**Steven Li**

**Computer Systems Lab**

**Dr. Peter Gabor**

**24 May 2022**

**Automated deep learning-based deepfake detection via a Chrome Extension**

**Introduction**

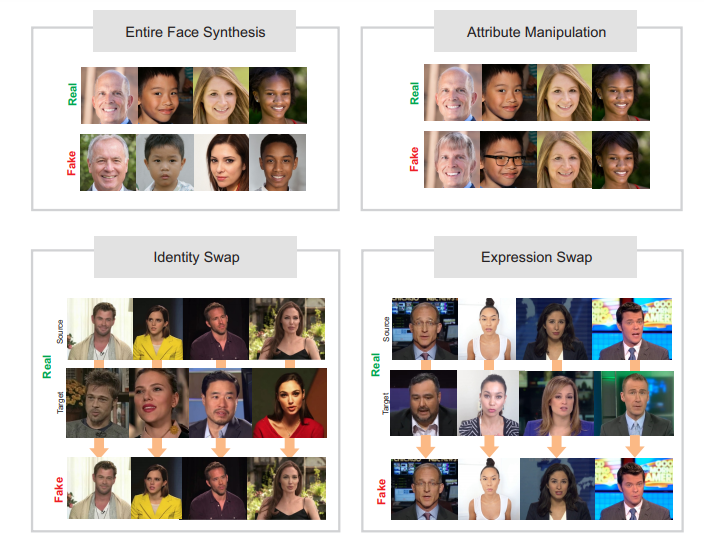
Recently, there has been a rise in fake videos and images of people created by digital manipulation, called deepfakes. Deepfakes alter videos by swapping one person’s face with another one using artificial intelligence techniques. There are two main methods of doing so. The first is by using an autoencoder to find similarities between the two faces and compress the images, and then using a decoder to recover the faces from the compressed images (5). The other method is using a generative adversarial network to generate synthetic images (5). By doing this process over every frame of a video, highly-realistic fake videos can be created. There are four main types of alterations as shown in Figure 1: entire face synthesis, attribute manipulation, identity swap, and expression swap (2). 

Figure 1: A demonstration of the four types of deepfakes

.

This process was initially created in 2017 by a redditor named “deepfakes” who was able to create fake videos with celebrity faces transplanted. Despite being a recent creation, deepfakes have already become a major public concern. In 2018, an artificially-generated video, shown in Figure 2, of Barack Obama calling Donald Trump an expletive went viral on the internet (7). Although it was actually a public service announcement that meant to serve as a warning against deepfakes, it accurately demonstrated the impact and deception that deepfakes could have. 

Figure 2: A deepfake where the movements and speech of comedian Jordan Peele are transplanted onto Barack Obama’s face through deep earning

In 2019, a deepfake of Mark Zuckerberg talking about controlling users’ data emerged, causing backlash against Facebook (4). These two examples demonstrate the possible implications that deepfakes can have on society. Other harmful usages of deepfakes include fake news, hoaxes, and fraud. In addition, since 2017, the process of creating synthetic images and videos have improved, and have also become more accessible to anyone. For example, apps such as FaceApp allow anyone to generate fake images. The rapid growth of AI-created videos and images online means that gradually, it will be harder to trust anything on the internet.

**Existing Works**

There are some existing solutions to detecting machine-created facial alterations, as well as readily available testing databases online such as FaceForensics++. Most of the solutions involved deep learning techniques such as convolutional neural networks (CNN). Mitra et al. was able to achieve an Area Under the Curve (AUC) score of 98.5% overall with the Face- Forensics++ dataset (3). First, they developed a novel key frame extraction process which lowered the computational requirements for running a classifier. Then they ran a convolutional neural network, Xception, to extract features and classify using a network of GlobalAverage- Pooling2D, 0.5 dropout layer, a fully connected layer, another 0.5 dropout layer, and a softmax layer.

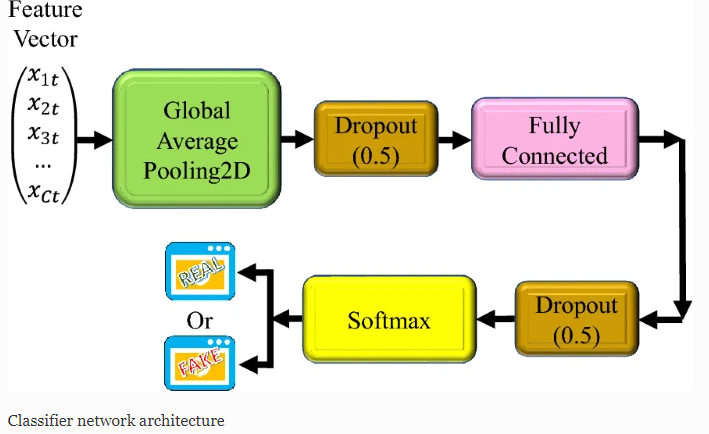


Figure 3: The architecture of the Xceptionnet neural network

Dang et al. also utilized Xceptionnet to detect forgery (1). However, they additionally proposed using an attention mechanism to automatically detect important regions of the image, which would increase the effectiveness of learning and extracting features. With this, they were able to achieve an AUC of 100% with entire face synthesis, as well as 99.43%, 99.40%, and 99.92% respectively for identity swap, expression swap, and attribute manipulation. Other studies proposed alternative techniques for classification such as k-Nearest Neighbors, Support Vector Machine, and Linear Discriminant Analysis (2). Another successful network for detecting deepfakes was MesoNet, which was able to detect Face2Face deepfakes at a 98% rate (7).

**Original Proposal**

Currently, there are few ways for the average internet consumer to check if a video is a deepfake. The only prominent solution is Deepware, a website that takes a link of a video and classifies the video as real or fake. However, this would be inapplicable in most cases as someone scrolling through the internet would not think to check if a video is a deepfake or not, since newer deepfakes look highly realistic. Instead, our goal was to build a chrome extension that automatically scans the websites for any deepfakes, and notifies the user of any possible deepfakes. First, the chrome extension will search for and extract the video from the webpage. Then, it will run the deepfake detection algorithm. We will first only extract key frames, which is any frame of the video that marks the beginning or end of a transition. Mitra et al. used this technique rather than extracting all of the frames in order to lower the overall computational cost (3). Then we would utilize the convolutional neural network Xception for feature extraction and classification, with a convolutional layer first followed by a Max Pooling layer, then the attention mechanism, a 0.5 dropout layer, a fully connected layer, another 0.5 dropout layer, and lastly, the softmax layer for classification. The attribute mechanism is placed near the front of the classification network since Dang et al. reported that they achieved better AUC scores that way (1). By doing this, we would make the algorithm efficient while attaining a high accuracy like Dang et al. did. We will also use the FaceForensics++ data set for training and testing. Our goal by the end of the year is to have a working chrome extension that will detect deepfakes with an AUC score of 95% overall for all 4 types of facial alterations, and notifies the user of possible deepfakes within 5 seconds.

**Chrome Extension Construction**

Our first step in building the chrome extension was to find a method of scraping videos from a given webpage. We planned to use the PyTube library, a Python library that can download Youtube videos. In order to run a Python script in the chrome extension, we wanted to use Rapydscript which compiles Python to Javascript, as well as Node.js. However, we realized that chrome extensions don’t support Python scripts. We then decided to use the Javascript library Youtube-dl, but we ran into the same issue: chrome extensions don’t support Node.js, and Youtube-dl is hosted on Node.js. Our solution to this problem was to set up a Flask server in TJ Director, where the Flask endpoints could accept a query parameter containing a URL to a Youtube video from the server. The server would then use PyTube to return the video back to the extension. This server would also allow us to run Python code on the images to detect for deepfakes.

While we were figuring out how to scrape the video from the website, we also built the initial UI for the chrome extension popup. Our first proposed UI intended to display every video that the website contained:

Figure 4: An initial blueprint of our envisioned extension UI

However, we soon scrapped this idea as the user is normally only watching one video at a time. Instead, we would just display information about the user’s current video. We decided to display the thumbnail and video title that the extension received from the server.

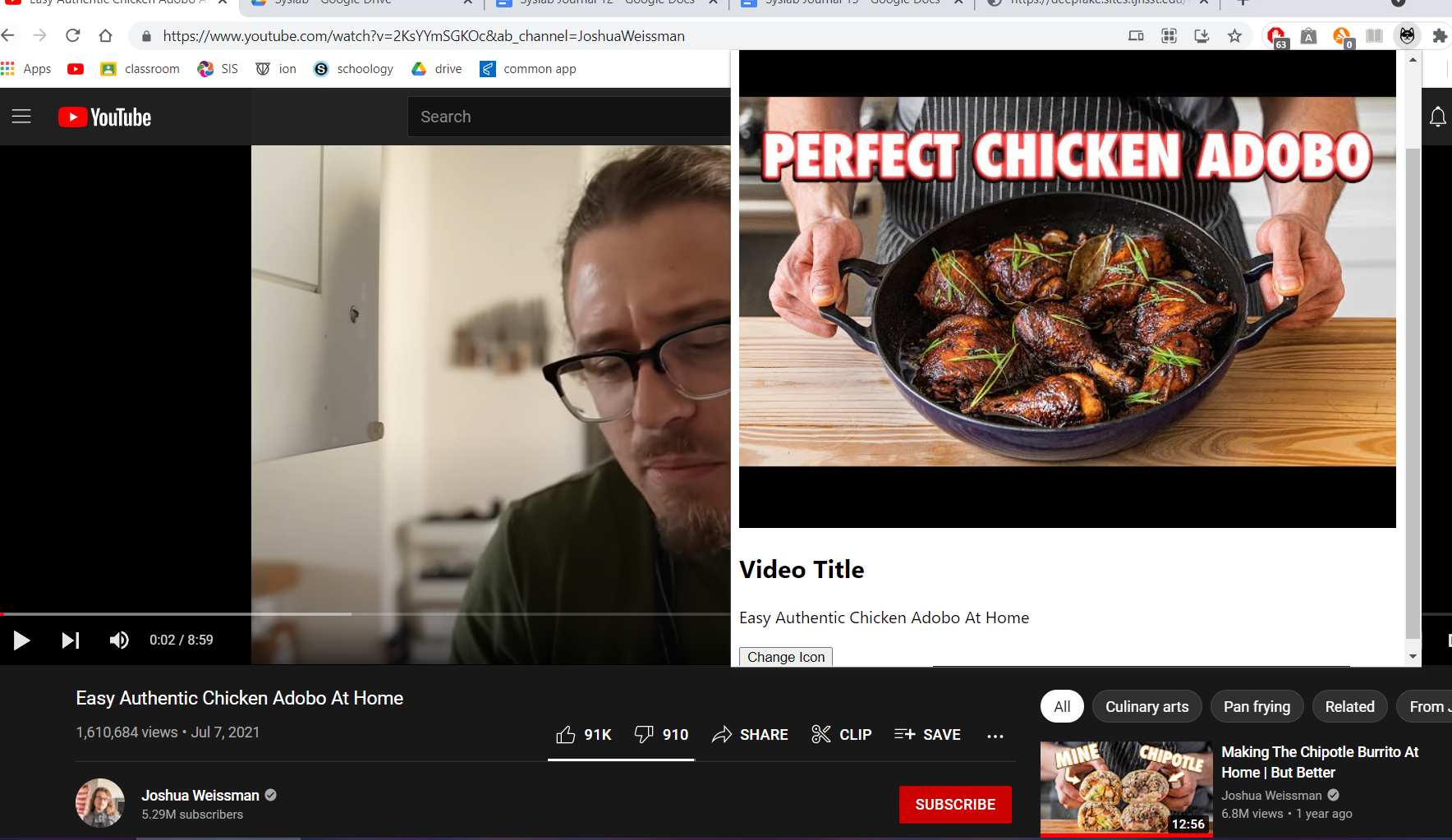


Figure 5: Our intermediate chrome extension UI which displays the thumbnail and title of the video

In addition, the PyTube library had functions which made videos readily available to download, and we planned to process the downloaded video in order to extract key frames. However, we found that downloading even a short, low-quality video took minutes to occur, and this was unsuitable for our project. While we found optimizations to the downloading process such as downloading in chunks and using asynchronous requests, the time required for downloading videos was still too large. Moreover, we found during our additional research that Deepware.ai, the leading deepfake detector, was extremely inefficient as well, timing out in any videos over five minutes. We realized that we would have to grab the image frames directly from the website.

Our new solution was to simply take a screenshot of the video every five seconds, and we planned to use the tabCapture Chrome API to take pictures of the tab which we could then process into a suitable frame. This method massively improved upon the inefficiency of the video downloading method as only a few frames of a video had to be processed in comparison to thousands. The user would also be able to enable or disable the extension in a manner similar to an ad blocker extension. We further improved upon this method by using the HTML QuerySelector to directly grab the frame of the video which eliminated the need of capturing the whole tab and processing the image. The last step of the chrome extension was to send the frame from the extension to the server using an HTTP Post Request. Post Requests can only contain binary data so we converted the image into a base64 string before sending it to the server.

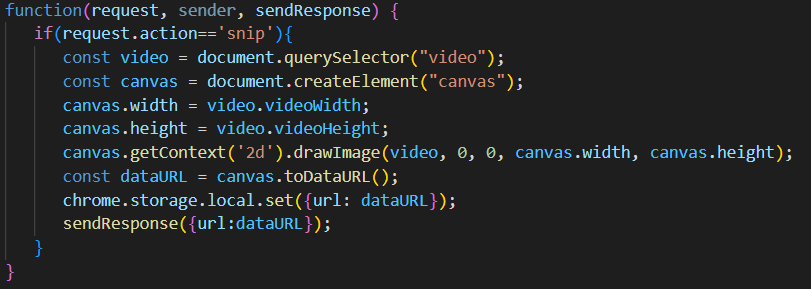


Figure 6: The code to grab the frame of the video. We use HTML QuerySelector to grab the current video, and paste it onto a canvas which is converted into a URL containing an image.

Lastly, we finalized the UI of the chrome extension. We used CSS to improve the look of the extension.

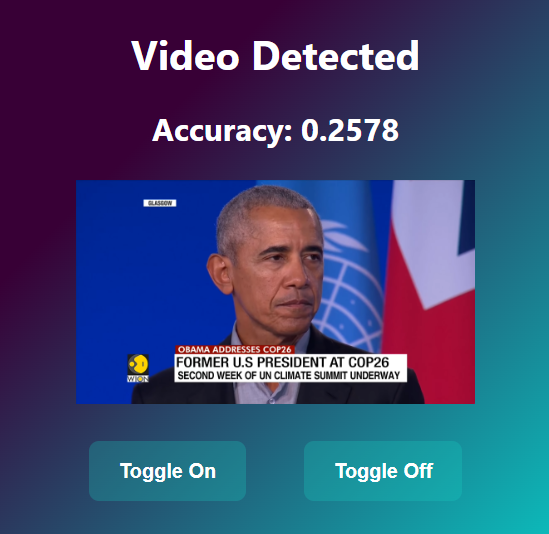
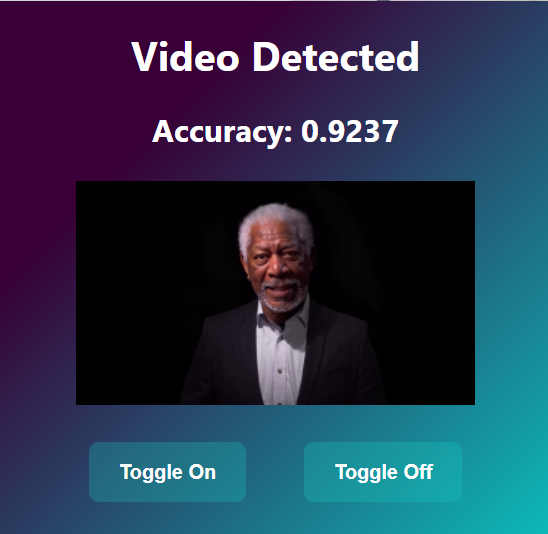


Figure 7: The extension in action on a deepfake of Morgan Freeman and a real video of Barack Obama

The last step was to alert the user whenever the probability of deepfake reached a certain threshold, and we set this threshold to 0.6. Our two options were to either use a chrome alert, or forcefully open a new tab in the user’s browser. We chose the latter option as most modern chrome extensions open new tabs, and chrome alerts have fallen out of common use.

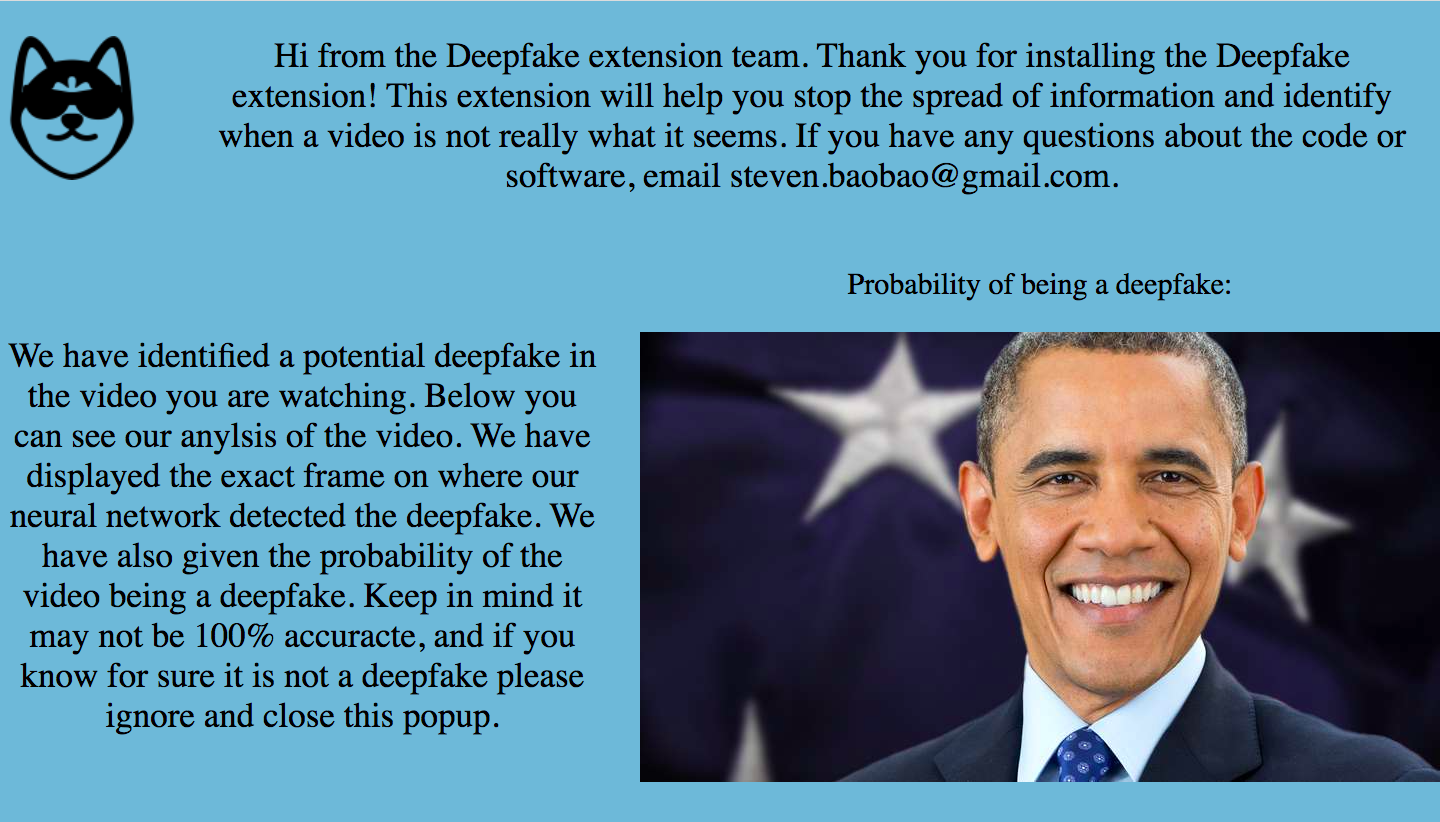


Figure 8: When the probability of deepfake is above 0.6, this tab will open to alert the user of a potential deepfake.

**Preprocessing/Neural Network**

We decided to use the MesoNet convolutional neural network for deepfake detection because the code was publicly available online (7). In order to build MesoNet, we used the deep-learning library Tensorflow. The convolutional neural network would be contained on the server. However, TJ Director did not contain enough space to import the Tensorflow library. Therefore, we set up a new server on Google Colab using NGROK, and we were able to install Tensorflow.

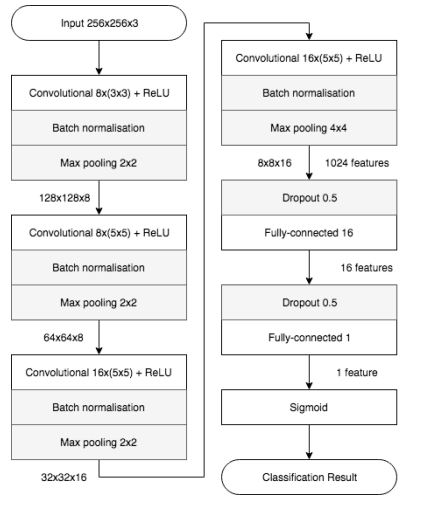
Next, we constructed the convolutional neural network. The network receives an input of size 256x256x3. Then, for feature extraction, we convolve the image with the ReLu activation function, run Batch Normalization, and use Max Pooling. This sequence occurs four times in total. Then for classification, we flatten the image and utilize a dropout layer before running a fully-connected network. Lastly, we use the sigmoid function to return the probability of a deepfake. We then used the FaceForensics++ dataset to train and test the network. We split the dataset into an 80-20 train-test split, and trained the network. We obtained a 90% accuracy on the testing data.

Figure 9: The neural network architecture

We then began preprocessing the image sent from the extension in the server. First, we used the PIL library to read the base64 string into a PIL image, which was then read into an numpy array. We then scaled the image directly to match the 256x256 size required for the network. However, this skewed our results as the aspect ratio of the image was altered. We then attempted to first scale the image down while maintaining the aspect ratio and then add padding to reach 256x256, but this was also ineffective because the size of the faces in these adjusted videos did not match the faces from the training dataset. We solved this by using the HaarCascade algorithm in OpenCV to detect and crop out faces in the image, which allowed us to start obtaining strong results in our extension.

**Results**

The architecture of our extension was successful as it was able to scrape the video and run the convolutional neural network on the video, thus accomplishing our original goal. We were also pleased with the final UI of the chrome extension. As for the performance of our network, we were able to achieve our time goal as the extension returns the probability of deepfake about two seconds after scraping the frame. We also obtained an accuracy of 90% on the FaceForensics++ testing dataset. However, when testing on various YouTube videos, the neural network seemed to perform well less consistently, and we estimated this new accuracy on YouTube videos to be about 70%. The reason for this was likely because it was difficult to crop faces out of YouTube videos in a way that matched the images from the original training dataset, and this would've skewed the performance of the neural network. Refining the face cropping algorithm would help increase the accuracy of our network for public use.

Additionally, minor bug fixes would improve the user's experience. One issue is that the face detection algorithm, HaarCascade, sometimes detects objects that aren't faces. However, this is just a limitation of current computer vision methods. Another issue was that the extension had trouble scraping videos on websites that had multiple videos. On Twitter, the extension often scraped the wrong video, and on Instagram, the extension often crashed and we were unable to successfully debug this problem. However, fixing these minor bugs would allow finalization of this project and deployment into the Chrome Store for public use.

References

1. <https://arxiv.org/pdf/1910.01717.pdf>
2. <https://arxiv.org/pdf/2001.00179v3.pdf>
3. <https://link.springer.com/article/10.1007/s42979-021-00495-x>
4. <https://www.theguardian.com/technology/2020/jan/13/what-are-deepfakes-and-how-can-you-spot-them>
5. <https://mitsloan.mit.edu/ideas-made-to-matter/deepfakes-explained>
6. <https://www.youtube.com/watch?v=bE1KWpoX9Hk>
7. <https://arxiv.org/pdf/1809.00888.pdf>