LOGISTIC NOISE AND LEAST SIGNIFICANT BIT STEGANALYSIS THROUGH MACHINE LEARNING

Rajvir Vyas
Computer Science
Cal Poly San Luis Obispo
California, United States
rvyas01@calpoly.edu

Dr. Sumona Mukhopadhyay
Computer Science
Cal Poly San Luis Obispo
California, United States
mukhopad@calpoly.edu

***Abstract*—This research focuses on the idea of Least Significant Bit Steganograhy, and how Deep Learning can be implemented to perform detection to prevent images with malicious content from being distributed. This paper aims to provide a background on what the problem is, current solutions regarding this field, as well as a look at a potential solution entailing Convolutional Neural Networks for this purpose and a discussion of how Logistic Chaos Maps can be utilized to further the encryption of steganographic images by adding noise, which makes detection problems even harder. Overall, I aim to educate individuals on these topics and showcase through trial and error the determination needed to 'solve' such a problem and create discussion on why it needs solving.**

I. Introduction

With the onset of deepfake technologies and advancements in AI-generated art, it is getting increasingly more difficult to tell whether or not a picture you are looking at is real. Furthermore, the art of steganography is a popular method to hide information within seemingly normal files. They can consist of simple text messages as ciphers or worse, have shell scripts embedded within, which can harm users when they try to open the innocent-looking files. This is why detecting such images is a very beneficial task- we can literally prevent hidden information from being transferred from an attacker's machine and save potential victims of cybercrime.

I plan on working on this problem using Deep Learning, more specifically through the use of Convolutional Neural Networks for image classification. I want to develop a model that is capable of detecting whether there has been steganographic usage within an image, and describe use cases for this research. Furthermore, I want to explore the addition of chaos noise for more enhanced encryption of the images and discuss how it affects model performance as well.

II. Motivation

I want to solve this problem because I find the intersection of machine learning and cryptography to be especially relevant today, with how much development there is in the field. There is always going to be a need for secure means of sharing information, and cyber privacy is a strong ideal to uphold.

The novelty of utilizing chaos maps in order to encrypt steganographic images further is also interesting to me, as it details a unique way to encrypt information. It's research applications have been interesting as well, from the countless papers published online about the usage of chaos maps.

This project serves to provide insights to stakeholders such as online content consumers, and technology-averse individuals- AI art and videos are spread everywhere now and as aforementioned, there are potentially malicious images and videos out there pretending to be innocuous. By helping consumers know what is real and not, and further what is safe or not, I think at a minimum, I can

educate people to be more cognizant of such threats online.

III. Data Collection

After exploring multiple ideas to gather data such as Imagenet, BOSS, and Alaska2 by Kaggle, and trying to gather those images and take a subset from those myself, I ultimately utilized a different approach.

After my literature review process, I continued looking for more resources for assistance in understanding this project. One of the papers that I will discuss in that section, titled "Detection of Steganographic Threats Targeting Digital Images in Heterogeneous Ecosystems Through Machine Learning" was particularly of note to me, since the researchers of that paper decided to share their dataset on Kaggle [1].

This dataset (referred to as the S.I. Dataset henceforth) contained a total of 44,000 images that were 512x512 pixels big and contained a variety of malicious embeddings such as "...Javascript and HTML code, Powershell scripts, URLs, and ehtereum addresses..." [2]. Even better is the fact that they use Least Significant Bit as their embedding technique, which is the one I was hoping to explore as well.

Now their data is split up into train, validation, and test folders, with subfolders for the 'classes', i.e. a subfolder for 'clean' images and one for 'stego' images. When I tried using this approach for myself, I had some difficulty, and experienced lengthy wait times to train my custom convolutional neural network model. After a consultation with Dr. Mukhopadhyay, I was able to reject their test and validation sets, and instead split apart their training dataset into my own test and validation sets, using a 70/15/15 (train, validation, test) split between the original dataset and mine. I should also mention that the training data was not lacking by itself, as they had 16,000 images present for the training data, consisting of 4000 clean, and 12,000 stego images. However, after the split, there were 11,200 images for training, and 2400 images each for validation and testing. I also made sure to shuffle the dataset before merging them altogether.

As for the chaos noise data, I had to get a little more creative.ergjoidjgolqgnleknvknlkvnl

IV. Background/Theory

As mentioned previously, Steganography is the practice of hiding data within files that look visually indistinguishable to the human eye from an unaltered file. This practice can involve image files, video files, and text files, to name a few. I particularly wish to observe this effect within image files. One popular technique of encryption is through Least Significant Bit (LSB) Encoding, wherein you can replace or match the least significant bits of pixels in an image with the bits of the text we want to hide within it.

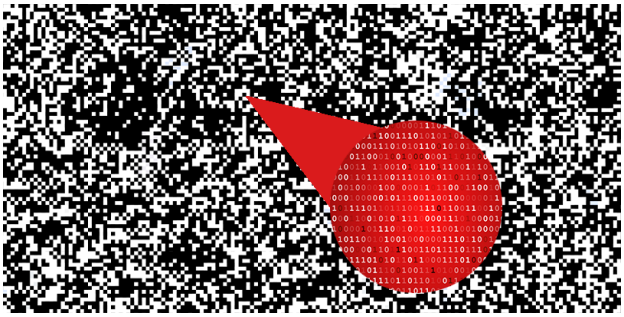

Fig. 1. Example of Steganography [3]

Out of these two approaches, LSB matching is more involved, as instead of simply replacing the image's bits directly, you use a secret key to randomize the pixels in the original image. After this, you would try to match the randomized pixels with the bits of the text we want to hide. Only if there is a mismatch between the pixels would you change the original image's pixels for the hidden data [4].

So how do you detect steganographic use in an image if it is meant to be indistinguishable visually from an unaltered image? The key point here is, this change mostly only affects humans, and that it's harder to trick computers into believing that the two images are identical. This practice can be called Steganographic detection or Steganalysis. Taking a look at detecting the LSB matching technique that I described earlier, we see that there is increased difficulty with detecting it. This is because using

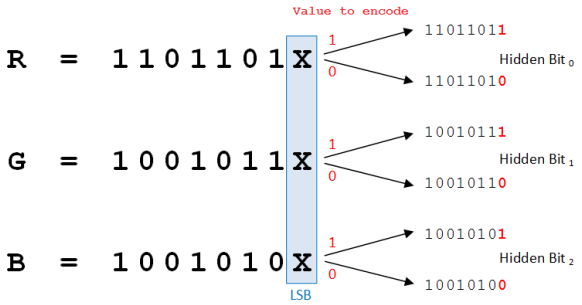

Fig. 2. Example of Least Significant Bit Steganography [5]

the secret key adds a layer of pseudo-randomization which makes it more difficult to detect than just the LSB replacement technique.

However, through research, I have found that a popular technique to detect LSB matching is through feature-based classifiers and decision trees. That is, it focuses on extracting specific features from the modified images and training classifiers through some heuristic/technique to distinguish those images from ones that are unmodified [6]. This is a technique that I intended to observe more closely when coming up with my own methodology.

Furthermore, I utilized chaos-based systems, which can create seemingly random behavior from deterministic processes by making tiny adjustments to the initial conditions. More specifically, I used logistic maps, denoted by the equation:

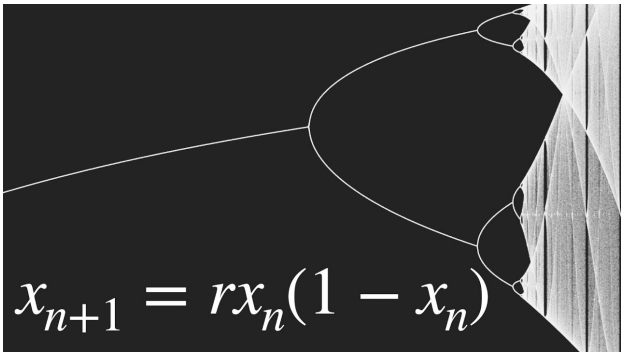

Fig. 3. Logistic Map Equation [7]

which is a simple way to generate so-called "randomness". As one can see, by changing the parameters and iteratively applying the equation, we can introduce noise. I used this in conjunction with a randomly generated time series to add a further layer of security to the message.

V. Literature Review

There are a multitude of techniques for both encryption and detection of steganography for images. My research into this topic led me down many paths. In this section, I will go through some of the most interesting and/or pertinent articles one by one.

Article 1: Usage LSB Method in Hiding Text Information within Text in an Image [4]

Audience: I believe this article is more aimed at cybersecurity professionals and researchers as it focuses more on the security aspect

Short Summary: This paper details a technique of first applying text-based steganography, and then using the LSB technique to embed the text within an image. More specifically, this approach focuses on increasing the capacity of the embedded text so that it can carry more letters while also having a reduced number of pixels that are needed to hide the text in the image.

Unique: Something new that I learned about from this paper was the usage of the Peak Signal-to Noise Ratio (PSNR) Which is used as a metric to measure the error rate between the original image and the image with the embedded text.

Article 2: Least significant bit steganography detection with machine learning techniques [6]

Audience: I believe this paper is also aimed towards researchers, and practitioners of steganography.

Short Summary: This paper is focused more on the detection side of things, namely using machine learning to detect image-based steganography i.e. the LSB technique. More specifically, this approach focuses on training classifiers on features that are extracted from the images and using something like a chi-square feature-based decision tree to perform

sequential LSB detection, which proved to be more fruitful for them.

Unique: Something new that I learned about from this paper was the usage of the Peak Signal-to Noise Ratio (PSNR) Which is used as a metric to measure the error rate between the original image and the image with the embedded text.

Article 3: Generative Steganography Diffusion [8]

Audience: The audience here is researchers and scientists who are interested in novel approaches of steganography

Short Summary: The authors of this paper have created a new technique called GSD, and have proposed a new model which supposedly allows for 100 percent recovery of hidden secret data whilst generating realistic steganographic images. More specifically it uses a Non-Markov chain and an ordinary differential equation based approach in order to outperform existing methods .

Unique: This article carries the benefit of allowing image generation, something that is useful for a latter part of my project.

Article 4: Undetectable Watermarks for Language Models [9]

Audience: Audience: The audience here is researchers concerned with natural language processing specifically within the context of security and detectability of AI generated text.

Short Summary: This paper is more focused on the difficulties in detecting AI generated texts through models such as gpt4. In order to combat this, they propose a concept of undetectable water marks for language models that can only be identified using a secret key which would ensure that even with adaptive queries users don't observe any degradation in text quality.

Unique: This article discusses some practical use cases of steganography encryption for images, namely watermarking them, but one can even go beyond and apply watermarks in other contexts, such as LLM models, which have been discovered to use it to hide their reasoning.

Article 5: Detection of Image Steganography Using Deep Learning and Ensemble Classifiers [10]

Audience: The audience here would be professionals and practitioners specifically interested in deep learning based techniques for steganalysis.

Short Summary: This is perhaps the most technical paper of them all, with it mainly addressing the challenge of detecting steganographic jpegs. It utilized images from the BOSS dataset, which is something to investigate. As far as techniques go, they utilized discrete cosine transform residuals (DCTR) and Gabor filter residuals (GFR). I'll admit this one is a little more complicated still to understand for me.

Article 6: Detection of Steganographic Threats Targeting Digital Images in Heterogeneous Ecosystems Through Machine Learning [1]

Audience: The audience for this paper is machine learning researchers who want to be able to utilize this paper as a benchmark. Once again, the audience should be aware of deep learning techniques in advance!

Short Summary: This paper was the one that laid the foundation for this quarter's research. It introduces their dataset early on, letting the readers know that it is available for them through Kaggle. It does a good job of explaining what Least Significant Bit Steganography is, as well as showcases a robust solution for detecting malicious scripts embedded within photographs with remarkably high accuracy and precision.

VI. Initial Plans

My initial plans were far too ambitious for the project, focusing on 3 key areas. The primary one being a steganography embedding and

detection web application, a secondary one being a steganographic authentication system that would let users access their AWS account based on if their photo had an embedded ”certificate” or not, and lastly attempting to use transfer learning to train a model using the Kaggle Dataset aforementioned and then apply a new generated dataset to run simulations on. Unfortunately due to both time restrictions as well as a high level of error from the models' side of things, I was unable to get to any of these ideas, though I do believe they are worth exploring in the future, hence why I mentioned them.

VII. Quarter 1 Deliverable

Last quarter I embarked on the journey to understand LS steganography in terms of how it works in encrypting and decryption images. More specifically, I Was able to create a tool that works on the command line and uses LSB stego along with chaos systems to add noise to the image for further security.

My tool uses randomly generated time series data in a logistic map chaos system to scramble the secret message, and then embeds the message within a cover image using the least significant bit steganography technique. This is referred to as the encryption stage. For the decryption stage, I can provide a stego image, and by reversing the methodologies aforementioned, the tool outputs the secret message embedded within.

One of the quirks I was working with is sizing constraints, because currently, the string has to be a set length for the message to be decrypted successfully. Sizing also comes into play during the encryption stage, where I am adding noise using the time series data, so it is an imperative task that needs to be made more general.

The results from this part were still fruitful, as showcased by an encrypted image using my model in Figure 4.

This is good because there's nothing visually noticeably different in this image that would indicate

Fig. 4. Dog Embedded with ”Hello, I'm a Cat”

that there is a word hidden inside of it. However when comparing the pixel intensity by frequency for this stego encrypted image and the original cover image, we can see that there are subtle differences between the two in Figure 5.

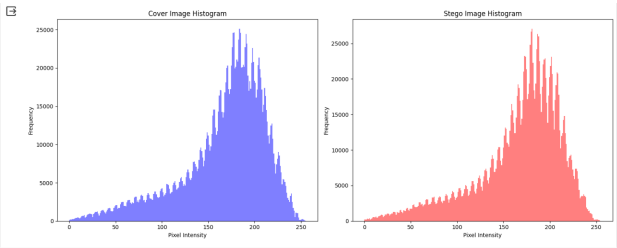

Fig. 5. Pixel Intensity By Frequency

This shows me that my program is working correctly, and that for detection, I will have to analyze some specifics of an image's metadata or pixel values steganography. to determine the use of Some other results and conclusions that I was able to draw was that detect-ability was dependent on message size, i.e. the longer a message, the more detectable it is in the image. This is due to some messages being longer than the embedding capacity of an image, which is something that can be dynamically computed to my knowledge. The same relationship applied for image resolution-poorer quality images made the embedding more apparent.

Now why did I mention all of this work that is seemingly disconnected from the machine learning research that I worked on this quarter? The answer

is because this work was important in helping my understanding of steganography and further I did end up using it again for this quarter's project. Specifically I utilized the logistic map based chaos noise from this project since I needed to add that noise to the training data and compare whether adding noise affected the machine learning models performance at all. So, later on, I will discuss this in more detail.

After that recap of everything from the last quarter to some new occurrences this quarter, let us move on to the methodology that was utilized for this quarter's project.

VIII. Methodology

As you may have noticed within the literature review section, there were quite a few journals, and research papers that I read through for inspiration. However when it came time for implementation, it was much tougher to find examples and resources to learn from to implement such detection tools. The reason being that a lot of the papers utilize more advanced techniques than the ones I was familiar with such as making use of generative diffusion models, complicated neural networks, pre-trained image classification models etc. This makes sense as a lot of these papers were set in a professional context or in a graduate school level context, But this definitely made things more difficult when trying to come up with my own implementation.

A. Note on File Structure

I must mention that the S.I. Dataset on Kaggle came with a training, testing, and a validation set each, and moreover these folders have subfolders within themselves labeled as "clean" or "stego" denoting the class of the image. This was a unique way (at least for me) to let the model know what image belonged to what class, as it would make the class based on whether the image belonged to subfolder 1 or subfolder 2. However, this is not to say this was without its issues; more on that later.

B. Convolutional Neural Network Implementation

Ultimately, I tried to approach a Convolutional Neural Network model as discussed previously, despite not having any deep learning background, mostly inspired from Article 6 [1] of my literature review. As per my research, I learned that a Convolutional Neural Network (or CNN for short), is a type of model that attempts to mimic the processing of the human brain. Like our brain, it uses multiple connected layers (for us its neurons) to process information. A CNN particularly is best at processing structured data like images, making it a perfect choice for this problem. Each layer as I mentioned performs different operations such as convolution, pooling, and nonlinear activation to perform the learning.

For example, a convolutional layer applies filters across the input data, which would allow it to find the spatial trends, kind of like how our brain is able to make sense of an object's shape as it moves towards us. Pooling layers downsize all the input to only what it needs to understand the object/image. The non-linear activation function basically just lets us make predictions about data that is not organized in a linear fashion (which is often the case with data like images). [12] Here is an example of how a convolutional neural network works (as visuals always help in clarifying concepts):

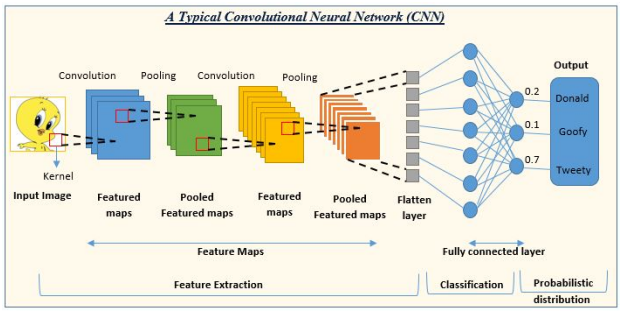

Fig. 6. CNN to determine which cartoon character it is [12]

In this figure, we can see that multiple layers for convolution and pooling have been generated in a process known as feature extraction and after all the layers have been created and flattened, they can be used to classify what cartoon character the

image is supposed to be [12].

As for how my CNN was set up? I utilized TensorFlow, ScikitLearn and Google Colab in order to handle this task. Firstly I set up a function called "weighted binary cross entropy", which was a custom loss function that was meant to particularly penalize errors in predicting stego images versus clean images. This was because if we were going to use this model for the authentication task as previously mentioned, then it is important we classify all cases of stego as actually being stego. After this I use a sequential model where in each layer has one input and output that gets built upon one another to build my CNN.

```
def create_custom_cnn(input_shape=(512, 512, 3)):
    model = Sequential([
        Conv2D(16, (3, 3), activation='relu',
               input_shape=input_shape,
                padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

        Conv2D(32, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
```

Fig. 7. Custom CNN I created

I included a convolutional layer, a batch normalization layer, Max pooling layer, flattening layer dense layer, and a Dropout and regularization layer. So once again, to explain the Conv2D layer is applying filters with the ReLU activation function to the image. Then, Batch Normalization exists as a way to help speed up the training process. The Max Pooling reduces the target size to 256x256 from 512x512, which was the original image size. Flattening just means to make my 3D output into a one-dimensional array which becomes the Dense layer that is used for binary classification in this case (through the use of the sigmoid activation function). I added the Dropout and Regularization layers to prevent overfitting of my training data.

Now to pass on my data into the model itself I needed to create data generators- specifically image data generators, which are features built into Tensorflow's Keras package. This allowed me to perform image augmentation on the dataset. Specifically, I was able to normalize the pixel values to a range from 0 to 1, then change the brightness of images in subtle ways (like stego images often have, as we observed in the pixel intensity graph from last quarter). And I was also able to manipulate the color channels.

That concludes the work that was required for the steganographic detection. Now, as for the chaos-map embedding, I utilized my logistic map and apply-noise functions from last quarter to add noise to a subset of the images from the dataset and then proceeded to train on that subset. There is rationale behind this as well as I will discuss in the results and challenges section.

IX. Results

Now it's time to discuss the mostly frustrating results that this experiment led to.

Firstly, after running the custom CNN model on the training data, for 10 epochs, which took about 90 minutes running on Google Colab's T4 GPU and longer on the Cal Poly Workstation, I obtained about a 74 percent training accuracy with a loss of about 1.03 and a validation accuracy of about 75 percent with a loss around 1.01. Since throughout the training, the training loss went down, I can confirm that the model did end up learning from the dataset. This is further reflected by the training accuracy increasing. Even the validation accuracy improved, meaning the model can handle data it hasn't seen before. There doesn't really appear to be any overfitting going on either.

However when I try to run the testing data, I am greeted with a testing accuracy of about 60.3 percent which looks decent on paper, but then you look

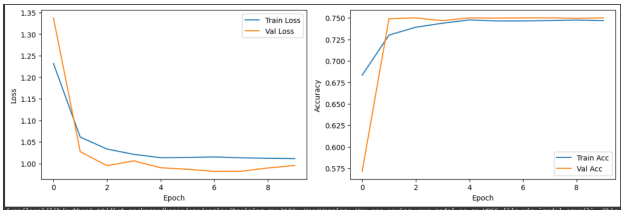

Fig. 8. Loss and Accuracy Visualizations

at the confusion matrix and realize that the model is only really able to detect steganographic images successfully, because clean images are barely represented within the confusion matrix. This isn't good, because this means we gave more weight to the steganographic images over clean ones. Unfortunately in trying to prevent an imbalance, we ended up causing one.

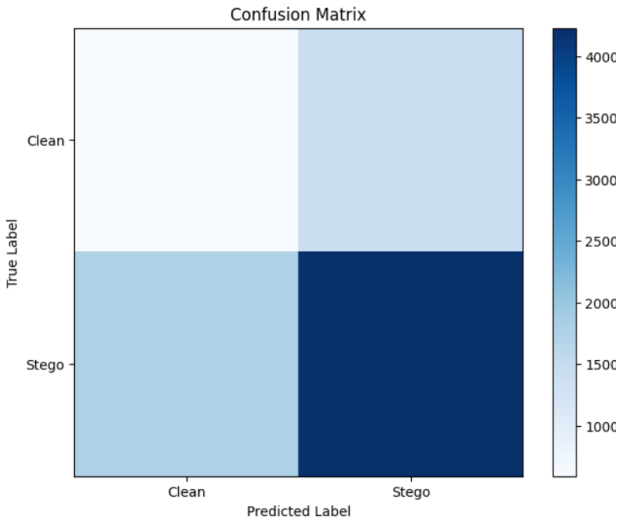

Fig. 9. Confusion Matrix

Next, with the same model, I added in the chaos noise functionality so that a subset of the data is embedded with that noise and then we attempt to detect if there has been any steganography added to the image. The issue with this, however, was that the training time(3-9 hours) was far longer than I had the capability to run it. Thus, unfortunately, I was unable to run it fully before the deadline of my study.

X. Challenges

Throughout this project, I dealt with several challenges with implementation, training, data collection and more. Here I will highlight the main concerns and discuss what I could have done to improve.

A. Time Constraints

The biggest challenge I had in completing this project was the lack of time. Time was a truly limited resource in this project because of two main reasons. I did not find a proper dataset until midway through this quarter. This made it difficult to get familiarized with it and understand its intricacies. One of the main issues I had with the existing data set was its organization.

For example, the training data was organized as '/content/train/train/clean'. This indicates that there was originally a train folder, within it is another train folder (for whatever reason), and then there's another subfolder called 'clean' and another subfolder called 'stego', which determine how the classes are labeled. Preserving this kind of formatting is extremely tedious as I discovered with multiple manipulations of the directories using the os package. You cannot use the ImageDataGenerator methods if your data isn't organized in this way, instead you would need to define your own custom data generator class and methods.

So why wouldn't I just use this dataset the way it is and not deal with that added complexity with potential for error? That ties into the second reason, which is that since there is a lot of data, the model takes a really long time to train. For example, a series of 10 epochs would end up taking 90 minutes regularly, and if you messed up somewhere, that's 90 minutes wasted, since you would need to retrain the model. This was on the workstation as well, so that ended up being really frustrating.

After discussing with Dr. Mukhopadhyay she recommended I split the training data they provide into my own training, test, and validation sets, to make it easier to run. Also it helped me avoid the weird fact that they had 2 classes each for training and validation (clean and stego), but for

the testing, they used 4 classes (clean, stego, stegozip, stegob64). But by shuffling the clean and stego values together into one training set, I had to forgo the preexisting clean and stego folders for classes, meaning I had to create a custom data generator that assigned classes based on filename. This process took extremely long just for it to not perform as expected. So, I decided to stick with the whole dataset as before so that I have something to show for my model, as the other method just led to my model making random guesses (as observable by a AUC of .5) due to issues with the way the data has to be organized.

To try and solve this issue, I ultimately purchased Google Colab Pro to gain access to faster GPUs.I was originally using Google Colab's free tier which comes with limited TPU usage, and I could barely run one runtime of the model before my access was revoked due to overuse. Thus, I truly believed faster GPUs would help me achieve this task quicker. However this is not as fruitful as I hoped, at best I was saving a minute or two, but to add more trouble, Colab would frequently freeze up its runtime causing me to restart the whole process, or it would fail to load the directories I needed. All in all, in 2 days, I went from 100 compute units to 38 compute units, due to countless such mishaps.

In the future, I would much rather consider two possibilities- one would be to find a better dataset in advance, i.e. one that I would have made myself more familiar with. And the second would be to train the models on a faster machine, locally, rather than through the cloud or the remote server.

B. Issue of Deep Learning

This concern was mainly due to me having no prior experience in Deep Learning, and so everything I was doing was at a rate much slower than usual as I would have to refer to countless tutorials and documentation for TensorFlow and CNNs. While I truly believe that making mistakes and fixing them is the best way to learn, when there is a time constraint placed on you and you encounter countless foreign errors, it does get difficult.

However, this is not to say that I didn't gain anything from this experience. I truly grasped more knowledge about neural networks like CNNs, and have a better sense of how they function and how to create basic ones. Fine-tuning them is where more experience and time could have helped. In fact, I think it would have been really beneficial, if I had emailed the authors of the paper and clarified parts of their research that was challenging for me to understand; I know now that this is actually encouraged to fully understand the outside research that you are attempting to build upon.

C. N Number of Research Directions

Lastly, since I got a late start on my project, there were actually multiple different routes I wanted to take this project and its research, where I expended time but it ultimately didn't prove to be useful for the moment. For example, I have python files to generate my own testing set with NBA players that are embedded with LSB and Chaos techniques from last quarter that I wanted to apply transfer learning with on a pretrained model like ResNet50, or VGG16. Similarly, I created a whole AWS pipeline for how I wanted the authentication method to work out and even built a small Flask application to handle simple image file uploads. These ideas were executed all prior to me getting my dataset, which in hindsight was not the wisest thing to do, since these components weren't as critical as the actual machine learning model.

But then again, in the time it took me to find a proper dataset (since originally I believed I may have to generate my own), I couldn't just do nothing, so I had to make use of that time as well. And even after I got my dataset, I tried so many different approaches to making this model work. These are showcased by the multiple collab notebooks on my repository, along with multiple iterations of commented out code since things would take too long to run to fully analyze properly. In the future, it would have helped to have a better understanding of exactly what it was that I was trying to solve,

rather than spending too long in the ideation stage and rejecting those ideas.

XI. Discussion and Conclusion

However, all things considered, I do not think this research was a failure. Yes, it was disappointing that I wasn't able to achieve the goals that I had set for myself with this research, but in the short time we did get, I was able to absorb a lot of info, especially information that will be handy for me to continue this project.

Spending two quarters with this project idea and countless hours messing with the intricacies of the model, I believe now I truly understand the problem. Yes, I was unable to achieve high success with detecting clean images, but I was able to detect ALL of the steganographic images due to the weightage error. While this isn't ideal, it is better to prevent True Negatives than allow them to be undetected. And the issue here should be simple to fix, seemingly, I have to switch around the class weights to give more balanced importance to both the clean and stego images. I chose the original weights due to their being a mismatch in the training and testing data regarding the classes, but I see now that it didn't really help.

Another positive is that my least significant bit and logistic chaos map noise embedding system truly works- fully demonstrating its potential within exploring how a model would compare in detecting noisy stego images. Again, with a different dataset, I sincerely believe the model would perform better, since the embedding part works properly, and since we compare the subtleties in pixel differences for images with augmentation, adding noise, in theory, should make it less detectable, since that pattern will be harder to detect.

Lastly, all the small tidbits I have built for this project- from the pretrained models to the web interfaces- are all great springboards to realizing the true potential of this project via implementing the steganography detection features in a practical sense for authentication or watermarking, etc.

So rather than take this is in a negative light, I choose to look at the positives- my improved understanding of machine learning research of steganalysis, how neural networks work, and functioning tools for encryption, detection, and models with good training accuracy that are just in need of a little fine tuning.

We are meant to talk about how our research can help others, and I believe this paper and my repository will serve as proper resources for people to see common fallacies as well as starter code for models and encryption/detection systems. However, I believe that even I will/have benefit(ed) from furthering my work in this project. Knowing what I know now, and without a looming deadline, I am certain that I can properly showcase why this topic is worth researching and discussing further.

References

[1] N. Cassavia, L. Caviglione, M. Guarascio, G. Manco, M. Zuppelli, "Detection of Steganographic Threats Targeting Digital Images in Heterogeneous Ecosystems Through Machine Learning", Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, Vol. 13, No. 3, pp. 50-67, September 2022.

[2] Zuppelli, Marco. "Stego-Images-Dataset." Kaggle, 8 Aug. 2022, www.kaggle.com/datasets/marcozuppelli/stegoimagesdataset/data.

[3] Dickson, Ben. "What Is Steganography? A Complete Guide to the Ancient Art of Concealing Messages." The Daily Swig — Cybersecurity News and Views, The Daily Swig, 6 Feb. 2020, portswigger.net/daily-swig/what-is-steganography-a-complete-guide-to-the-ancient-art-of-concealing-messages.

[4] Hussein, Sahar, et al. "Usage LSB Method in Hiding Text Information within Text in an Image ." Journal of Algebraic Statistics, vol. 13, no. 2, 2022, pp. 1228–1235.

[5] Zion3R. "LSB-Steganography - Python Program to Steganography Files into Images Using the Least Significant Bit." KitPloit, 3 Feb. 2018, www.kitploit.com/2018/02/lsb-steganography-python-program-to.html?m=0.

[6] Ge, Shen, et al. "Least significant bit steganography detection with machine learning techniques." Proceedings of the 2007 International Workshop on Domain Driven Data Mining, 12 Aug. 2007, pp. 24–32, https://doi.org/10.1145/1288552.1288556

[7] "This Equation Will Change How You See the World (the Logistic Map)." YouTube, Veritasium, 29 Jan. 2020, www.youtube.com/watch?v=ovJcsL7vyrk.

[8] Wei, Ping, et al. “Generative Steganography Diffusion.” arXiv.Org, 6 Sept. 2023, arxiv.org/abs/2305.03472.

[9] Christ, Miranda, et al. ”Undetectable Watermarks for Language Models.” Eprint.Iacr.Org 26 May 2023, eprint.iacr.org/2023/763.pdf

[10] Plachta, Mikolaj, et al. ”Detection of Image Steganography Using Deep Learning and Ensemble Classifiers.” MDPI. Multidisciplinary Digital Publishing Institute, 13 May 2022, www.mdpi.com/2079-9292/11/10/1565.

[11] Dickson, Ben. “Language Models Can Use Steganography to Hide Their Reasoning, Study Finds.” VentureBeat, VentureBeat, 9 Nov. 2023, venturebeat.com/ai/language models-can-use-steganography-to-hide-their reasoning-study-finds/.

[12] Shah, Saily. “Convolutional Neural Network: An Overview.” Analytics Vidhya, 15 Mar. 2022, www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/.