StegNet CovertCast

Derek Zhang
Department of Computer Science
University of Maryland
College Park
dzhang21@umd.edu

Siyuan Peng
Department of Computer Science
University of Maryland
College Park
peng2000@umd.edu

Abstract

Steganography is hiding a secret message within another message. With the recent rise of popular streaming services, we intend to hitch a ride on them to send obfuscated messages over the internet. In particular, we use deep learning steganography to achieve this. In this paper, we investigate the use of correlation loss on a deep neural network to encode a message to embedded images and apply this pipeline to a real-world scenario. We test our neural network with a wide range of inputs: images, texts, and HTML files and analyzed the performance of the model with different settings. We found that while prior works are mostly replicable, they only hold up in the natural image domain. It is quite non-trivial to use the CovertCast method and be able to recover the binary after quantization.

1 Introduction

Recently, the fast and vigorous development of the internet has facilitated the rise of the live stream industry. Millions of people watch live streams on popular platforms like Google Live and Twitch. Some existing techniques of circumventing internet censorship are not enough such as those by McPherson u. a. (2016) who created a method to transmit data through encrypted live-stream videos using a model called CovertCast. The key idea is to evade censorship as long as any (foreign) video streaming service is permitted in such a country. CovertCast videos will "hide" among the myriad of legitimate live streams. Most streaming services will use encryption. In particular, YouTube's streaming protocol seems to use fixed-sized packets. Thus it isn't easy to detect the use of CovertCast. However, this paper does not consider the possibility of social engineering or an arrangement between the streaming service and the censoring country to block CovertCast. To a human, it is very obvious (see appendix) that CovertCast is being used. Thus this puts users in danger if their country's intelligence agency can correlate them to have watched a CovertCast live stream. Another possibility is collusion or a complete buyout of the streaming service by the censoring country. The streaming service might agree to block CovertCast or noise-like videos in exchange for business in the censoring country. Our work acts as a first step towards making it impossible to detect if a video contains a secret message. Such technology will have the effect of forcing every country to either allow any data or block videos entirely. The main contribution is to a) analyze the use of correlation loss (section 3.4), b) investigate the use of noise on Stegnet Wu u. a. (2018), and c) apply Stegnet steganography to ConvertCast. We hope in the future that once this field has matured, that we will be able to send realistic data over live streaming platforms. We've published our source code here: github.com/chromestone/Stegnet-Covertcast

2 Related Work

The fast development of the internet and multimedia has greatly facilitated video steganography. Besides the traditional information embedding method, deep learning also plays a significant part

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

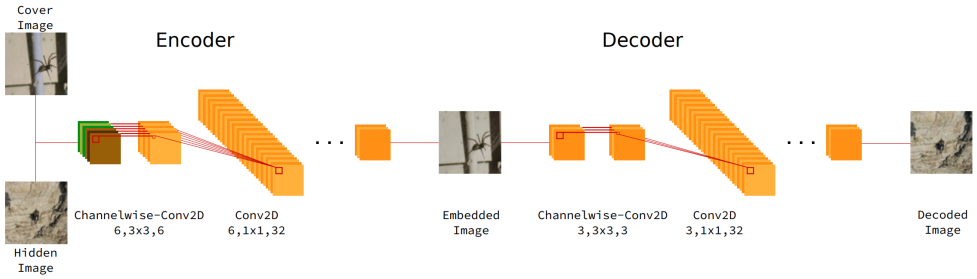

Figure 1: StegNet Architecture Overview

in this process. In this section, we will explore the three types of video steganography. There are three types of video steganography, based on how the information is embedded Liu u. a. (2019): pre-embedding, intra-embedding, and post-embedding.

Pre-embedding, as suggested by the name, happens when the video is still in raw format and then the sender compresses the video along with the hidden message. The characteristic of this method is that it has high capacity and it's simple but hard to prevent steganalysis. To add an extra layer of security, Cheddad u. a. (2008) includes the RSA algorithm in the pipeline. Kanade-Lucas-Tomasi, a widely-known technique for object tracking, is been used to encode secret messages Mstafa und Elleithy (2016). This method exhibits a good capacity to encode but it's very complex. In recent years, mature AI models are also used in downstream steganography tasks. Mstafa und Elleithy (2015) identify region of interest (ROI) using a face detection and tracking model and then perform embedding in that specific area, increasing the security of the message. Generally speaking, pre-embedding methods do not depend on the video coding processing or the encoding method. With modern techniques, more information can hide in the video more securely. However, the cost of this method is that it's time-consuming and may suffer information loss.

The intra-embedding type describes the embedding that occurs in the compressed domain. A major advantage is that videos generally are stored as compressed versions. However, with modern video coding techniques, such as H.26X, the redundant part of the raw video has been greatly compressed. Recent publications He und Luo (2008); Fang und Chang (2006) focus on using motion analysis, specifically motion vectors, to encode part of the video. By manipulating the motion vector of the video or the search process, secret messages can hide in the video.

Last but not least, post-embedding video steganography encodes messages directly into the bitstream of a video. It's not computationally achievable to encode the whole bitstream so most of the post-embedding methods only take a fraction of a video. A downside of this method is that it depends on the video compression method. Shahid u. a. (2011) uses Context-adaptive variable length coding (CAVLC). Xu und Wang (2015) designs the pipeline based on context-adaptive binary arithmetic coding (CABAC).

As far as deep learning steganography goes, there has been work in both hiding bits directly in images using CNNs Tang u. a. (2019) and works on hiding images in images using CNNs Baluja (2017); Wu u. a. (2018). In the works by Tang u. a., they look into hiding bits in images by dividing the image into common sections and adjustable sections. We found this paper hard to understand for someone not in the steganography field. Hence, we pivoted to Baluja and Wu u. a.'s work which seemed for intuitive. Baluja proposed a three model network: one to "prepare" the secret image, one to encode the prepared image into a cover image, and, finally, a decoder. However, Wu u. a. showed that a two model network composed of an encoder and decoder is possible. Hence, in the interest of time and practicality, we chose Wu u. a.'s architecture. This had the added benefit that they had not analyzed their model on some settings that Baluja did.

3 Method

3.1 Model Architecture

Following Wu u. a. (2018)'s design, we modify the first part of the network. However, there was a discrepancy between their paper and the code. In their paper, they included batch normalization and the ELU activation at the ends of the encoder and decoder model. However, we found that this was not the case in their code. We believe that using the raw outputs of the CNN makes more sense as the outputs need to be negative to match the normalized ground truth (cover and secret) images. Thus, we proceeded to terminate our encoder and decoder at the final convolutional layer.

A second discrepancy was that skip connection were implied by the "Separable Convolution with Residual Block" figure in their paper to be applied after the ELU activation. In their code, the skip (addition operator) was before the ELU activation and the tensor added also came from the pre-activation of the previous layer. Since the skip connections did not reach the end of the model, we did not deem this worthy of our time to investigate.

The only modification that we made was to remove the skip connection stemming from the model inputs (concatenation of cover and secret images in the encoder case and the embed images in the decoder case). In Wu u. a. (2018) code, this skip connection allows the second layer to access such inputs. We thought this was strange since the inputs do not need any gradient. Their implementation also applied batch normalization and ELU to the model inputs **before** any convolutions were applied. Although it's arguable that this removes the model's ability to easily learn the identity for the first layer, we decided to remove the first skip connection, batch normalization, and activation.

To summarize, our model is composed of an encoder and decoder with the same architecture. The encoder expects 6 input channels (the concatenation of cover and secret) and the decoder expects 3 input channels. The architecture has 6 separable convolutions (with 5 skip connections) followed by 2 normal convolutions.

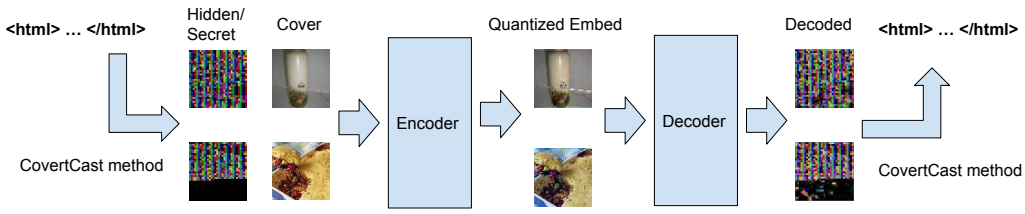

Figure 2: Our model/pipeline with examples from the HTML dataset. First, the HTML is treated as binary and converted to an image using CovertCast method. Second, the secret image is concatenated with a cover image from the Tiny-Imagenet dataset and fed through the decoder. This generates the embed images which are quantized before transmission (e.g. over the internet). Lastly, the decoder extracts the secret image from this embed image and the CovertCast method recovers the HTML.

3.2 Datasets

We used four different inputs for the model: Tiny-Imagenet-200 Li u. a. (2016), HTML files, Shakespeare text dataset, and random dataset.

Tiny-Imagenet-200 is a small subset of the ImageNet dataset Russakovsky u. a. (2014). It contains 200 classes. This small image dataset makes our training fast and efficient.

For the HTML files data, we retrieved the Top 200 websites from the internet and only saved the raw HTML file. However, not all 100 websites responded and some responded with the correct encoding. We only allowed 'utf-8' and 'iso-8859-1' encodings to keep it simple. During the training process, we encode the raw HTML file into the cover image using the CovertCast method we detail later.

We obtained our Shakespeare dataset from Karpathy (2015) from his Char-RNN project. We chose Coriolanus for validation and King John for test. This was simply because they were at the ends of the file which made splitting easy and that they were short enough so that we'd have most of the data

to train. While looking through this dataset, we found that there were some omissions. This most likely did not affect our results as our data generation method yielded millions of examples with only 4MB of data.

To create the random dataset, we first created a tape of 0's and 1's (with equal probability of 1/2) with a length of the number of examples in the Tiny-Imagenet training set (100000) plus the number bits per image (6144 = 32 * 32 * 6). This then allows us to use a sliding window approach of moving along the tape 1 bit at a time to generate a random hidden image for each Tiny-Imagenet example.

3.2.1 CovertCast Encoding and Decoding

To encode binary into images and images back into binary, we used the method proposed from the paper CovertCast McPherson u. a. (2016). Given an array of bits A, we encode two bits into each channel of a pixel using the formula $160 \times A[i] + 32 \times A[i+1]$. Thus, we can encode 6 bits into a single RGB pixel. However, in order to make this image robust to noise, we upscale the image using nearest neighbor and we call this the "six bit resolution". For instance, when the six bit resolution is 2×2, each 4 RGB pixels represents 6 bits. Therefore, for a 64x64 image, we can theoretically encode 6144 bits or 768 bytes.

As it turned out, there was an error in their pseudocode. Upon closer inspection of their paper, we found that they probably intended to write $32 + 128 \times A[i] + 64 \times A[i+1]$. However, we found this out too late after we had trained all of our models. Regardless, this did not hinder us from recovering the binary data for our end to end model. We simply recovered the bits by mapping values of range [0, 16] to **00**, (16, 96] to **01**, (96, 176] to **10**, and (176, 255] to **11**. Later we'll discuss that it's possible that this could affect our results for quantized embeddings.

3.3 Training Setup

All experiments are trained on NVIDIA RTX 3090 GPU with the same configuration. Specifically, models are trained with a batch size of 64 for 100 epochs. We used Adam as optimize with all default parameters. All input data are normalized.

For the HTML dataset, we introduced two modifications to simulate transmission in the real world. First, we reserved 1 byte per example to be a sequence number (that wraps around on overflow). The idea was that the receiver would take screenshots of the broadcast at a higher frequency than the broadcaster's frame rate. This would create duplicate frames which could be easily removed by a small sequence number (which would be robust to edge cases such as data with repetition).

Second, we zeroed out a random amount contiguous pixels in the flattened image. The amount to zero out was picked from a uniform distribution between 0 and 1/4 of the number of pixels. The idea was to simulate the 0 byte padding that would occur at the end of a transmission.

3.4 Loss Function

All models have trained with two sets of loss functions: the image difference loss function and the combination of the image difference loss and correlation loss. In this section, we will explain and explore the two loss functions.

3.4.1 Image Difference Loss

We used the same loss as the Wu u. a. (2018) did in their paper. Specifically, we want to use this loss to minimize the difference between the cover image and the embedded image and the difference between hidden and decoded images. Here we define symbols for Cover Image (C), Hidden Image (H), Embedded Image(E), and the Decoded Image(D). The loss measures the difference between the embedded image the cover image L_{CE} and the loss between the hidden image and the decoded image is L_{HD}. Furthermore, they include the variance between the images to encourage the model to distribute the loss (or the encoded information) evenly across the image, instead of clustering on one location. Therefore, the weighted loss function is:

$$loss = L_{CE} + L_{HD} + Var(L_{CE}) + Var(L_{HD}) \tag{1}$$

In this paper, we will use Hidden and Secret interchangeably. They both refer to the image H.

3.4.2 Additional Correlation Loss

As discussed in the paper Baluja (2017), what if the decoder has access to the original, unmodified cover image? The authors explore the consequences when someone has access to the original cover image by simply calculating the difference between the original cover and the container (encoded image). At 20X enhancement, the residual revealed the feature of the secret image. To combat this, Baluja (2017) designed correlation loss that measures the pixel-wise correlation between the residual and the secret image, denoted:

$$corr := covariance(C - E, H)/\sqrt{Var(C - E)Var(H)} \tag{2}$$

Minimizing this loss would reduce the relationship between the Cover Image and the Embedded Image. Therefore, the final loss function would be equation 1 + equation 2.

$$loss_{corr} = L_{CE} + L_{HD} + Var(L_{CE}) + Var(L_{HD}) + abs(corr) \tag{3}$$

The downside of using this loss function is that the quality of the decoded image has dropped.

4 Experiments

4.1 Experiment Analysis

We have 8 experiments in total. For each input (Imagenet, Random, Shakespeare, HTML), we trained two types of losses: Image Difference Loss and Image Difference Loss + Correlation Loss. Figure 5 is the log mean squared error between the cover and the embedded image. Note that we did not optimize on this metric. We are only taking the log because we are often dealing with numbers smaller than 10^{-3}. While it's meaningless to compare the result across different inputs, it's worth to take a look at the performance between the two loss functions. Although correlation loss reduces the possibility of exposing the secret when someone has access to the original cover image, we can clearly see that the correlation loss increases the loss. There are two possible hypothesis for this: a) The model needs to learn a more complex function. For a set capacity, the model that is the least constrained will perform better. b) The encoder introduces noise which decreases downstream decoder performance.

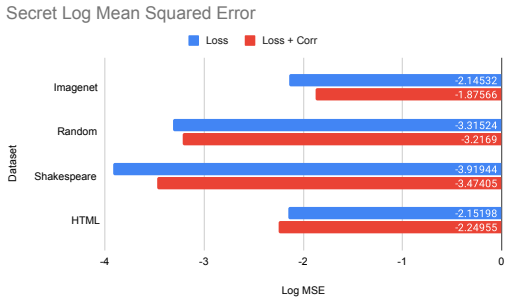

Figure 3: The log MSE difference between secret and decoded image.

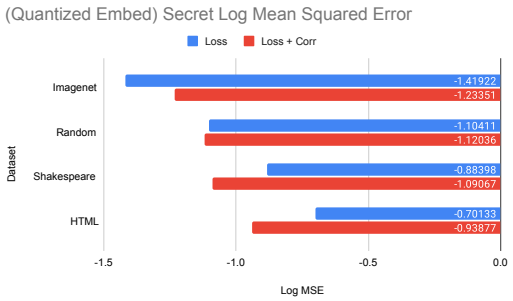

Figure 4: The log MSE difference between secret and the output of the decoder run on quantized embed images.

Figure 3 shows the performance of the model, as it measures the difference between the secret and the decoded image. Figure 4 is the graph for log MSE on quantized embedded images, providing less accurate results. In other words, it shows that the model has trouble recovering the secret from the embedded image when it's perturbed even a little bit. The model's sensitivity causes it to performance worse on quantized embeds. As shown in the graph, the pipeline does a good job of recovering Shakespeare's text (the longer, the better), but it does not perform as well in terms of image and HTML recovery. One noticeable factor is that a small change in the decoded image will not impede the human understanding of an image. It's not the case when it comes to HTML files. As shown in figure 6, the recovered HTML text is not comprehensible.

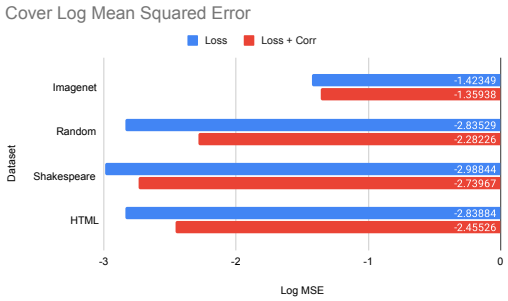

Figure 5: The log MSE difference between cover and embedded image. We used a log scale because the MSEs were less than 1. A longer bar is better because the log MSE is more negative.

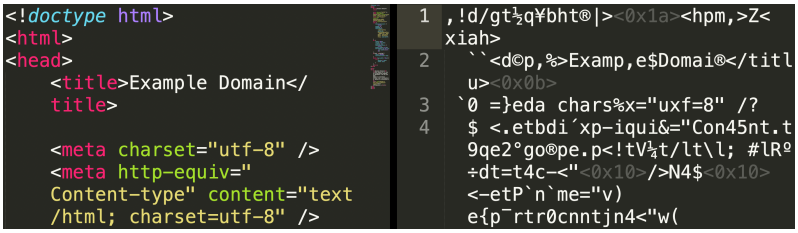

Figure 6: Comparison between the secret message and the recovered message for the HTML model trained without cross correlation.

Finally, we tried to send our encoded stream to Youtube Live, but we failed to do so as the 84% bit accuracy is not enough for a naive decode and broadcast implementation. We will discuss more regarding the real-world application in the discussion section.

4.2 Result Analysis

In figure 7, we display one set of example cover, embedded, secret, and decoded images along with their feature in the frequency domain. We converted all four images via fast Fourier transform (FFT) using the discrete Fourier transform.

As shown in the graph, the embed image contains mostly low-frequency features. (It's faint but it looks like high frequency components in green appear less in the embed FFT compared to the cover FFT.) Also the vertical feature of the embedded image is more observable. There is a minimal change between the FFT graph of the secret image and decoded image – more low-frequency appears.

If someone has access to the embedded image without access to the cover image, it's not likely for them to recognize the embedded image as an encrypted one. Besides a slight color shift for the secret image, the decoder is able to preserve the original shape and texture of the secret image. Also, it's intriguing that the model learns to preserve the structure of the cover image while using color to code for the secret image.

The last column of figure 7 is a departure from the results of Baluja (2017). In their work, the simple difference between the cover image and the embedded image reveals the shape of the hidden one. It's curious that this is not the case for the architecture of Wu u. a. (2018). It's possible that the use of separable convolutions has a role in this discrepancy.

Many previous papers Xu und Wang (2015); Wu u. a. (2018); Fang und Chang (2006) measured their performance through the receiver operating characteristic (ROC) curve. We provided a similar graph 10 to our experiments. The thick green line refers to a model that randomly guesses on the dataset. Since all of our models are close to this line, this means that the detector StegExpose is unable to accurately detect encrypted images. This is surprising as Wu u. a. (2018) had shown StegExpose to have an accuracy somewhat better than random guessing. Shakespeare is the one that's closest to the random guessing line.

Interestingly, the ROC curve shows evidence against our initial hypothesis that correlation loss would minimize detection. Although this difference is very small, it could be due to the encoder purposefully adding noise to minimize correlation. Ironically, StegExpose could be picking up on this noise (meant to de-correlate the secret) as evidence of the image being suspicious.

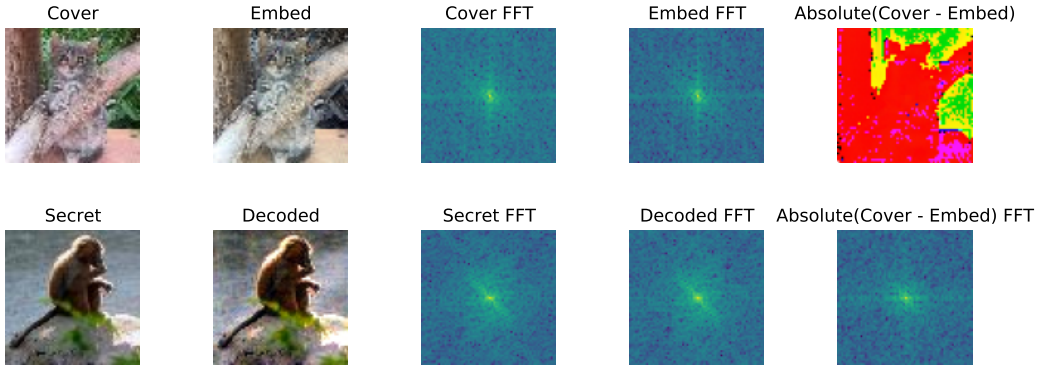

Figure 7: Concrete example from the Imagenet baseline model (without correlation loss). The cover and secret images were concatenated along the channel dimension and fed through the encoder to obtain the embed. The embed is fed through the decoder to obtain the decoded image.

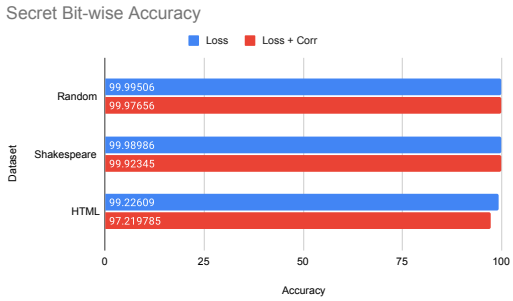

Figure 8: The bit-wise accuracy between the ground truth binary and the data extracted using the CovertCast method on the decoded image.

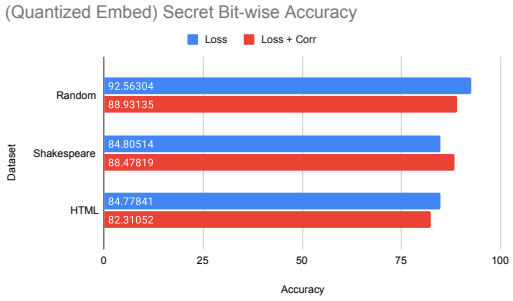

Figure 9: The bit-wise accuracy between the ground truth binary and the data extracted using the CovertCast method on the decoded image outputted by the decoder run on quantized embed images.

5 Further Discussion

We can see that for real data, the correlation loss generally increased both the cover and secret MSE loss. This is interesting because we had initially hypothesized that the correlation loss would encourage the model to learn an embedded image closer to the cover image but with a lossier and more complex encoding for the secret. Instead, it seems that the model has introduced noise to decorrelate the embedding-cover residual from the secret.

One conclusion is that since the correlation loss appears to make the embed images noisier, the decoder is trained to be more robust to noise. This can explain the quantized embed secret loss for Figure 4. While the secret MSE loss is generally greater on quantized embeds compared to its end to end counterpart, the models trained using correlation loss are less affected due to having seen noisy inputs.

According to Figure 9, the random and Shakespeare models trained on correlation loss had an accuracy of about 88%. The approximately 6% accuracy drop found in the HTML model trained with

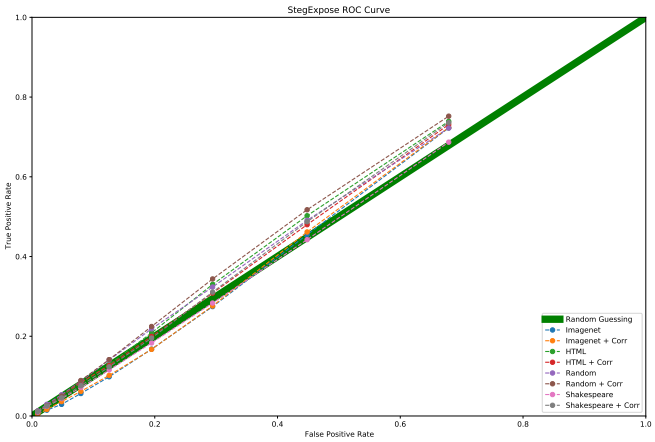

Figure 10: The receiver operating characteristic (ROC) curve for StegExpose on our models.

correlation loss could be due to using random 0 images (to simulate padding in real transmissions). However, since the base loss models for Shakespeare and HTML both performed equally well, this could be due to the data. It could also be a combination of these two including other factors not yet considered. Furthermore, it's not clear why in Figure 8 that the accuracy is nearly 100% for all models except for the HTML correlation loss model. Further investigation is warranted.

One potential solution for increasing accuracy with quantized embed images is to simply create a new dataset from the outputs of the encoder. Then we can finetune the decoder on the quantized encoder outputs.

Another area that should be fixed is the lopsidedness of the encoding. We reserve only 17 values to represent **00** while reserving 80 values to represent **01**. This imbalance gives the model less room to work with (and does not match our loss function which weighs error in both directions equally). Keep in mind that this was due to an erro in the McPherson u. a. (2016) paper. The simplest solution is to re-adjust the CovertCast encode/decode functions and re-train the models.

6 Conclusion

Overall, our work is a departure from previous works in that we analyze the ability to encode arbitrary binaries in images (albeit naively) using deep-learning steganography methods. We've shown that while previous works are promising, there are still ways to go for hiding images produced by a binary-to-pixel encoding, and application on real-world streaming services.

We implemented the model with 2×2 six bit resolution. In the future we could try extra redundancy all the way up to 8×8 six bit resolution used originally by McPherson u. a. (2016). However, this would require more training time as we'd need bigger images to encode the same number of bits.

Another possibility is to add another model into the pipeline: optical character recognition (OCR). Then we could simply render text into images. Our models have been shown to preserve shapes better than color. It's possible that all of this could be totally unnecessary and that we can just simply use the long tried and tested error correction codes.

Finally, we have mostly focused on increasing robustness or accuracy in this section. Correlation loss is interesting in that it attempts to decrease the presence of the secret image by preventing linear relationships. However, this doesn't consider quadratic or other non-linear encodings. Thus, the correlation loss is an idea that naturally leads to using Generative Adversarial Networks. This would have been an excellent approach but we were worried about the difficulty of training them.

References

[Baluja 2017] BALUJA, Shumeet: Hiding images in plain sight: Deep steganography. In: *Advances in neural information processing systems* 30 (2017)

[Cheddad u. a. 2008] CHEDDAD, Abbas ; CONDELL, Joan ; CURRAN, Kevin ; MC KEVITT, Paul: Skin tone based Steganography in video files exploiting the YCbCr colour space. In: *2008 IEEE International Conference on Multimedia and Expo* IEEE (Veranst.), 2008, S. 905–908

[Fang und Chang 2006] FANG, Ding-Yu ; CHANG, Long-Wen: Data hiding for digital video with phase of motion vector. In: *2006 IEEE International Symposium on Circuits and Systems (ISCAS)* IEEE (Veranst.), 2006, S. 4–pp

[He und Luo 2008] HE, Xuansen ; LUO, Zhun: A novel steganographic algorithm based on the motion vector phase. In: *2008 international conference on computer science and software engineering* Bd. 3 IEEE (Veranst.), 2008, S. 822–825

[Karpathy 2015] KARPATHY, Andrej: *char-rnn*. `https://github.com/karpathy/char-rnn`. 2015

[Li u. a. 2016] LI, Fei-Fei ; KARPATHY, Andrej ; JOHNSON, Justin: *Tiny-Imagenet-200*. `https://www.kaggle.com/c/tiny-imagenet`. 2016

[Liu u. a. 2019] LIU, Yunxia ; LIU, Shuyang ; WANG, Yonghao ; ZHAO, Hongguo ; LIU, Si: Video steganography: A review. In: *Neurocomputing* 335 (2019), S. 238–250

[McPherson u. a. 2016] MCPHERSON, Richard ; HOUMANSADR, Amir ; SHMATIKOV, Vitaly: CovertCast: Using Live Streaming to Evade Internet Censorship. In: *Proceedings on Privacy Enhancing Technologies* 2016 (2016), S. 212 – 225

[Mstafa und Elleithy 2015] MSTAFA, Ramadhan J. ; ELLEITHY, Khaled M.: A novel video steganography algorithm in the wavelet domain based on the KLT tracking algorithm and BCH codes. In: *2015 Long Island Systems, Applications and Technology* IEEE (Veranst.), 2015, S. 1–7

[Mstafa und Elleithy 2016] MSTAFA, Ramadhan J. ; ELLEITHY, Khaled M.: A video steganography algorithm based on Kanade-Lucas-Tomasi tracking algorithm and error correcting codes. In: *Multimedia Tools and Applications* 75 (2016), Nr. 17, S. 10311–10333

[Russakovsky u. a. 2014] RUSSAKOVSKY, Olga ; DENG, Jia ; SU, Hao ; KRAUSE, Jonathan ; SATHEESH, Sanjeev ; MA, Sean ; HUANG, Zhiheng ; KARPATHY, Andrej ; KHOSLA, Aditya ; BERNSTEIN, Michael ; BERG, Alexander C. ; FEI-FEI, Li: *ImageNet Large Scale Visual Recognition Challenge*. 2014. – URL `https://arxiv.org/abs/1409.0575`

[Shahid u. a. 2011] SHAHID, Zafar ; CHAUMONT, Marc ; PUECH, William: Fast protection of H. 264/AVC by selective encryption of CAVLC and CABAC for I and P frames. In: *IEEE Transactions on Circuits and Systems for Video Technology* 21 (2011), Nr. 5, S. 565–576

[Tang u. a. 2019] TANG, Weixuan ; LI, Bin ; TAN, Shunquan ; BARNI, Mauro ; HUANG, Jiwu: CNN-Based Adversarial Embedding for Image Steganography. In: *IEEE Transactions on Information Forensics and Security* 14 (2019), Nr. 8, S. 2074–2087

[Wu u. a. 2018] WU, Pin ; YANG, Yang ; LI, Xiaoqiang: Stegnet: Mega image steganography capacity with deep convolutional network. In: *Future Internet* 10 (2018), Nr. 6, S. 54

[Xu und Wang 2015] XU, Dawen ; WANG, Rangding: Context adaptive binary arithmetic coding-based data hiding in partially encrypted H. 264/AVC videos. In: *Journal of Electronic Imaging* 24 (2015), Nr. 3, S. 033028

A Appendix

Figure 11: Simulated CovertCast Image (Their code was so old that it did not work anymore.)