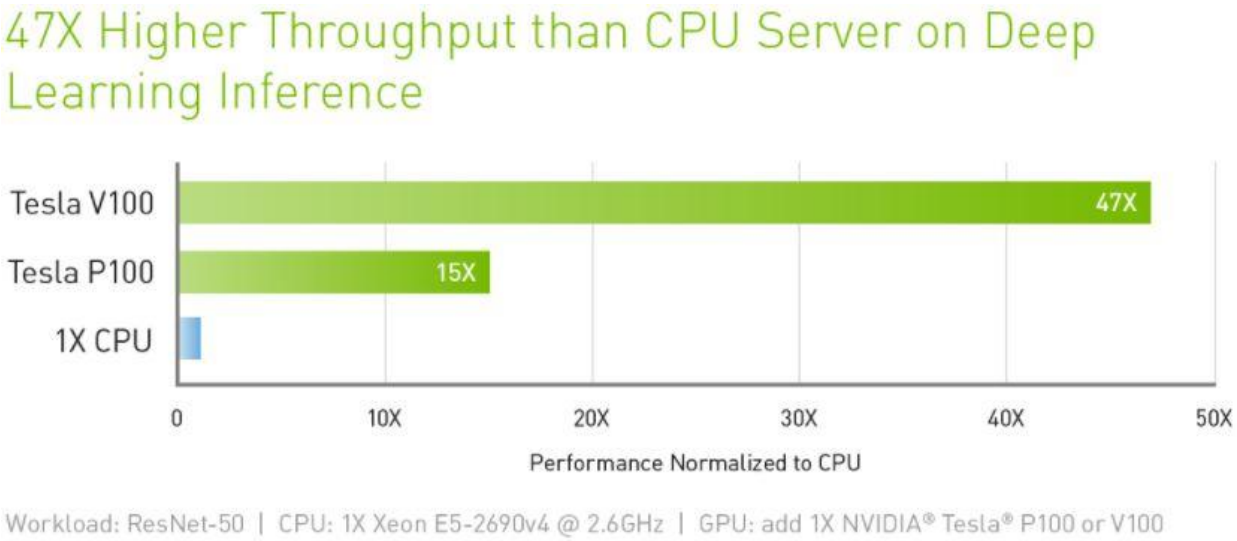


Figure 10.1 NVIDIA Tesla V100 vs CPU Throughput Comparison [53]

As of April 2018, this GPU is the highest performing GPU for deep learning, beating the Tesla P100's training time of 15.5 hours by 10.4 hours, as well as major increases in performance compared to CPU-only training, as shown in the figure above. Since our project needs to train in a moderately fast pace due to our time constraints, we will need the best GPU we can get our hands on, and the throughput of this GPU is promising.

The development of the neural networks will be done on personal computers outside the lab. Coding will be done on both personal laptops and desktops, while the training will be done on the lab desktops. The first and second prototype training will be done on the personal desktops. Computer specifications for each team member is as follows:

Serra Abak

Desktop

- CPU: Intel Core i5-3570, 3.40GHz
- GPU: NVIDIA Geforce GTX 660 TI
- RAM: G.SKILL Ripjaws X Series 4x4GB DDR3 2133 MHz (16GB total)

- OS: Window 10

Laptop
- CPU: Intel Core i5-6300 2.40GHz
- GPU: Intel HD Graphics 520
- RAM: 8 GB
- OS: Windows 7

Mel Pelchat

Desktop
- CPU: Intel Core i7, 7th Gen
- GPU: NVIDIA Geforce GTX 960
- RAM: 32 GB DDR3 2800
- OS: Windows 7

Laptop
- CPU: Intel Core i7 8550U, 8th Gen
- GPU: Intel UHD Graphics 620
- RAM: 16 GB
- OS: Windows 10

Patrick Maier

Desktop
- CPU: Rx 580
- GPU: AMD Ryzen 5 1400
- RAM: 8 GB
- OS: Windows 10

Laptop
- CPU: Intel Core i7 5500U
- CPU: Intel HD Graphics 5500
- RAM: 6 GB
- OS: Windows 10

Wyatt Waterfield

Desktop
- CPU: Intel Core i7-3960X 3.30GHZ
- GPU: NVIDIA Geforce GTX 555
- RAM: 2x8GB DDR3 2133 MHz (16GB Total)
- OS: Windows 10

Laptop

- CPU: Intel Core i5-4300U 1.9GHz
- GPU: Intel HD Graphics Family
- RAM: 4 GB
- OS: Windows 10

11. Budget and Financing

11.1 Fingerprint sensor

The cost of fingerprint scanning hardware may factor into the project if we decide to test our system on an existing commercial sensor.

Figure 11.1.1 Commercial fingerprint scanner on Amazon website

Estimated Cost: $20 - $140

The exact cost depends on quality and brand of sensor we wish to test. This range was estimated based on publicly available fingerprint scanners sold online.

11.2 Server / AWS

If server/database resources are not provided by our sponsor, this will need to be factored in to the budget. Amazon would be the likely choice for these needs, through AWS.

Figure 11.2.1 Amazon Web Services logo

Estimated Cost: Database (Amazon RDS) – $0.018 - $5.844 (Total/hr)
Deep Learning AMI - $0.023 - $5.520 (Total/hr)

Large price range for these services, as it is difficult at this time to determine how much data we will be working with.

11.3 3D Printing

Due to creation of a 3D printed master fingerprint being one of our stretch goals, we may need to consider the cost of materials for the 3D printing process.

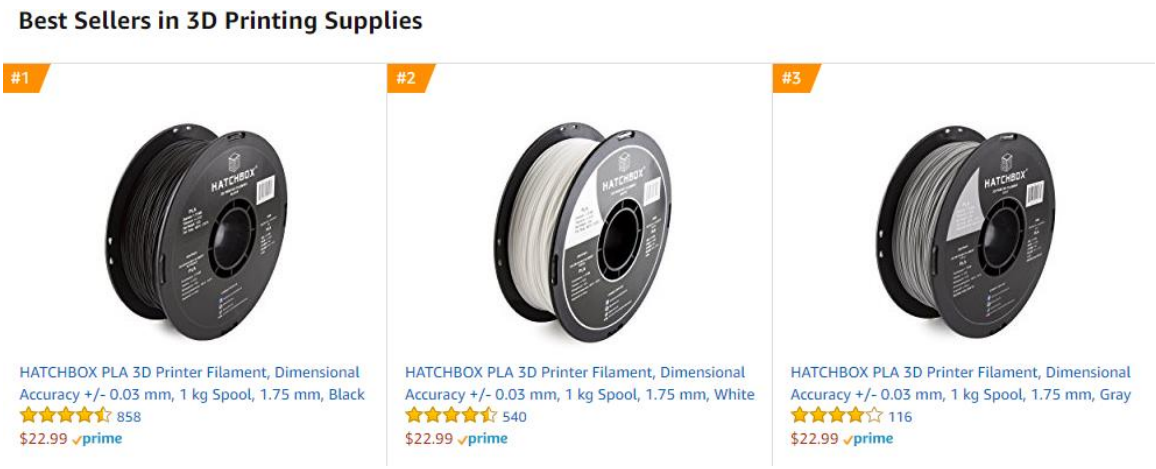


Figure 11.3.1 3D printing material on Amazon Store

Estimated Cost: Flexible Filament ~ $35/Kg on average
Specialty Flexible Filament ~ $80-$105/Kg
Generic PLA/ABS ~ $25/Kg on average

The cost to 3D print depends heavily on the type of material used. Flexible filaments could be used for a more life-like model of a fingerprint but cost more on average than typical 3D printing material.

11.4 GPU

The cost of GPUs would not be factored in if we paid for Amazon's Deep Learning AMI service. If we forgo that service, we may need to purchase GPUs to accelerate the deep learning process if this hardware is not provided by our sponsor.

Figure 11.4.1 GPU for sale on Newegg.com

Estimated Cost: GTX 1060 6GB ~ $450 - $600
GTX 1080 Ti ~ $1300 - $2000

The type of GPU is dependent on the size of the neural networks that we will implement. This is difficult to determine at this stage of the project. If we wish to parallelize the deep learning process, multiple GPUs will be needed.

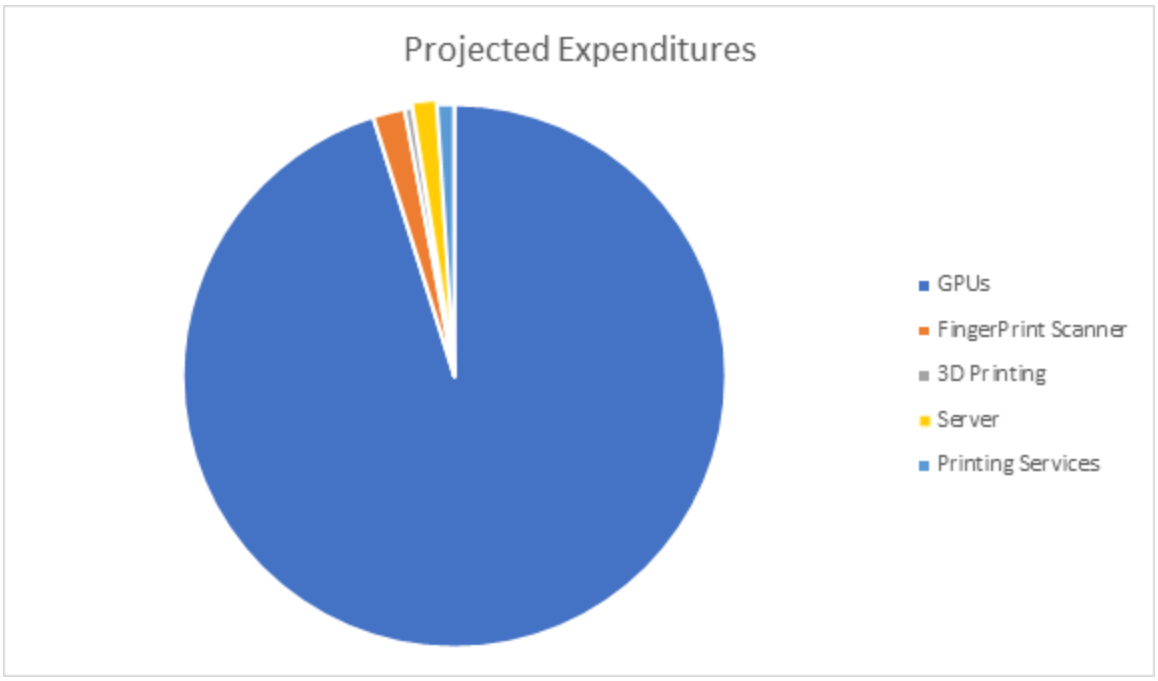
11.5 Printing Services

In order to complete the documentation portion of our senior design project, we obviously need to print out our document and have it professionally bound with a non-paper cover page. We basically have to viable options available for these surfaces. The Spot on UCF campus or The Office Depot. We decided on The Spot for the sake of convenience.

Figure 11.5.1 The Spot logo

Estimated Cost: The Spot - $unknown
Office Depot ~ $46.84

Although the Spot was chosen for the sake of convenience, they do not have transparent pricing available online for their services. Unlike Office Depot, in order to determine an estimated cost from the Spot, we would need to contact an employee for additional information in order to receive a quote.


Projected Expenditures
GPUs
FingerPrint Scanner
3D Printing
Server
Printing Services

12. Milestones

The milestones of the projects are as follows:

02/15/2018	Initial project description and requirements
03/01/2018	Initial research, budgeting, milestones
03/15/2018	High-level architecture design and technical stack confirmation
03/25/2018	Environment set-up, first prototype of initial fingerprint authentication neural network
04/25/2018	Final design document, first prototype of adversarial neural network
07/01/2018	Web interface and UI, database, neural network training
08/25/2018	Final fingerprint authentication neural network
10/01/2018	Final adversarial neural network
11/01/2018	Test cases and stress testing
12/03/2018	Final, finished product

These milestones indicate checkpoints without stretch goals. If the last base requirements checkpoint was reached three (3) months in advance, then the milestones of the stretch goals, starting from mid-September, would be as follows:

8/20/2018	Modified prototype of adversarial neural network to work with fingerprinting authentication hardware
10/01/2018	3D print of master fingerprint key created
10/30/2018	Final adversarial neural network that successfully attacks hardware
12/03/2018	Test cases and stress testing, final product due

If the last base requirements checkpoint was reached (2) months in advance, then the milestones of the stretch goal, starting mid-October, would be as follows:

10/15/2018	3D print of master fingerprint key created

11/15/2018	3D print of master fingerprint key successfully attacks our own authentication neural network
12/03/2018	Test cases and stress testing, final product due

13. Project Summary

The requirements of this project were broken up into three main goals:

1. Develop a Discriminator Neural Network that can successfully extract minutiae from fingerprint images and authenticate the minutiae information against a database of accepted prints.
2. Develop a Generative Adversarial Neural Network that can train using limited information from the Discriminator Network and generate fingerprints that can deceive the authentication system with a certain rate of success. This success rate should be high enough that the generated fingerprints can be considered to be “master” fingerprints.
3. Create a simple, intuitive web interface that allows the user to upload their own fingerprint information to be processed and used as training material for the Discriminator and the GAN. The output is a master fingerprint that corresponds to the data that the user uploaded.

The following are descriptions of how we have accomplished these goals and what difficulties we faced as well as how we overcame them. The numbering corresponds to the objective as stated above. Before that though, the division of labor between members, and their individual successes and failures will be listed.

Review of Final Labor:

Wyatt Waterfield:

- Original Delegated Responsibilities:
  - ➢ Website backend (Management of backend resources including containerizing tensor flow models and control the serving between training models)
  - ➢ Development of Generative Adversarial Network (each group member developing one)
  - ➢ Training of Neural Networks under his own jurisdiction.
- Successes in Each Responsibility:
  - ➢ Wyatt effectively automated the backend processes of the final web application. He ensured that the virtual machine running the containers had all appropriate dependencies and that all models had access to the correct data and resources for the training process.

The process of swapping between training models was done seamlessly, with the discriminator network and deceiver network exchanging information limited based on the type of attack we were attempting (white box, black box, or grey box).

- ➢ Wyatt's final prototype for his GAN had nine layers in the discriminator and deceiver neural networks. This was on par with the complexity of the Generative Adversarial Networks developed by the other members of the group. This discriminator paired with his personally created network was the same one used by the rest of the team. The discriminator was based off of what we had learned from FingerNet, but with a much simplified neural network. Their network had around sixty layers, while ours had four to eight from first prototype to last. Wyatt's created network was successful in creating a master fingerprint.

- ➢ All members of the group used the same training data originally, so Wyatt's training was not dissimilar to that of the other group members. The training only differed once the web interface was complete and users could upload their own data sets.

- Failures/Learning Experiences:

  - ➢ There was some confusion in the early stages of orchestrating the back end. Some group members believed we would need to have individual Virtual Machines that were activated by a puppet master. The additional resources that these Virtual Machines needed only depended on what was different in their template from the original. This idea was changed though when we realized that we would only really need one virtual machine for all of the training models. The dependencies were all identical for each training model, so individual virtual machines did not need to be generated.

  - ➢ Wyatt faced the same issues with his implementation of the GAN as everyone else in the group. His early networks started off in a simple state and could not develop the complexity to really learn how to fool the discriminator. Through future iterations he adjusted layer types and even added more layers until the fake prints more accurately matched those in the database.

  - ➢ The main difficulty with the training data was having complete information to train with. The information for the deceiver network came from the discriminator network and the information stored in the database. In order to get metadata about the fingerprints though, we had to do more labor to generate orientation and minutiae-labeled images.

Serra Abak:

- Original Delegated Responsibilities:
    - ➢ Website frontend logic and some user-interface design.
    - ➢ Developing her own implementation of a Generative Adversarial Network.
    - ➢ Website and database security.
- Successes in Each Responsibility:
    - ➢ We ultimately decided on Vue.js for the front, per Serra's recommendations. The logic of the frontend was not really an issue with this project. Serra implemented logic to communicate with the backend and only allow for valid account registration and logins. Besides that we only had to upload the fingerprint information and update the progress of the learning asynchronously and display that information through graphs and charts. Serra found convenient tools for implementing these features and the front and retrieving the information form the backend was not too complicated.
    - ➢ Serra's GAN also successfully generated a master fingerprint that the discriminator accepted with a high level of confidence. Her Generative Adversarial Network utilized unique combinations and ordering of layer types, but she had similar numbers of layers in her prototypes.
    - ➢ Serra being the expert of the group in cyber security best practices was the person in charge of the website and database security. She handled the scrubbing and sanitization of inputs and database queries. She also ensured that the information in the database was effectively encrypted and that passwords were hashed after being salted for storage.
- Failures/Learning Experiences:
    - ➢ The main learning experiences for the frontend decisions happened in the original planning of the whole project design. Once Serra decided on what framework we were working with and what features we were implementing on the rest was not much of an issue to implement.

- ➢ Serra faced the same struggles as everyone else. The original neural network did not generate anything close to a recognizable fingerprint, and just as with the other implementations, adding and manipulating layers within the neural network and tweaking the loss function allowed for a final working GAN to be developed.

- ➢ Serra had no issues in implementing the security for the system as a whole. She had the necessary experience and know-how.

Mel Pelchat:

- Original Delegated Responsibilities:

  - ➢ Automation of the labeling of minutiae (necessary for training and implemented using computer vision techniques that Mel had learned before this project).

  - ➢ Develop his own implementation of a GAN and specifically have it work under black box conditions (no information from the discriminator network, the defenses it had in place, or what fingerprint information was in the database – minutiae or whole fingerprint images).

  - ➢ Investigate possible ways to defend against all forms of GAN attacks.

- Successes in Each Responsibility:

  - ➢ Mel's method for annotating the fingerprint minutiae, which was a necessary step of preprocessing for the training data to be ready for the neural networks, was successfully developed with his prior work in computer vision programs, especially those that were used for corner detection.

  - ➢ Mel's GAN was successful, with similar generation abilities of master fingerprints as all other members the group under normal training conditions. But Mel also was tasked with performing the black box attack, which was wildly unsuccessful due to time constraints and limited research.

  - ➢ Mel did find ways to defend against grey box attacks with some effectiveness, which in this field is always significant as defenses are limited and largely ineffective. He found no way to defend successfully against the white box style attacks, although this section of responsibility bled into the stretch goals and research and

implementation were contingent on the satisfactory completion of the necessary elements of the project.

- Failures/Learning Experiences:

  - ➢ Mel's challenges with the annotation processes derived from the need to adapt old code from previous projects that was not commented or organized very well. Eventually we had access to labelled fingerprint information from NIST databases, but not before Mel ultimately completed the computer vision techniques. Also, the minutiae from the database were more accurately labeled, as the labeling was done by hand from fingerprint experts.

  - ➢ Mel's main difficulties with the GAN implementation were obviously the limited nature of the attack conditions he was working with. Black box attacks are extremely difficult, and unique modifications were necessary to create a more "probing" process for the training of the GAN.

  - ➢ White box attacks are nearly impossible to defend against which was the source of Mel's difficulties in this area.

Patrick Maier:

- Original Delegated Responsibilities:

  - ➢ Created and managed the database (PostgreSQL with python driver).

  - ➢ Also created his own implementation of a GAN, just as those before him.

  - ➢ Developed the authentication system for the final form of the discriminator network and helped with the implementation of the discriminator neural network.

- Successes in Each Responsibility:

  - ➢ Successfully setup a PostgreSQL database on the server provided by our sponsor, Dr. Jha. The database stored the meta data for the fingerprint images, which significantly sped up the serving of fingerprint information and reduced the overhead need to manage the data in the database.

- ➢ As previously stated, Patrick also developed an implementation of the GAN. Having better knowledge of the discriminator network and authentication system, Patrick had the best neural network.

- ➢ The final authentication system was based on a paper referenced in FingerNet. It utilized two-dimensional tensor minutiae matrices and a fusion of tensor matrices to compare relative locations of minutia pairs and pairs of minutia pairs.

- Failures/Learning Experiences:

  - ➢ Struggled to find the most efficient way to store and organize the image information.

  - ➢ Same failure and success story as the other members of the group.

  - ➢ Patrick did not know much about tensors before going into the project so implementation of the opensource algorithms for the minutiae tensor matrix were difficult at the start. Eventually this issue was resolved.

The following are descriptions of how we have accomplished these goals and what difficulties we faced as well as how we overcame them. The numbering corresponds to the objective as stated above.

1. Develop Discriminator: The most important aspect of the authentication system was the neural network tasked with extracting the minutiae from the fingerprint data. The actual matching process was simplified for training purposes and for use in the early prototypes. The method of determining whether an extracted minutia was correct was inspired by the method used by FingerNet. In their design, a correctly extracted minutia was approved if it was within fifteen pixels of the location of the marked minutia from their professionally labeled training data. Since we wanted to ensure a successful implementation for our first prototype, we were more lenient with this matching and increased the radius of possible correct locations to 25 pixels from the true minutiae. This meant that the training took less time, although it led to a more imprecise process of minutiae extraction and meant that the GAN would have an easier time fooling the discriminator. For our purposes, this was not necessarily a bad thing as it was more of a proof of concept with the first prototype. The first discriminator network, as well as those that followed, was developed using Tensorflow. We started with a network that only had three layers within its neural network, and slowly ramped up the complexity with each improved iteration. Eventually our final collective discriminator trained on the fingerprint data with a network comprised of fifteen layers as opposed to the original three. We also decreased the margin of error that was acceptable to mark an extracted

minutia as correct. We could confidently take that step because the increased complexity of the neural network translated into a more finely tuned ability to extract the minutiae accurately. The final prototype discriminator was operating with nearly perfect accuracy and a very small false positive rate when trained on a complete and robust fingerprint dataset. This goal of the project was very thoroughly and successfully completed. We expected this to be the case, having FingerNet as an example to build off of and having the focus of the project be the Generative Adversarial Networks designed to fool the discriminator network.

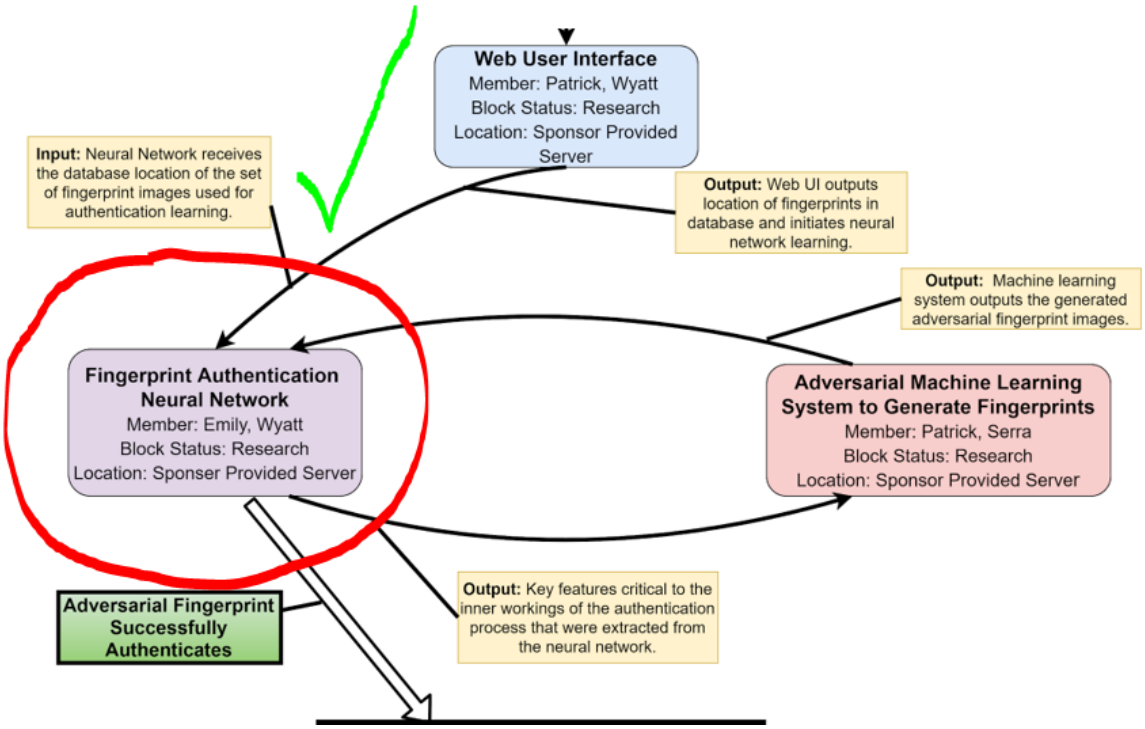


Figure 13.1 High Level Design Process 1

2. Develop GANs (Generative Adversarial Networks): To accomplish our goal of developing a GAN that could output master fingerprints, we had create a new deceiver neural network that would train off of the discriminator network we already implemented. The deceivers we built worked mostly off of limited information from the discriminator, and the fingerprint data in the database it was linked with. Each group member was tasked with creating their own version of a GAN. Members chose to use differing amounts of layers, types of layers, and preprocessing when implementing their own GAN. The goal was to have each member produce the most effective network they could, while working with varying levels of information that their network would have access to about the discriminator. Each group member successfully generated fingerprints that could fool the discriminator, but those that trained with more information from the discriminator neural network found success more quickly and at higher rates. The white box attacks were easily able to generate fake fingerprints that could be accepted, while the

contrasting black box attacks were met with little success. In a similar process to how we developed the discriminator, each member first started with a simpler prototype with a very small number of layers in the neural networks. This layering became more complex in the GANs as our group implemented new iterations of the networks. From the first prototype to the final product, we improved the efficiency and accuracy of our fingerprint generating models.

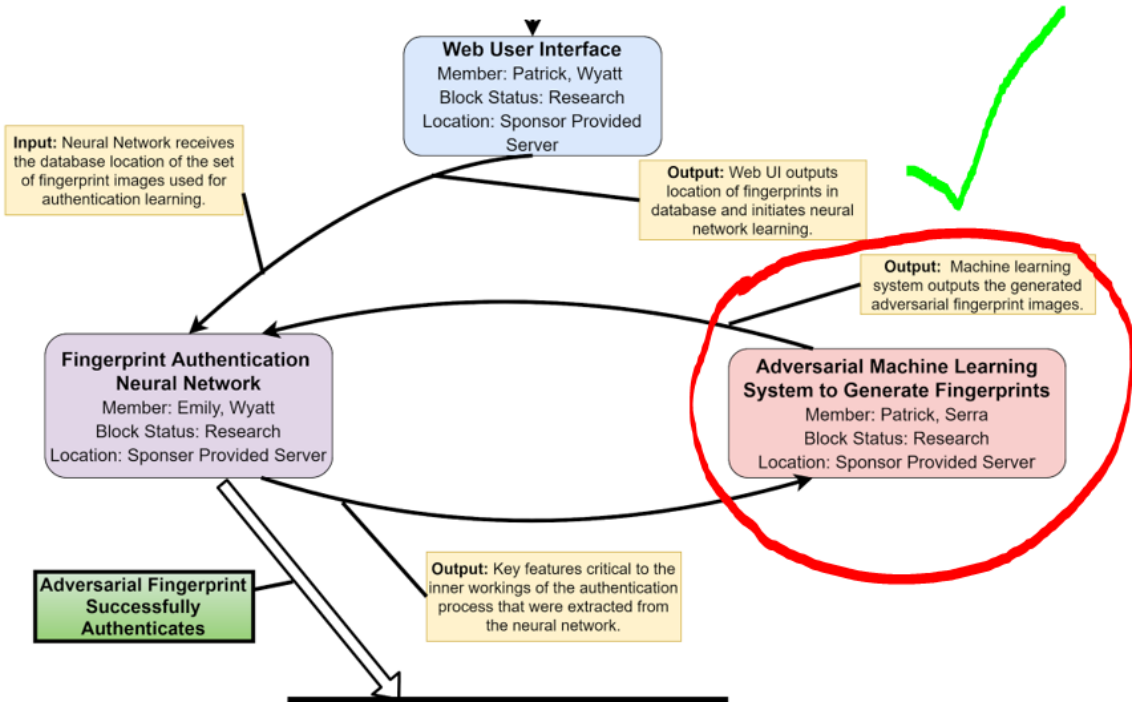


Figure 13.2 High Level Design Process 2

3. Less important than the GANs, but still essential to the project was the web interface that we created. In order to demonstrate the abilities and performance of our neural networks, we built a user friendly web application that required login credentials and allowed for verified users to upload their own fingerprint datasets and receive a master fingerprint that corresponded to that data once the training process was completed. The user is able to view statistics related to the training in real time as the process is taking place in the containerized backend. Once the training is complete, the user can view all information related to the training, including time to train, number of epochs, and the values from the loss functions as the neural networks become more optimized. No member of the group was particularly skilled with artistic design, so we used open source tools to develop the user-interface for the project. A key aspect of this section of the project was the operational processes of the backend. We developed a system to orchestrate the training of the neural network models by placing the in containers and using serving procedures to feed the training data to each model.

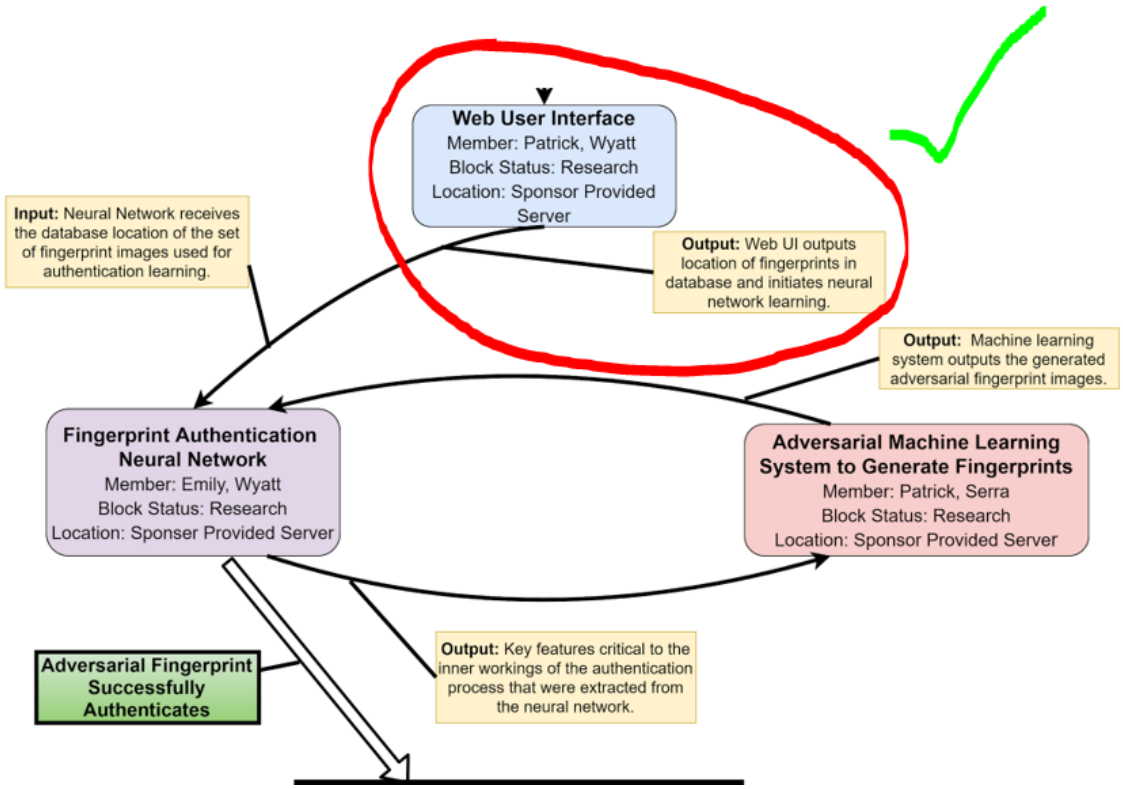


Figure 13.3 High level Process 3

All of the original requirements of the project were met as described above. Sadly, we did not meet many of our stretch goals, such as creating a 3D master fingerprint.

13.1 Conclusion

Our group's decision to undertake this project was guided by two main motivations: to further our understanding of machine learning systems, specifically generative adversarial neural networks, and to get a sense of the security risks and foolability of contemporary neural networks. Mel was driven by equal parts professional and personal curiosity. Projects that he was tasked with in his internship required knowledge of neural networks used for visual recognition of human shapes. He also imagined that he would develop skills and knowledge that could be applied in a more light-hearted way such as video game development. Serra came to the group with a cybersecurity background and a passion to discover how artificial intelligence systems could be exploited and how to address those vulnerabilities. She hoped that this project would open up new avenues of research and a better idea of what it would mean to be on the cutting edge of software security. Wyatt was drawn to this problem after considerations of the ethics and risks of AI technology. He believed in his responsibility as a software engineer to be cognizant

of the impacts of his trade and the vulnerabilities that new advancements may bring. He acknowledged that the way forward clearly should involve machine learning, but critical systems that will depend on these techniques will need to be integrated with very careful attention. Patrick sought to discover the processes and mechanisms of machine learning and to have a challenging and rewarding conclusion to his undergraduate education.

With diverse backgrounds and motivations, our group created common goals and a vision of what this project would achieve. Considering the broader picture, we wanted to demonstrate that machine learning systems were not impervious to exploitation. More specifically for this project, we attempted to show that fingerprint authentication systems based on neural network recognizers were susceptible to being fooled by other adversarial neural networks.

To achieve this goal, each individual member of the group created their own GAN (Generative Adversarial Network) to deceive the recognizer. The GANs were modified throughout the life of the project, mostly through a trial and error approach. The success of each GAN was determined by the ability of its fake fingerprints to be accepted by the authentication system. Higher acceptance rates meant that the GAN was closer to knowing what was needed to generate a convincing fingerprint.

As described previously, we hoped to generate a “Master Fingerprint” that could achieve a very high acceptance rate when presented to the recognizer. Some of our GANs came closer to this goal than others for various reasons. Different members of the group attempted to work with varying levels of information from the recognizer when implementing their GAN. This was intended and meant to demonstrate the various types of attacks that neural networks are exposed to in real world conditions. With some previously compromised information, it is easier to essentially reverse engineer the targeted neural network. Our work clearly reflected this, as the GANs with more insider knowledge learned more quickly, and ultimately performed better when deceiving the recognizer.

The GANs that operated under “black box” attack conditions expectedly performed the worst out of all. In a “black box” attack, the generative adversarial network is operating with no knowledge of the recognizer. Fingerprint images generated by this system were almost completely unrecognizable as fingerprints by the human eye. This was mostly true from the recognizer’s perspective as well. Despite Mel’s best efforts, this GAN only achieved an acceptance rate of about 9%. With the standards set by previous research, this was not nearly enough to proclaim that a master fingerprint had been generated.
The story was completely different when assessing performance of the GANs operating under “grey box” conditions. This is when the GAN has knowledge from the system it is trying to compromise, but that knowledge is limited. Obviously, there was some room for discretion when deciding what information this GAN would have access to. The “grey” nature of this attack was most realistic for real

world applications, but in our own implementations we leaned more towards a system with advanced knowledge of the recognizer. We were biased in this decision making, as we sought to have a valid demonstration of a “grey box” attack for our presentation. Under Wyatt’s direction this GAN was largely successful in producing a master fingerprint. It’s also important to note that we developed our own authentication system for testing, so this success may not translate completely to a commercial fingerprint recognition system. The “grey box” GAN produced a master fingerprint that had an acceptance rate of 79% when tested against our fingerprint database. This acceptance rate is clearly high enough to demonstrate the capabilities of a generative adversarial network to deceive a discriminator network. In a real-world situation, the attacker would need some information to have the kind of success we had, but that is not entirely out of the question with a devoted attack against a common system that has proliferated in the market.

The final attack condition we addressed was a “white box” attack. This means that the GAN has all knowledge of the discriminator and authentication system. This type of attack is not realistic as an attacker would generally not have all of this information, and if they did it would not be very helpful to try to train a GAN to deceive the system as it is already compromised. For our purposes, we used this style of attack more as a proof of concept than to show a legitimate security concern for commercial systems. Serra and Patrick were tasked with creating separate GANs under these conditions. The focus of these GANs was on efficiency of the training as it was easier for the GAN to successfully generate the master fingerprints. The average time to effectively train these GANs was 15 minutes 43 seconds. In this time period the models achieved an average acceptance rate of 98.7% from their generated master fingerprints. This achieved our goal of demonstrating the idea and ensuring that we had a working GAN to meet the project requirements.

The web component of the project was simple to complete and ended up integrating seamlessly with our neural networks. The containerization and serving system that Wyatt implemented on the backend made it easy to isolate and swap our training models. The web interface met the requirements set out by Dr. Jha, allowing users to upload their own fingerprint, which was then processed with Mel’s automatic annotation system (utilizing computer vision techniques) to prepare it for the training process. The data is then delivered to the containerized training models and once they are satisfactorily trained, the master fingerprint is sent to the user based on their corresponding data. This process took some time, so the results were sent to the user through email.

Connecting back to the overall goal of the project, we can confidently say that our project achieved its purpose. All GANs besides those under “black box” conditions were able to produce master fingerprints with at least a 79% acceptance rate by the discriminator neural network. This rate is comparable to the success of GANs in recent research papers on the topic. We were able to demonstrate the ease with

which these adversarial systems could be successfully developed, as our project was completed in under a year, with each group member taking care of outside obligations such as school work, internships, and other research. Our authentication system was developed by the group, and successfully integrated a neural network that could learn to extract the required minutiae with excellent accuracy. Our work clearly shows the inherent dangers of machine learning systems, especially those used for critical systems and infrastructure, including self-driving cars, government identification systems, etc. We were not able to implement very effective defenses against these attacks due to time constraints and the fact that this was a stretch goal for the project. Our hope is that the vulnerabilities that we have shown will spur action by those in the research and software engineering communities to look for better ways to create and defend their machine learning systems.

Looking forward, we all hope to continue researching and working in this area of computer science. Our group is planning to continue work on the original stretch goals of the project to delve more deeply into the defense side of these adversarial networks. Successful defense against “white box” attacks is nearly impossible, but there are ways to defend and against the more common “grey box” attacks that we have been interested in implementing and testing.

Overall, this project was a great adventure that we are all glad we could experience before graduating.

14. References

[1] “TensorFlow in Use,” *tensorflow.org*, Mar. 30, 2018. [Online]. Available: https://www.tensorflow.org/about/uses. [Accessed: Apr. 1, 2018]

[2] A. Nain, “TensorFlow or Keras? Which one should I learn?” *medium.com*, May 13, 2018. [Online]. Available: https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0. [Accessed: Apr. 1, 2018].

[3] “Architecture Overview,” *tensorflow.org,* Jan. 27, 2018. [Online]. Available: https://www.tensorflow.org/serving/architecture_overview. [Accessed: Apr. 1, 2018].

[4] “Using GPUs,” *tensorflow.org,* Jan. 27, 2018. [Online]. Available: https://www.tensorflow.org/programmers_guide/using_gpu. [Accessed: Mar. 2, 2018].

[5] “Premade Estimators,” *tensorflow.org*, Jan. 30, 2018. [Online]. Available: https://www.tensorflow.org/get_started/premade_estimators. [Accessed: Mar. 1, 2018].

[6] “A Guide to TF Layers: Building a Convolutional Neural Network,” *tensorflow.org*, Jan. 30, 2018. [Online]. Available: https://www.tensorflow.org/tutorials/layers. [Accessed: Mar. 1, 2018].

[7] “Deep Convolutional Generative Adversarial Networks with TensorFlow,” *github.com*, Oct. 24, 2017. [Online]. Available: https://github.com/dmonn/dcgan-oreilly/blob/master/DCGANs%20with%20Tensorflow.ipynb. [Accessed: Apr. 1, 2018].

[8] A. Kristiadi, “Generative Adversarial Nets in Tensorflow,” *wiseodd.github.io,* Nov. 17, 2016. [Online]. Available: https://wiseodd.github.io/techblog/2016/09/17/gan-tensorflow/. [Accessed: Mar. 1, 2018].

[9] felixTY, “felixTY/FingerNet,” *GitHub*. [Online]. Available: https://github.com/felixTY/FingerNet. [Accessed: Mar. 1, 2018].

[10] R. Verger, “Computer scientists are developing a 'master' fingerprint that could unlock your phone,” *Popular Science*, Apr. 13, 2017. [Online]. Available: https://www.popsci.com/computer-scientists-are-developing-master-fingerprint-that-could-unlock-your-phone. [Accessed: Feb. 29, 2018].

[11] Y. Zhang, “Transforming Autoencoder VS Convolutional Neural Network,” *Everything About My Thoughts*, Apr. 20, 2015. [Online]. Available: https://yingzh.wordpress.com/2015/04/15/transforming-autoencoder-vs-convolutional-neural-network/. [Accessed: Mar. 1, 2018].

[12] “Databases,” *docs.djangoproject.com.* [Online]. Available: https://docs.djangoproject.com/en/2.0/ref/databases/. [Accessed: Mar. 4, 2018].

[13] “Welcome to Flask,” *flask.pocoo.org.* [Online]. Available: http://flask.pocoo.org/docs/0.12/. [Accessed: Mar. 4, 2018].

[14] E. Nwankwo, “MongoDB, Express, AngularJS (1.6) and NodeJS (MEAN),” *blog.cloudboost.io*, Aug. 9, 2017. [Online]. Available:

https://blog.cloudboost.io/mongodb-express-angularjs-1-6-and-nodejs-mean-29f136f482a9. [Accessed: Mar. 23, 2018].
[15] "What is a Container," *docker.com*. Available: https://www.docker.com/what-container. [Accessed: Mar. 12, 2018].
[16] "What is a Container," *docker.com*. Available: https://www.docker.com/what-container. [Accessed: Mar. 12, 2018].
[16] "tensorflow/tensorflow," *hub.docker.com*, Mar. 1, 2018. [Online]. Available: https://hub.docker.com/r/tensorflow/tensorflow/. [Accessed: Mar. 12, 2018].
[17] "NVIDIA Container Runtime for Docker," *github.com*, Feb. 21, 2018. [Online]. Available: https://github.com/NVIDIA/nvidia-docker. [Accessed: Mar. 12, 2018].
[18] S. Arora, "Vue.js Is Good, But Is It Better Than Angular Or React?" *valuecoders.com,* Sept. 5, 2017. [Online]. Available: https://www.valuecoders.com/blog/technology-and-apps/vue-js-comparison-angular-react/. [Accessed: Mar. 6, 2018].
[19] "Frontend frameworks showdown: Angular vs. React vs. Vue," *clockwise.software,* Nov. 16, 2017. [Online]. Available: https://clockwise.software/blog/angular-vs-react-vs-vue/. [Accessed: Mar. 8, 2018].
[20] "Link Functions in AngularJS," *c-sharpcorner.com,* Mar. 21, 2016. [Online]. Available: http://www.c-sharpcorner.com/article/link-function-in-angularjs/. [Accessed: Mar. 6, 2018].
[21] I. J. Goodfellow, J. Shlens, C. Szegedy, "Explaining and Harnessing Adversarial Examples," Mar. 20, 2015. [Online]. Available: https://arxiv.org/pdf/1412.6572.pdf. [Accessed: Mar. 12, 2018].
[22] A. Athalye, L. Engstorm, A. Ilyas, K. Kwok, "Fooling Neural Networks in the Physical World with 3D Adversarial Objects," *labsix.org,* Mar. 31, 2017. [Online]. Available: http://www.labsix.org/physical-objects-that-fool-neural-nets/. [Accessed: Mar. 21, 2018].
[23] P. Bontrager, A. Roy, J. Togelius, N. Memon, "DeepMasterPrint: Fingerprint Spoofing via Latent Variable Evolution," Feb. 16, 2018. [Online]. Available: https://arxiv.org/pdf/1705.07386.pdf. [Accessed: Mar. 12, 2018].
[24] J. Bruner and A. Deshpande, "Generative Adversarial Networks for beginners," *O'Reilly Media*, Jun. 7, 2017. [Online]. Available: https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners. [Accessed: Apr. 3, 2018].
[25] Ujjwalkarn, "An Intuitive Explanation of Convolutional Neural Networks," *The Data Science Blog*, May 29, 2017. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Accessed: Apr 2, 2018].
[26] C. Bupe, "Why do neural networks need an activation function?," *Quora*, Aug. 13, 2016. [Online]. Available: https://www.quora.com/Why-do-neural-networks-need-an-activation-function. [Accessed: Apr. 2, 2018].
[27] CS231n Convolutional Neural Networks for Visual Recognition. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: Mar. 29, 2018].

[28] N. Shibuya, “Understanding Generative Adversarial Networks – Towards Data Science,” *Towards Data Science*, Nov 3, 2017. [Online]. Available: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef. [Accessed: Mar. 20, 2018].
[29] Soumith, “ganhacks,” *GitHub*, Dec.16, 2016. [Online]. Available: https://github.com/soumith/ganhacks. [Accessed: Mar. 23, 2018].
[30] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. McCabe, S. Janet, and K. Ko, “User's Guide to NIST Biometric Image Software (NBIS),” Oct. 5, 2004. [Online]. Available: https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=51097. [Accessed: Mar 4, 2018].
[31] Y. Adi, M. Cisse, J. Keshet, N. Neverona, “Houdini: Fooling Deep Structured Prediction Models,” Jul. 17, 2017. [Online]. Available: https://arxiv.org/pdf/1707.05373.pdf. [Accessed: Mar. 9, 2018].
[32] “PostgreSQL vs MySQL,” *2ndquadrant.com.* [Online]. Available: https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/. [Accessed: Mar. 15, 2018].
[33] E. Levy, “Postgres vs. MongoDB for Storing JSON Data – Which Should You Choose?” *sisense.com*, Nov. 22, 2017. [Online]. Available: https://www.sisense.com/blog/postgres-vs-mongodb-for-storing-json-data/. [Accessed: Mar. 15, 2018].
[34] “Compose Compare: MongoDB vs. PostgreSQL” *compose.com*, Oct. 10, 2017. [Online]. Available: https://www.compose.com/articles/compose-compare-mongodb-vs-postgresql. [Accessed: Mar. 15, 2018].
[35] Ravi, “Ext4 file system how to, tips and tricks,” *aboutLinux.info*, Jun. 26, 2009. [Online]. Available: http://www.aboutlinux.info/2009/06/ext4-file-system-how-to-tips-and-tricks.html. [Accessed: Mar. 24, 2018]
[36] X. Fu, C. Liu, J. Bian, J. Feng, H. Wang, and Z. Mao. Extended clique models: A new matching strategy for fingerprint recognition. In Biometrics (ICB), 2013 International Conference on, pages 1–6. IEEE, 2013.
[37] K. Stanley, Ph.D., “The NeuroEvolution of Augmenting Topologies (NEAT) Users Page,” *NeuroEvolution of Augmenting Topologies*, May 5, 2015. [Online]. Available: https://www.cs.ucf.edu/~kstanley/neat.html. [Accessed: Mar 2, 2018].
[38] FVC2002 - Second International Fingerprint Verification Competition. [Online]. Available: http://bias.csr.unibo.it/fvc2002/databases.asp. [Accessed: Apr. 5, 2018].
[39] CoreUI - Open Source Bootstrap Admin Template. [Online]. Available: https://coreui.io/demo/Vue_Demo/. [Accessed: Apr. 20, 2018].
[40] N. Foundation, Node.js. [Online]. Available: https://nodejs.org/en/. [Accessed: Apr. 8, 2018].
[41] “Express - Node.js web application framework,” *Express - Node.js web application framework*. [Online]. Available: https://expressjs.com/. [Accessed: Apr. 20, 2018].

[42] “30.19. Example Programs,” *PostgreSQL*. [Online]. Available: https://www.postgresql.org/docs/8.3/static/libpq-example.html. [Accessed: Apr. 10, 2018].
[43] “Python,” *Python - PostgreSQL wiki*. [Online]. Available: https://wiki.postgresql.org/wiki/Python. [Accessed: Apr. 10, 2018].
[44] “Psycopg2 Tutorial,” Psycopg2 Tutorial - PostgreSQL wiki, Apr. 9, 2015. [Online]. Available: https://wiki.postgresql.org/wiki/Psycopg2_Tutorial. [Accessed: Apr. 19, 2018].
[45] N. Nuemah, J. Roth, A. De George, C. Lin, and Tompratt-AQ, “SQL Server Application Patterns on VMs,” *SQL Server Application Patterns on VMs | Microsoft Docs*, May 31, 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sql/virtual-machines-windows-sql-server-app-patterns-dev-strategies. [Accessed: Apr. 4, 2018].
[46] J.C.T. Rogers III, “Protecting Your PostgreSQL Database From SQL Injection Attack With Prepared Statements,” Oct. 25, 2016. [Online]. Available: https://www.linkedin.com/pulse/protecting-your-postgresql-database-from-sql-attack-rogers-iii/. [Accessed: Apr. 13, 2018].
[47] “Fingerprinting Lesson,” *Forensic Science Investigation Unit*. [Online]. Available: https://forensicunit.weebly.com/fingerprinting-lesson.html. [Accessed: Mar. 27, 2018].
[48] M. Gomez-Barrero, Ed., “INSTRUCTIONS FOR DOWNLOADING ATVS-FFp DB,” *Biometric Recognition Group - ATVS, 2015*. [Online]. Available: https://atvs.ii.uam.es/atvs/ffp_db.html. [Accessed: Apr. 8, 2018].
[49] Lady Ada, “Adafruit Optical Fingerprint Sensor,” *Adafruit Learning System*, Feb. 28, 2018. [Online]. Available: http://https://cdn-learn.adafruit.com/downloads/pdf/adafruit-optical-fingerprint-sensor.pdf. [Accessed: Apr. 22, 2018].
[50] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: a simple and accurate method to fool deep neural networks,” [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/app/S12-10.pdf. [Accessed: Apr. 1, 2018].
[51] V. K. Sagar and K. J. B. Alex, “Hybrid fuzzy logic and neural network model for fingerprint minutiae extraction,” IEEE Conference Publication, Jun. 16, 1999. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/836178/. [Accessed: Mar. 14, 2018].
[52] Bontrager, Philip, Roy, Aditi, Julian, Memon, and Nasir, “DeepMasterPrint: Fingerprint Spoofing via Latent Variable Evolution,” [1705.07386], Feb. 16, 2018. [Online]. Available: https://arxiv.org/abs/1705.07386. [Accessed: Apr. 15, 2018].
[53] “NVIDIA Tesla V100,” *nvidia.com*. [Online]. Available: https://www.nvidia.com/en-us/data-center/tesla-v100/. [Accessed: Apr. 19, 2018]

15. Appendix

15.1 Copyright

15.1.1 MIT License