Issues in Information Systems
Volume 25, Issue 3 pp. 119-132, 2024

DOI: https://doi.org/10.48009/3_iis_2024_110

A comparison of AI models to detect hidden messages in images

George Stefanek, *Purdue University Northwest, stefanek@purdue.edu*
Leif Gulbransen, *Purdue University Northwest, lgulbran@pnw.edu*
Griffin Spink, *Purdue University Northwest, gspink@pnw.edu*
Jack Morawski, *Purdue University Northwest, jmorawsk@pnw.edu*
Dylan Fila, *Purdue University Northwest, dfilla@pnw.edu*
Ronald C. Rabello De Castro, *Purdue University Northwest, rdecastr@purdue.edu*

Abstract

Steganography, the art of concealing secret information within seemingly innocuous cover media, poses significant challenges for digital security and forensic analysis. With the increasing use of digital images as carriers for hidden data, the need for efficient and accurate steganalysis techniques becomes imperative. This research compared several machine learning models including K-Nearest Neighbor, Gaussian, Multi-Layer Perceptron, Stochastic, Random Forest, aa non-pre-trained Convolutional Neural Network, and a pre-trained Convolutional Neural Network model called ResNet-18 in their effectiveness at detecting images that have a steganographic message embedded in it. The study found that the convolutional neural network was the best model in detecting steganographic content with 99% accuracy.

Keywords: machine learning, steganography, steganalysis, CNN, AI, MLP, KNN

Introduction

Steganography is the art of concealing secret messages within various media and poses significant challenges for digital security and forensic analysis. Steganography conceals a message's existence in media content such as image**,** audio, video, and text by methods such as invisible inks, microdots, character arrangement, digital signatures, covert channels, and spread spectrum communications (Johnson & Jajodia, 1998). It can be used as an additional layer of security in transmitting sensitive information but can also be used by cyber criminals to conduct secret and malicious communication. Steganography has also been found useful in hiding code inside multimedia objects, mostly images with the goal of infiltrating malware (stegomalware) into organizations or personal devices (Pérez, Rosales & Cruz-Cortés, 2016). There is an increasing use of digital images as carriers for hidden data, thereby increasing the need for efficient and accurate steganalysis techniques. This research compares the effectiveness of various AI machine learning models in performing image steganalysis to detect hidden messages in images.

Background

This research builds on previous research at the institution where a steganographic message was added using StegHide software to a set of 400 images that did not have a uniform size. A Convolutional Neural Network (CNN) from the FastAI library was used to try to detect the hidden messages with a result of roughly 50% accuracy. This paper presents research that was focused on trying various AI machine learning

models on a larger dataset of images to see which models would perform best at detecting images embedded with steganographic messages (i.e., messages containing a hidden message). Additionally, the research focused on examining the previous work on how the images were formatted, sized and labeled to see if these factors influenced the poor accuracy of detection and determining any possible improvements.

Initially a review was done of which AI models might be most effective at processing a dataset of images with steganographic content. The first model that was looked at was a transformer because of the recent popularity of generative AI using Generative Pre-Trained Transformer (GPT) models for chat applications. According to Rick Merrit at NVIDIA, "Any application using sequential text, image or video data is a candidate for transformer models." (Merritt, 2022). Since the focus was only images, this was a possible candidate, but was ruled out because of the way transformers process the input data. Transformers are designed to process sequential data such as natural language text. Since messages were being embedded into images, the Convolutional Neural Network used in the previous study was revisited. Transformers were ruled out since CNNs are designed to process data that has a grid-like structure, such as images. CNNs do this by applying a series of convolutional filters to the input data, which are designed to detect specific features in the image. The filters are typically small and move across the image in a sliding window style, creating a feature map that encodes the presence of the features detected by the filter [Pratap, 2023].

The previous project that looked at detecting images with steganographic messages had a poor result using CNN. We looked back at what was done in the previous project to try to determine why CNN performed poorly. There were a few possible reasons for the previous poor performance which included the use of a small dataset of 800 images used for training (400) and testing (400), a mix of images with different sizes and possible labeling issues. Additionally, models that were able to do something like CNNs such as K Nearest Neighbor (KNN) were looked at and decided as a candidate for this research project. A KNN was available using the scikit-learn library (Pedregosa, 2011) as were other models that we decided to try such as the multilayer perceptron. The multilayer perceptron is an artificial neural network that organizes the data in at least three layers. The multilayer perceptron has an activation function which is used so that the model can learn. This was thought to be a good candidate model where it could be trained to learn the difference between a photo without steganography and a photo with steganography.

A significantly larger library of images needed to be identified that could be used to provide a large training set for the models. The CIFAR10 dataset was found to be a good source for images. Using this library a dataset of 120,000 photos was built to train and test the models to be used in this study.

Literature Review

A variety of papers discuss the use of neural networks for performing image steganography such as Georges and Magdi (Georges, Magdi, 2020), Shelke, Dongre and Soni (Shelke, Dongre & Soni, 2014), and Marwaha and Marwaha (Marwaha, & Marwaha, 2010). The following papers focus on performing steganalysis of images using neural networks which was also the focus of this research.

Bhattacharyya (Bhattacharyya, 2011) presents a survey of steganography and steganalysis techniques. Specifically in the area of image steganalysis techniques include: 1) video steganalysis using the temporal correlation between frames, 2) video steganalysis based on asymptotic relative efficiency (ARE), 3) video steganalysis based on spatial and temporal prediction, and 4) steganalysis methods using neural networks and support vector machines to detect hidden information by exploring spatial and temporal redundancies.

Jarusek, Volna and Kotyrba (Jarusek, Volna & Kotyrba, 2019) used a steganographic method StegoNN based on neural networks. If an image is signed by this technique, then the detection of any modifications does not need any external data, such as a database of non-modified originals. The quality of the signature

in various parts of the image serves to identify modified parts of the image. The study was performed on photographs from the CoMoFoD dataset, and its results were compared with other approaches that used the database based on standard metrics. The study showed the ability of the StegoNN method to detect corrupted parts of an image and to mark where most probably the image was manipulated.

Berg et al. (Berg et al., 2003) did some early work on the feasibility of using machine learning to detect stegged images in both content-based (GIF) and compression-based (JPEG) image formats. They used an error back-propagation neural network to distinguish clean and stego-bearing files. The dataset consisted of 150 images consisting of 50 each of flowers, mountains and trees. The results showed accuracy of 51% - 81% on JPEGs depending varying in accuracy depending on the image in the file. They found an 85.5% accuracy in stegged GIFs. The results reported here show that ML algorithms can work in both content- and compression-based image formats.

Pérez, Rosales, and Cruz-Cortés (Pérez, Rosales & Cruz-Cortés, 2016) present a steganalysis method based on an Artificial Immune System (AIS), to detect JPEG images modified with three steganographic tools: F5, Outguess, and Steghide. Their research used Haar Wavelets to extract a feature vector that best described the analyzed image. For characterization purposes a small set of features are extracted to describe the image using the two-dimensional Haar Wavelet Transform. After the transformation, the obtained matrix has coefficients 1) containing average changes (AC) that include information about the global properties of an image, 2) horizontal changes (HC) that include information about the horizontal lines hidden in an image, 3) vertical changes (VC) that contain information about the vertical lines hidden in an image, and 4) diagonal changes (DC) containing information about the diagonal details hidden in an image. The horizontal coefficients HC reached the best steganograph detection rate of 94.33%. In contrast the coefficients VC and DC reached a detection rate of 85.71% on average.

Mohamed, et al. (Mohamed, Rabie, Kamel & Alnajjar, 2021) used a transform domain-based steganalysis scheme that utilized the architecture of AlexNet is an eight-layer deep convolutional neural network that was designed for object classification. The network was trained using over one million images from the ImageNet database. Some modifications were performed on the existing AlexNet architecture to enhance the detection performance of the model on stego images. Experiments showed that the FB-GAR steganography scheme was successfully detected with an accuracy of 74.72% using AlexNet. Other older papers by Zhi and Fen (Zhi & Fen, 2004) and Kerr (Kerr, 2004) address steganalysis, but do not use neural network techniques.

Even though there has been use of neural network techniques for the application of steganography in images there has been limited content on using the latest neural network techniques for the detection of messages in images. This research focuses on looking comparing both conventional non-AI and AI (neural network) techniques for performing steganalysis in images.

Statement of the Problem

Steganography has been used to hide code inside multimedia objects, mostly images with the goal of infiltrating malware into organizations or personal devices (Pérez, Rosales & Cruz-Cortés, 2016). It has also been used by cyber criminals to conduct secret and malicious communication. This increasing use of digital images as carriers for hidden data requires efficient and accurate steganalysis techniques to detect the hidden content. There is limited work in using neural network techniques to improve detection. This research compares the effectiveness of various classification methods and neural network models to perform image steganalysis to detect hidden messages in images.

Research Question and Scope of Study

The research question for this study was "Can a current machine learning model accurately detect steganographic content in images?". The scope of the study was 1) to select seven to eight machine learning models, 2) select a large dataset of images that could be used in training and testing the models, 3) embed steganographic content into a portion of the images, 4) train the models to attempt to predict whether an image has steganographic content or not, and 5) test the models using a large dataset of images. The end goal was to try to find methods and a neural network that can perform hidden message detection in images with high accuracy.

Methodology

To get a larger set of images to train and test with the machine learning models that were to be evaluated in this study, the CIFAR10 dataset was downloaded from the TensorFlow catalog. The dataset consisted of 60,000 color 32 x 32 images in 10 classes representing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The images in the CIFAR10 dataset were resized to 100 x 100 pixels. The reason the images were resized was so that steganography could be added since the photo needs to be large enough to be able to add the actual text message into it. All the images in the dataset that were used in the study were the same size after processing. There was no additional data cleaning or preparation done on the images to extract any additional features. The only image preparation was resizing the images for a fixed and identical image size. See Figure 1 for the script to resize the images.

```python
from PIL import Image
import numpy as np
import os

def resize_images(input_dir, output_dir, size):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(input_dir):
        if filename.endswith('.jpg'):
            image_path = os.path.join(input_dir, filename)
            output_path = os.path.join(output_dir, filename)
            image = Image.open(image_path)
            resized_image = image.resize(size, Image.ANTIALIAS)
            resized_image.save(output_path)

# Example usage
input_dir = 'photos'
output_dir = 'resized_cifar10_images'
target_size = (100, 100)  # Desired size for resizing

resize_images(input_dir, output_dir, target_size)
```

Figure 1: Code for resizing the CIFAR10 dataset.

Steganography

The next step was to add a steganographic message to the resized 100 x 100 images. A script was created to add a message (message was in a file) into half the images in the dataset using Steghide software. See

Figure 2 for the bash script used in adding the steganographic message to the images. The same message was added to all the images that were getting steganographic content.

```
  GNU nano 4.8                                                    steghide_script.sh
#!/bin/bash

# Specify the directory containing the image files
IMAGES_DIR="/home/seed/Desktop/Senior/resized_cifar10_images"

# Specify the directory to save the stegged files
OUTPUT_DIR="/home/seed/Desktop/Senior/steg_512Photos"

# Specify the directory to save the cover images
COVER_IMAGES_DIR="/home/seed/Desktop/Senior/cover_images"

# Initialize counter
count=0

# Change to the images directory
cd "$IMAGES_DIR" || exit

# Create the output directory if it doesn't exist
mkdir -p "$OUTPUT_DIR"
mkdir -p "$COVER_IMAGES_DIR"

# Iterate over each image file in the directory
for image_file in *.{jpg,jpeg,png}; do
    # Check if the file is a regular file
    if [ -f "$image_file" ]; then
        echo "Processing $image_file ..."

        # Hide data in the image using Steghide
        steghide embed -ef /home/seed/secret/secret.txt -cf "$image_file" -sf "$OUTPUT_DIR/stegged_$image_file" -p "Senior"

        # Check if embedding was successful (exit code 0)
        if [ $? -eq 0 ]; then
            # Increment the counter
            ((count++))

            # Copy the cover image to the cover images directory
            cp "$image_file" "$COVER_IMAGES_DIR/$image_file"
        fi
    fi
done

echo "Total images stegged: $count"
echo "All images processed."
```

Figure 2: Script used in adding steganographic message to images.

Setting up a Remote Jupyter Notebook

To run the models, the computer to be used had the following specification:

- GPU: NVIDIA 4090
- CPU: Intel I9 14900k
- Memory: 32gb DDR5

To utilize the GPU to its fullest potential, a remote Jupyter notebook was set up so that all the members of the research team could access it. Tailscale, a VPN service, was used to allow remote access to the computer desktop using SSH. The Windows "task scheduler" was used to start the Jupyter notebook on startup of the computer and run it in the background. Computers with Tailscale client software were able to SSH into the computer and run the Jupyter notebook.

Machine Learning Model Selection

An evaluation was performed to determine which AI machine learning models were to be selected for use in this study. The models that were selected from the scikit-learn library were K-nearest neighbor (KNN), Gaussian, Multilayer perceptron (MLP), Stochastic, Random Forest, and a non-pre-trained Convolutional Neural Network (CNN). Additionally, a pre-trained CNN called ResNet-18 was used which was downloaded from a professor's Github account.

Additionally, other libraries were needed to work with the AI models. These included PyTorch, TorchVision and Cuda. Cuda is a python library from NVIDIA that allows a GPU to be used when running the models to speed up processing. PyTorch is a framework for building and deploying deep learning models that use Cuda. TorchVision is a library with datasets, model architectures, and image transformations for computer vision that provides capabilities such as semantic segmentation which classifies specific pixels into categories.

Training and Testing the Models

During the implementation of the different models, there were a few problems that needed to be resolved. There was an issue regarding labeling of the images where an improper labeling system resulted in training only being able to read about half the images. Initially, all the images were in one folder, with the labeling coming from the filename of the image. For example, an image file of a truck with a hidden message in it would be named "stegged_truck1234", while a non-stegged image would be named "truck1234". This initial organization of image files and the file labeling did not properly work with the models, leading to improper training which led to poor accuracy. This problem was resolved by putting the images with steganographic message content in a separate folder and non-stegged images in a separate folder. This allowed for proper training and made a significant difference in terms of accuracy, F-1 measure, precision, and recall.

Another problem encountered was memory issues when loading the dataset into the GPU. When implementing the models, the study began with 120,000 images from the CIFAR-10 dataset; 60,000 normal images and 60,000 images with steganographic content (all of the images were resized to 100x100). Loading that amount of data was a problem due to the large amount of video memory needed. The 120,000 images required 37-40GB of video memory to load into the Jupyter notebook causing an error when running. The NVIDIA 4090 GPU that was used only had 24 GB of video memory so a dataset size needed to be found that the GPU could handle. The dataset size that was able to work with all of the models consisted of 20,000 images from the CIFAR-10 dataset: There were 10,000 images with steganographic content and 10,000 images without steganographic content. All the images were sized at 100 x 100 resolution. Keeping the dataset uniform at a 100 x 100 image size was important when running the models. Resizing images with steganographic content would degrade the integrity of the steganography itself.

In preparing the datasets for training and testing, the dataset was randomly split into 20% for training and 80% for testing. See Figure 3 for the code to split the dataset into training and test sets. Since the d

```
images = DataBlock(
    blocks = (ImageBlock, CategoryBlock),
    get_items = get_image_files,
    splitter = RandomSplitter(valid_pct=0.2, seed=42),
    get_y = parent_label
)
```

Figure 3: Code for randomly splitting the dataset into training and test datasets.

dataset consisted of 20,000 images, splitting the data at 20/80 created a training dataset with 4,000 images and testing dataset with 16,000 images. The epoch size, learning rate, and the batch size were also set. The model libraries would be imported into the Python program, and the 20,000 images loaded into the Jupyter Notebook. Testing would run through the images to predict if the image had steganography content or not. It would compare the predicted image with what the image was and calculate the accuracy, F-1 measure, precision, recall, error rate, training loss and valid loss.

Results

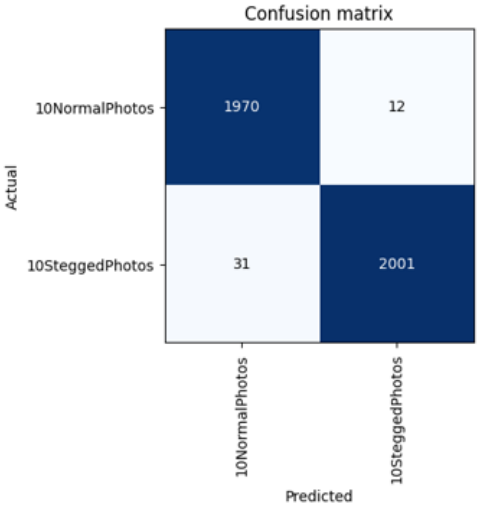
ResNet 18 and ResNet 101: ResNet-18 is a Convolutional Neural Network (CNN) pretrained with millions of images and compares them to the dataset that is loaded into the model. ResNet also uses FastAI which is a library built on top of the PyTorch library. FastAI is an open-source library that helps with deep learning. The "18" in ResNet-18 represents the number of layers in the neural network. Table 1 below shows 10 different runs with different variables. The variables include the number of Total Epochs, Total Layers, Batch Size, Photo Size and Learning Rate. The comparison statistics are F1 Measure, Accuracy, Precision, Time to Finish(minutes) and Recall.

Table 1: ResNet 18 and 101 results.

Amount of Pictures	Epochs	Time to Finish	Layer Amount	Batch Size	Learning Rate	F1 Measure	Accuracy	Precision	Recall	Photo Size
~20000	4	10	18	10	0.1	0.97	0.97	0.97	0.97	100x100
~20000	10	23	18	10	0.1	0.98	0.98	0.98	0.98	100x100
~20000	10	108	101	10	0.1	0.98	0.98	0.98	0.98	100x100
~20000	4	48	101	10	0.1	0.97	0.97	0.97	0.97	100x100
~20000	20	45	18	10	0.1	0.99	0.99	0.99	0.99	100x100
~60000	4	70	18	10	0.1	0.5	0.5	0.5	0.5	512x512 (resized to 100x100)
~20000	30	116	18	10	0.1	0.99	0.99	0.99	0.99	100x100
~20000	4	15	18	20	0.1	0.97	0.97	0.97	0.97	100x100
~20000	4	15	18	32	0.1	0.98	0.98	0.98	0.98	100x100
~20000	4	15	18	64	0.1	0.97	0.97	0.97	0.97	100x100

The best results using the pre-trained ResNet was a 0.99 F-1 Measure, or 99% detection rate. The variables were set to 20 epochs, 18 layers, batch size of 10, learning rate of 0.1, and the image size was sized at 100x100 for all images. That run took longer than most ResNet-18 runs because of the epoch size being greater. Table 2 shows the confusion matrix for this run. The top left and bottom right represent the images that were identified correctly by the model, while the bottom left and top right represent the images that were not correctly identified by the model. This model produced very good results for finding the hidden message within the images with 99% of the training dataset images being correctly identified.

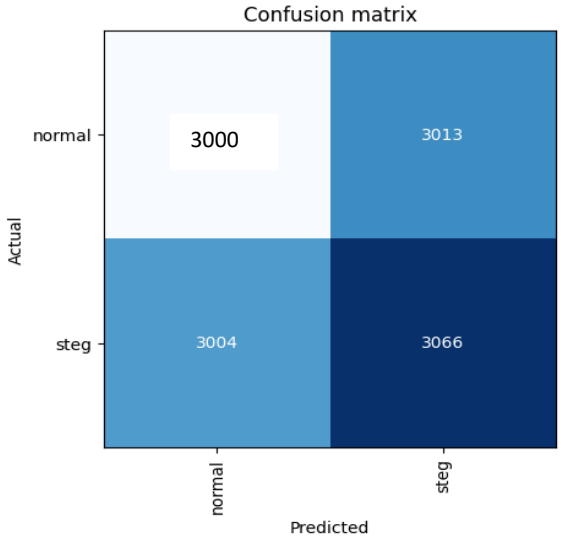
Table 2: Confusion matrix for the best ResNet run.



The worst result using the ResNet models was for 60,000 512 x 512 images with 4 epochs, 18 layers, batch size of 10, and a 0.1 learning rate. It had a 0.5 F-1 measure and accuracy due to the image size - essentially a coin flip. Image size played a significant role in the result. From these tests, it was determined that resizing was impossible in terms of detecting steganography. Table 3 shows the confusion matrix for this run. Because the dataset had roughly 60,000 images, the training set had roughly 12,000 images. Out of the 6,000 images with steganographic content, only about 3,066 were detected. While 3,013 images were not identified correctly.

Other observations that were made for ResNet-101 were that it takes significantly longer to run in comparison to ResNet-18. Both were given 10 epochs, 10 batch size, learning rate of 0.1 and the same dataset. Both models performed similarly to each other, around 0.97-0.99 F1 Measure respectively, but ResNet-18 would run faster by 85 minutes when given the same parameters. Another observation that was found was that batch size did not affect time-to-finish. ResNet-18 was given identical parameters besides

Table 3: Confusion matrix for the worst ResNet run.



batch size and found that the time-to-finish did not change at 15 minutes. The 3 batch sizes that were compared were 20, 32, and 64. A batch size of 32 did yield a better F-1 Measure than the other 2 tested with a score of 0.98 (batch sizes of 64 and 20 had an F-1 Measure of 0.97).

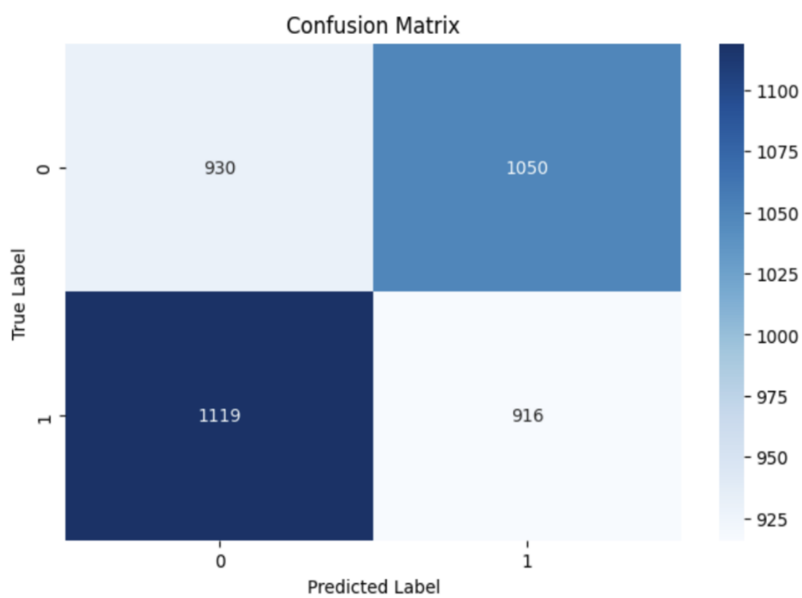
Gaussian Model: The Gaussian model is a "non-parametric, supervised learning method used to solve regression and probabilistic classification problems" [Supervised learning. (n.d.-b). Scikit-learn] and is part of the scikit-learn library. The Gaussian model was trained on 20,000 images (same as ResNet). Unlike ResNet, there were no variables that could be changed to determine if there would be an impact on detecting the hidden messages in the images. Table 4 shows the results from the Gaussian model. All 10 runs ended with 0.46 for F-1 Measure, Accuracy, Precision, and Recall.

Table 4: Gaussian model results.

Amount of Pictures	Time to Finish	Layer Amount	Batch Size	Learning Rate	F1 Measure	Accuracy	Precision	Recall	Photo Size
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100
~20000	1 min	N/A	N/A	N/A	0.46	0.46	0.46	0.46	100x100

Table 5 shows the confusion matrix from the results of the Gaussian model. The labels for this confusion matrix are slightly different in comparison to the others but have the same concept. The number 0 on the top left y-axis represents predicted images with steganographic content that were actually images with

Table 5: Confusion matrix for Gaussian model.



steganographic content. The number 1 on the bottom left y-axis represents the images that were predicted incorrectly. The top and bottom on the right represent images that were predicted incorrectly vs correctly. This model had little success in identifying steganographic content. There is a less than 50% detection rate. The majority of images in the dataset were not correctly identified, out of roughly 2,000 images that had steganographic content, only 930 images were detected correctly.

Convolutional Neural Network (CNN): The key difference between the ResNet model and this CNN is that the CNN had to be trained - it was not pre-trained with any images. Table 6 shows the results from the

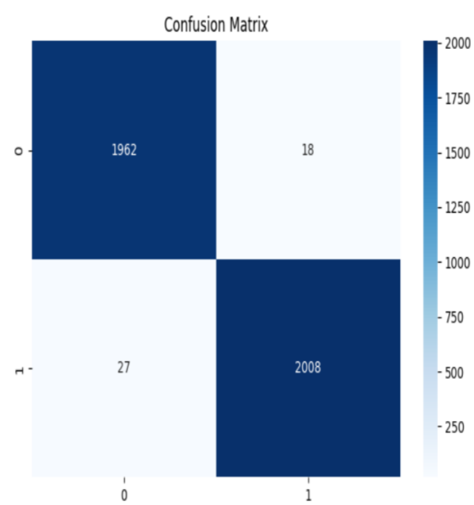
Table 6: Convolutional Neural Network results.

Amount of Pictures	Epochs	Activation Functions	Time to Finish (MIN)	Layer Amount	Batch Size	Learning Rate	F1 Measure	Accuracy	Precision	Recall	Photo Size
~20000	100	Functional ReLU	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional ReLU	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional ReLU	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional ReLU	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional ReLU	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional Leaky Relu	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional Leaky Relu	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional Leaky Relu	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100
~20000	100	Functional Leaky Relu	10	3	32	0.0001	0.98	0.98	0.98	0.98	100x100
~20000	100	Functional Leaky Relu	10	3	32	0.0001	0.99	0.99	0.99	0.99	100x100

CNN model. Out of the 10 runs, half use the activation functions of ReLU, while the other half use Functional Leaky ReLU. The CNN was very effective at identifying the dataset images properly with a 0.99 F1-Measure, Accuracy, Precision and Recall result. One run had a result of 0.98 for Functional Leaky ReLU.

Table 7 shows the confusion matrix for the CNN. It shows very good results in identifying the hidden messages with steganographic content. The top left and bottom right represent the images that were

Table 7: Convolutional neural network not pre-trained



identified correctly by the model, while the bottom left and top right represent the images that were not correctly identified by the model. Images without steganographic content were correctly identified 1,962

times and incorrectly identified only 27 times. The images that contain steganography had even more success with 2,008 images being correctly identified, and only 18 incorrectly identified.

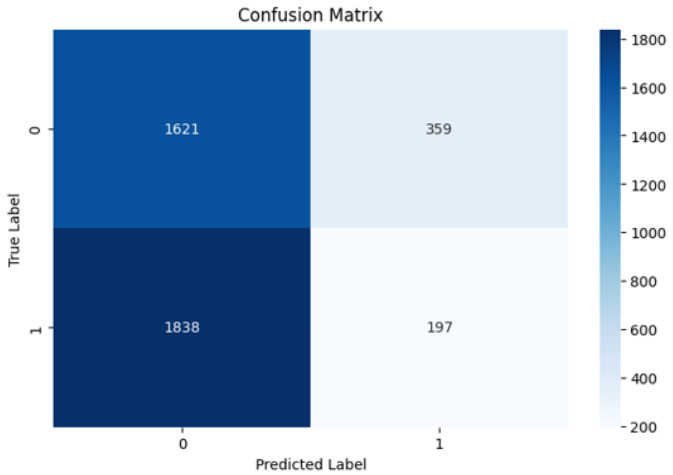
Stochastic Gradient Descent (SGD): The Stochastic Gradient Descent model finds the best fit using the parameters provided. The model runs efficiently with an average processing time of around 4 minutes. The results from 10 runs using the SGD model are shown in Table 8. Using the same dataset as the other models, except for a few different variables, the SGD model did not perform well averaging about a 0.4 F-1 measure (40% of images with steganography). Adjusting the epochs did not change the results very much, and increasing the epochs would worsen the results due to over training of the dataset. The best results were found using 75 epochs that produced an F-1 measure of 0.42, accuracy of 0.43, precision of 0.42 and recall of 0.43.

Table 8: Stochastic Gradient Descent results.

Amount of Pictures	Time to Finish	Epoch	F1 Measure	Accuracy	Precision	Recall	Photo Size
~20000	3	100	0.4	0.45	0.42	0.45	100x100
~20000	3	25	0.37	0.48	0.43	0.48	100x100
~20000	3	50	0.33	0.48	0.4	0.48	100x100
~20000	3	75	0.42	0.43	0.42	0.43	100x100
~20000	4	125	0.38	0.44	0.4	0.44	100x100
~20000	4	150	0.36	0.47	0.41	0.47	100x100
~20000	4	175	0.42	0.43	0.42	0.43	100x100
~20000	5	200	0.34	0.45	0.37	0.45	100x100
~20000	7	225	0.37	0.47	0.41	0.47	100x100
~20000	8	250	0.41	0.44	0.42	0.44	100x100

Table 9 shows the confusion matrix for SGD. It had pretty good success at predicting images without steganographic content, correctly identifying 1,621 images correctly and 359 incorrectly. However, there were a lot of false positives with the steganographic images. 1,838 images were incorrectly labeled as images without steganographic content while only 197 were correctly predicted as images with steganographic content.

Table 9: Stochastic Gradient Descent confusion matrix.



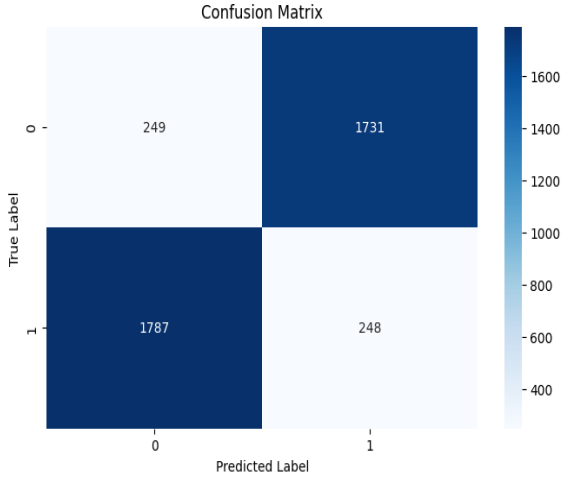
Random Forest Classifier Model: The Random Forest Classifier Model operates by creating multiple decision trees during training and outputs the mode. The randomness in the algorithm comes from two sources: random sampling of the training data points when building each tree, and random subsets of features considered when splitting nodes. This randomness helps to reduce overfitting and improves the generalization ability of the model. The results of the Random Forest Classifier model are shown in Table 10. Overall, it did not perform well when trying to predict the images in our dataset. The same datasets were used as in previous models. In the test runs, max depth was adjusted to see if it would make a difference. Max depth refers to the maximum depth allowed for each individual decision tree in the forest, with a higher number indicating more depth. Adding more depth produced worse results. The highest F-1 measure was associated with a max depth of 1.0. The depth was increased one by one, to determine if the model performance decreased. The worst result resulted from a max depth of 10.

Table 10: Random Forest model results.

Amount of Pictures	Time to Finish	Max Depth	F1 Measure	Accuracy	Precision	Recall	Photo Size
~20000	3	1	0.42	0.45	0.43	0.45	100x100
~20000	3	2	0.4	0.4	0.4	0.4	100x100
~20000	3	3	0.37	0.37	0.37	0.37	100x100
~20000	3	4	0.31	0.31	0.31	0.31	100x100
~20000	3	5	0.27	0.27	0.27	0.27	100x100
~20000	3	6	0.22	0.22	0.22	0.22	100x100
~20000	3	7	0.19	0.19	0.19	0.19	100x100
~20000	3	8	0.16	0.16	0.16	0.16	100x100
~20000	3	9	0.14	0.14	0.14	0.14	100x100
~20000	5	10	0.12	0.12	0.12	0.12	100x100

The confusion matrix for the Random Forest model is shown in Table 11. The results are completely inverted where most of the images are incorrectly identified with 1,731 of the total 1,980 images without

Table 11: Confusion matrix for Random Forest when the depth is



steganography being labelled as images with steganographic content. While only 249 were correctly identified as images without steganographic content. Random Forest classification struggled with both images with steganographic content and images without steganographic content when increasing the depth.

Discussion

The results showed that the *ResNet 18 / 101* Convolutional Neural Networks and the non-pretrained Convolutional Neural Network (CNN) models were best at determining whether an image had steganographic content (a hidden message). The F-1 measures for the CNNs beat anything else we tried with the other models. It did not matter whether the CNN was pre-trained or not. Both models yielded very good results regardless of the comparison statistics. The confusion matrices for the CNN models showed their performance at differentiating images. About 99% of images in the dataset were properly identified. The dataset was a key factor in the success of the model. As long as the images were uniform in size, the model worked very well. Most other models that were tried had an F-1 measure around 0.5 F-1. The worst model was the Random Forest Classifier. As depth was increased to the forest, it progressively got worse to where it was only accurately predicting the images at 12% accuracy. The poor performance of the other models (non-CNN models) may be attributed to the models not being built for image classification and that the models may not have been properly trained on the dataset.

Conclusion

This research project has concluded that it is possible to accurately detect steganographic content in images by the use of Convolutional Neural Networks. Proper labeling of the image files and organizing the files in separate directories played a significant role in creating a good training dataset. Also formatting the images to have a uniform image size had a significant role in improving the accuracy of the results. Future work can explore if there are ways to have multiple image sizes within a dataset and still get an accurate result in detecting steganographic content. Additionally, including multiple different types of hidden messages in the image dataset would make it more broadly applicable across real-world scenarios. Even though this research used a GPU, using cloud resources from Amazon, Microsoft, or Google to build a machine learning architecture for this type of application would be beneficial. Having more computing power available would help get through runs faster, which would enable more tests to be run over time.

References

Merritt, R. (2022, September 16). *What is a transformer model?*. NVIDIA Blog. https://blogs.nvidia.com/blog/what-is-a-transformer-model/

Abiodun, Oludare, Jantan, A., Omolara, O., Dada, K., Umar, A., Linus, O., Arshad, H., Aminu Kazaure, A., Gana, U., & Kiru, M. (2019). Comprehensive review of artificial neural network applications to pattern recognition. IEEE Access, 1-1. https://doi.org/10.1109/ACCESS.2019.2945545

Bhattacharyya, S. (2011). A survey of steganography and steganalysis technique in image, text, audio and video as cover carrier. *Journal of global research in computer science*, *2*(4).

Berg, G., Davidson, I., Duan, M. Y., & Paul, G. (2003, August). Searching for Hidden Messages: Automatic Detection of Steganography. In *IAAI* (pp. 51-56).

Calix, R. (2024, March 6). *Transferlearning/FASTAI/imgclassification/classifycifar10withfastai.ipynb at main · RCALIX1/transferlearning*. GitHub. https://github.com/rcalix1/TransferLearning/blob/main/fastai/ImgClassification/ClassifyCIFAR10WithFastAI.ipynb

Facebook/Detr-resnet-50 · hugging face. facebook/detr-resnet-50 · Hugging Face. (n.d.). https://huggingface.co/facebook/detr-resnet-50

Georges, J., & Magdi, D. A. (2020). Using artificial intelligence approaches for image steganography: A review. *Internet of Things—Applications and Future: Proceedings of ITAF 2019*, 239-247.

Jarusek, R., Volna, E., & Kotyrba, M. (2019). Photomontage detection using steganography technique based on a neural network. *Neural Networks*, *116*, 150-165.

Johnson, N. F., & Jajodia, S. (1998). Exploring steganography: Seeing the unseen. *Computer*, *31*(2), 26-34.

Ker, A. D. (2004, May). Improved detection of LSB steganography in grayscale images. In *International workshop on information hiding* (pp. 97-115). Berlin, Heidelberg: Springer Berlin Heidelberg.

Maboudi, M. (2019). Writing Cifar10 dataset to image files as ".tif" or ".jpg" . GitHub. https://github.com/MBMS80/Writing-Cifar10-dataset-to-image-files-as-.tif-or-.jpg-

Marwaha, P., & Marwaha, P. (2010, July). Visual cryptographic steganography in images. In *2010 Second international conference on computing, communication and networking technologies* (pp. 1-6). IEEE.

Mohamed, N., Rabie, T., Kamel, I., & Alnajjar, K. (2021, April). Detecting Secret Messages in Images Using Neural Networks. In *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)* (pp. 1-6). IEEE.

Pedregosa, F. (2011). Scikit-learn: Machine learning in python Fabian. Journal of machine learning research, 12, 2825.

Pérez, J. D. J. S., Rosales, M. S., & Cruz-Cortés, N. (2016, August). Universal steganography detector based on an artificial immune system for JPEG images. In *2016 IEEE Trustcom/BigDataSE/ISPA* (pp. 1896-1903). IEEE.

Pratap, A. (2023, January). Comparing CNNs and Transformers: Understanding the Differences and Key Components of These Popular Deep Learning Architectures. Medium. https://medium.com/deep-learners-in-deep-learning-and-machine/comparing-cnns-and-transformers-understanding-the-differences-and-key-components-of-these-popular-4cecdec2d0d9

Shelke, F. M., Dongre, A. A., & Soni, P. D. (2014). Comparison of different techniques for Steganography in images. *International Journal of Application or Innovation in Engineering & Management*, *3*(2), 171-176.

Supervised learning. (n.d.-b). Scikit-learn. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Zhi, L., & Fen, S. A. (2004, September). Detection of random LSB image steganography. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004* (Vol. 3, pp. 2113-2117). IEEE.