2023

A Machine Learning Approach to Deepfake Detection

Delaney Conrad

A Machine Learning Approach to Deepfake Detection

by

Delaney Conrad

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of Bachelor of Science

in

Cognitive Science

with an emphasis in Computer Science

Minnesota State University, Mankato

Mankato, Minnesota

May 10, 2023

We approve the thesis of Delaney Conrad

Date of Signature

Rushit Dave
Assistant Professor, Department of Computer Information Science
Thesis Advisor

Rebecca Bates
Professor & Chair, Department of Integrated Engineering
Thesis Committee

Julie Wulfemeyer
Associate Professor, Department of Philosophy
Thesis Committee

Abstract

The ability to manipulate videos has been around for decades but a process that once would take time, money, and professionals, can now be created by anyone due to the rapid advancement of deepfake technology. Deepfakes use deep learning artificial intelligence to make fake digital content, typically in the form of swapping a person's face in a video or image. This technology could easily threaten and manipulate individuals, corporations, and political organizations, so it is essential to find methods for detecting deepfakes. As the technology for creating deepfakes continues to improve, these manipulated videos are becoming increasingly undetectable. It is crucial to create methods to combat this problem. Previous research has been conducted on the various techniques to detect deepfakes, and though some models show promising results, many models struggle with reproducibility and practicality when exposed to real-world scenarios. Future work could consist of creating models without the tools used to generate deepfakes and the collected dataset in mind. Thus, the aim was to create a more general model that could be repeated on a variety of real data. To achieve this, a deepfake dataset was used to train models, and the results were analyzed. After comparing the strengths and limitations of previous models, we created simple, machine learning models that can accurately detect real-world deepfake images. Three methods, random forest, KNN, and SVM were utilized, and all achieved high accuracies compared to state-of-the-art models. Random forest had the best detection performance with accuracy results over 98%, followed by KNN and SVM. As deepfake technology continues to accelerate, it is essential to continue building models that can detect them because if not, it will be impossible to discern digital truth from reality.

Table of Contents

Table of Figures

List of Tables

Acknowledgements

I would first like to thank Minnesota State University, Mankato for helping me get to where I am today and allowing me to make so many wonderful connections. In addition, I give my biggest thanks to my professor and advisor Rushit Dave for guiding and supporting me while I conducted this research. I thank my advisor Becky Bates for helping and providing suggestions along the way. Finally, I thank my family for being so supportive, and my boyfriend Hayden Zeigler for cheering me on and supporting me through the stressful times.

Chapter 1

Introduction

Deepfakes are a type of digitally manipulated image in which the face of an individual is altered and replaced by the face of another. Using deep learning technology, any individual is now able to manipulate images in seconds to provide highly realistic and convincing content. This technology is advancing at a rapid rate, and it is becoming more difficult to discern real images from fakes. Humans are already subject to manipulation by these images and they are only becoming more and more indistinguishable. As this process continues, individuals will not be able to distinguish the reality of any online media. This issue produces a significant threat to the authenticity of digital content. Deepfakes can be used to deceive individuals, manipulate organizations, and spread false information on a massive scale.

Recently, there has been a surge in using artificial intelligence (AI) to generate self-portraits or fictitious people. This type of technology certainly poses similar risks to people and society. The techniques used to create the generated images are similar and use the same algorithms as those used to create deepfakes, but the fictitious images have been created too recently to be included in the dataset. So, this research does not address the challenges of detecting artificially generated images. Instead, this work focuses on detecting deepfake images that involve face manipulation or swapping, which includes alterations of the entire face or portions of the face.

Deepfakes have been used for various purposes, including political propaganda, revenge porn, and financial fraud. For example, deepfakes can be used to spread false information about a political candidate or to incriminate an individual in a crime they did not commit [5]. In some cases, deepfakes can even be used to impersonate someone in a video call or to create fake endorsements for products or services. While deepfake and artificially generated technologies can have potential humorous benefits, such as creating realistic or entertaining characters for use in media, games, and marketing, individuals should not be misled by the content they encounter. Because of this and the potential malicious actions, there is an urgent need to develop effective methods for detecting manipulated videos and images.

In response to the urgent need for detection, much research has been done on the topic in recent years [1-3, 6, 9, 10, 11, 13-17, 19-23, 26, 30-34]. This has, in hand, produced better techniques for creating fakes. In order to detect more sophisticated manipulated images, researchers must first create convincing datasets to train their models with. So, there is both a battle to detect images and create challenging fake images. Deep learning and neural network techniques are commonly used both to create and detect false images. This creates an increasingly complex cycle between the generator and the detector, but it does not necessarily benefit the reproducibility of detection when applied to real-world issues.

Instead of trying to add more levels of sophistication to already complex models that are discussed in Chapter 2, simple machine learning techniques were used in this work to detect real-world deepfake images. Chapter 3 discusses the WildDeepfake dataset chosen and the processing steps. In Chapter 4, different feature extraction methods from the scikit-image library [28] are applied, and features are extracted. Then, common machine learning models, including random forest, k-nearest neighbor (KNN), and support vector machine (SVM), were trained and

tested against the features extracted from the images using scikit-learn functions [29]. The results in Chapter 5 demonstrate that less sophisticated models may successfully detect deepfake images. By sticking with simple methods, the results show that machine-learning techniques can achieve high accuracy scores and may be a viable option for detecting deepfakes. More analysis is provided in Chapter 6. Finally, Chapter 7 discusses that more research is necessary to optimize these techniques and address the limitations, but this study highlights the potential of simpler machine learning models in the fight against deepfakes.

Chapter 2

Literature

The rise of deepfake technology has raised concerns about the ability to distinguish between authentic and fake facial content. This issue has become increasingly relevant due to the potential for this technology to be used for malicious purposes [5]. As a result, researchers have been exploring different methods to detect deepfakes, with a particular emphasis on using machine learning and deep learning algorithms. The aim of this literature review is to examine the existing research on deepfake detection using machine learning. Specifically, the different techniques used by researchers in the field, as well as some of the strengths and limitations of each approach will be analyzed. By synthesizing the findings of these studies, insights into the current state of research on deepfake detection and areas for further research will be identified.

2.1 Background

To understand the work of this thesis, it is first necessary to understand machine learning. Machine learning is a subfield of artificial intelligence that involves training algorithms to make predictions based on the data. This includes a variety of models and types of learning including supervised, unsupervised, and reinforcement. Supervised learning involves training algorithms on labeled data, in which the model is exposed to data with the category it belongs to, and then it is tested by predicting the labels or category on unseen data. Unsupervised learning, on the other hand, involves training algorithms on unlabeled data, with the goal of discovering underlying patterns or structures in the data. Finally, reinforcement learning involves training algorithms to

learn from feedback using rewards or penalties [4, 24]. For this thesis, all of the models use supervised learning as the photos of deepfakes are labeled as “real” or “fake.”

Deep learning is a subset of machine learning that uses neural networks to compute complex relationships between inputs and outputs. Neural networks are composed of layers of interconnected nodes that process data and make predictions. Due to the complex interconnection of nodes, these models resemble the structures of the human brain and are capable of learning and adapting to new information. This type of learning has proven useful in deepfake detection because the extra layers or nodes add more complexity to the model and require less human intervention, allowing them to analyze more complex input data [8]. This is particularly useful for feature extraction in images, as less work is needed in processing the data beforehand. However, more simple machine learning techniques prove to be just as capable of deepfake detection.

2.2 Search Strategy

To identify relevant papers for this literature review, a comprehensive search of databases was conducted including Google Scholar [12] and Papers With Code [25], a site that hosts academic papers and connects with the arXiv database. In addition to papers, they provide connected software and datasets. This was helpful in finding papers that used the datasets of interest, including WildDeepfake or Deepfake Detection Challenge (DFDC) datasets. In searching, a combination of keywords was used such as "deepfake detection," "machine learning", and "face forgery". The search was limited to studies published after 2017. The inclusion criteria for the studies were as follows: (1) the study identified a method for deepfake detection and utilized either the WildDeepfake or DFDC dataset, (2) the study was published in a peer-reviewed journal, and (3) the study provided an evaluation of the proposed deepfake

detection method. After applying the inclusion criteria, fifteen papers were identified that met the criteria for inclusion in this literature review.

2.3 Datasets

The search began with an exploration of the DeepFake Detection Challenge (DFDC) and the papers surrounding it [10]. The creators of this paper and dataset included industry partners and experts, such as Facebook and Kaggle, who publicly released this large dataset to encourage deepfake detection. The DFDC dataset is the largest deepfake detection dataset to date, containing over 128,000 total videos, with over 104,000 of them being unique fake videos. Comparing this to the size of others available shows an impressive jump in scale, as shown in Figure 1. It was created by commissioning videos of individuals who consented to have their images manipulated and used in a deepfake dataset. The videos were pre-processed using a face tracking and alignment algorithm and then further manipulated using a variety of current state-of-the-art deepfake generation methods, including the Deepfake Autoencoder method, MM/NN face swap, FSGAN, StyleGAN, refinement, and audio swaps. The dataset was divided into four sets: training, validation, public test, and private test, and the training and validation sets were publicly released. The dataset also includes various augmentations including face masks and Poisson blending to improve the quality of the deepfake videos and make detection more challenging.

Not only was this dataset interesting in regards to its size and quality, but there was also a competition behind it. In order to accelerate research within deepfake detection, the creators held a Kaggle competition with the public dataset [9]. The competition had over 2,000 participants. During the contest, the public test set was used to evaluate the models, but when it came to actually judging, the models were evaluated with a private, never-before-seen, testing set. This,

expectedly, caused precision and accuracy to drop. The average precision of the top models was 75.30% [10]. More information about the competition can be found on Kaggle [9].

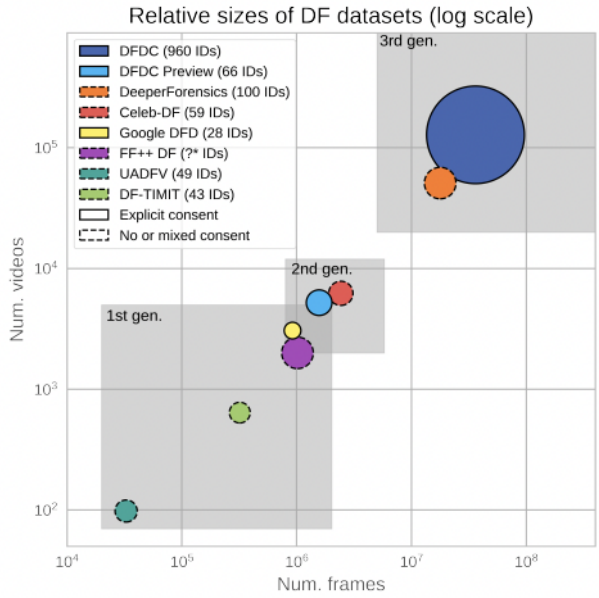

Figure 1: Size comparison of popular deepfake datasets. The DFDC is significantly larger than any other available. The boundaries show rough dataset "generation" (image from [10]).

In addition to this popular dataset, the WildDeepfake dataset was intriguing. Unlike the DFDC which was created with consenting individuals and algorithms to create the deepfakes, the WildDeepfake dataset was collected completely from the internet with unknown creation methods for individual images [34]. It contains 7,314 face sequences from 707 videos, making it a smaller but useful dataset. The authors hoped this would support the development of more effective deepfake detectors by using it along with other datasets. The dataset is diverse and contains a variety of quality, angles, and unknown deepfake methods, making it a challenging dataset for deepfake detection. The authors argue that existing virtual deepfake datasets lack diversity and have low quality, which may not fully generalize to real-world deepfakes. The paper goes on to conduct experiments with baseline detection networks on both existing and

WildDeepfake datasets. Then, the authors, Zi et al., propose their own method for detection using two new attention-based deepfake detection networks (ADDNets) for more advanced deepfake detection. The proposed methods use an attention mask to adjust the feature map at different abstraction levels, making the method more effective. This dataset was additionally interesting due to the lack of research conducted on it. Even though it has a variety of high-quality deepfakes, it is used much more rarely in papers than the DFDC dataset. In addition, papers and previous methods that it has been tested on have received lower accuracies. Zi et al. believe 3D detection networks may be less effective than 2D networks for WildDeepfake detection due to distorted temporal information in deepfake face sequences. Thus, the temporal information in their videos needs to be treated differently from that in real videos in order to improve the accuracy of sequence-level deepfake detection [34].

Another popular dataset is FaceForensics++ (FF++) [26]. It was created using 1,000 real videos from the internet. These videos were manipulated using four popular deepfake techniques including Deepfake, Face2Face, FaceSwap, and NeuralTextures, resulting in 4,000 fake videos to test against. Other popular datasets were created similarly, with fewer unique identities and algorithmic manipulations. For example, the DF-TIMIT [19] dataset contains 43 unique identities, UADFV [33] contains 49, Google DFD [11] contains 58, and Celeb-DF [20] contains 59. The models created from these datasets struggled to generalize due to the small number of unique subjects [10].

2.4 Previous Methods

When attempting to find “machine learning” methods for detection, it was surprising how few papers used simple machine learning methods. Even papers claiming to use machine learning would quickly use more advanced techniques involving deep learning and neural

networks. Due to time constraints, these techniques have not been studied or worked with. Still, an attempt was made to learn as much as possible from these papers.

One of the most helpful was a large research project conducted on the current state of deepfakes [17]. In it, Juefei-Xu et al. analyzed current deepfake generation and detection methods, along with the battleground between the two. The authors highlight how the battle between generation and detection in deepfakes leads to improvements in both and inspires new directions of research, such as avoiding deepfake detection. After analyzing over 318 papers, Juefei-Xu et al. categorize both generation and detection methods. However, this review focuses solely on deepfake detection methods.

2.4.1 Deepfake Detection Categories

In the deepfake detection section of this study [17], Juefei-Xu et al. highlight three main types of deepfake detection methods: spatial-based, frequency-based, and biological-based. In spatial-based detection, models focus on observing various artifacts in the spatial domain to distinguish real and fake. This includes subsections of image forensics-based detection, which inspects the disparities at the pixel level using a variety of techniques, and deep neural network (DNN) based detection, which uses existing or new DNN models to extract spatial features in the images. Frequency-based detection methods can reveal artifacts in fake images that are not easily detectable in the spatial domain. These differences arise from the imperfections in the generative adversarial networks (GANs), which is a generative model that works by "fighting" itself to create more convincing data. Biological-based detection methods use natural signals in real images and videos to distinguish them from fakes. This includes the mismatch in visual and audio signals, the lack of naturalness in synthesized faces, and biological signals like motion, facial expressions, lip-sync, and even heart rate.

The spatial and frequency-based methods above work well when the deepfake data exhibits obvious visual artifacts, but both have poor generalization to unknown techniques and low robustness to adversarial attack, which means the model is easily fooled or manipulated by small changes made to the input data. As a result, the model is unable to detect examples that are created to exploit its weaknesses, and it is not reliable in real-world scenarios where such attacks are common. The authors argue that in the near future, deepfakes could be so realistic that spatial and frequency artifacts are no longer detectable. This would lead to biological signals being a more effective solution for fighting real-world deepfakes. However, this solution could lose its validity as biological signals become more enhanced in fake videos. Juefei-Xu et al. make note that there are other detection methods that did not fit into these categories, but these prove to be less popular approaches. Figure 2 provides a summary and visualization of these detection methods. Overall, the authors found that the popular choice for deepfake detectors was convolutional neural networks (CNN) models, at least as a backbone. CNN are a type of deep learning often used for visual tasks in image classification because its convolutional layers allow for feature extraction. In addition, linear machine learning methods were rarely employed for detection.

Type ID	Name
Type I1	Image Forensics based Detection
Type I2	DNN-based Detection
Type I3	Obvious Artifacts Clues
Type I4	Detection and Localization
Type I5	Facial Image Preprocessing
Type II1	GAN-based Artifacts
Type II2	Frequency Domain
Type III1	Visual-audio Inconsistency
Type III2	Visual Inconsistency
Type III3	Biological Signal in Video
Type IV1	Others

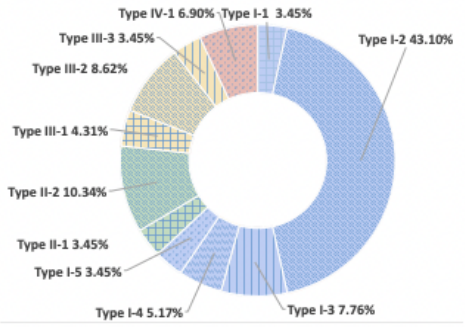

Figure 2: Summary of deepfake detection methods. Blue represents spatial, green is frequency, yellow is biological, and red is other methods (image from [17]).

2.4.2 Model Evaluation

Models and machine learning methods can be evaluated on a variety of different evaluation metrics. Previous research does not follow a standardized procedure for evaluating models. Thus, researchers must compare a variety of different metrics. Popular metrics include accuracy, precision, recall, F1 score, confusion matrices, ROC curves, and area under the curve (AUC).

Accuracy is a metric that measures the proportion of correctly classified instances out of all instances. Precision and recall are metrics that evaluate the performance of the positive class. Precision measures the proportion of correctly classified positive instances out of all instances classified as positive. Whereas, recall measures the proportion of correctly classified positive instances out of all actual positive instances. The F1 score is a combined measure of precision and recall that provides a single value to evaluate the model's overall performance.

These can all be visualized and understood more easily with a confusion matrix. A confusion matrix shows the true positive, true negative, false positive, and false negative counts for each model. The true positive count represents the proportion of positive instances that were correctly classified as positive, while the false positive count shows the proportion of negative instances that were incorrectly classified as positive. The true negative count and false negative count represent the same values but for negative instances.

Finally, a ROC (receiver operating characteristic) curve is a graph that plots the true positive rate against the false positive rate. Because of this, it is easier to evaluate and compare the performance of different machine learning models. The area under the curve (AUC) is referring to the area under the ROC curve, and it measures the entire 2-dimensional area under the curve. This means a model that classifies everything completely correctly has an AUC of

100%. This helps provide an aggregate measure of performance across classification measures. By using these performance metrics and evaluation methods, the effectiveness of machine learning models in detecting deepfakes can be determined and decisions can be made about which models to use for future research.

2.4.3 Models

Because they are commonly deployed and have the most research conducted on them, a valuable contrast to the simpler methods used in this thesis are approaches that use convolutional neural networks (CNN). A CNN is a class of deep neural networks commonly used to analyze images and visual data. Bonettini et al. [3] present a method for detecting manipulated faces in videos using an ensemble of different CNN-trained models. Ensembling is accomplished by using multiple models and voting on the best to improve prediction performance. The EfficientNet family of models was used as a starting point. This family of models is able to automatically scale CNN and is known for their high accuracy and efficiency when compared to other CNN models. The EfficientNetB4 was specifically chosen due to its low number of parameters, quick run time, and accuracy. To improve the performance of EfficientNetB4, the authors of the paper propose two modifications. The first is an attention mechanism, which not only helps focus on the most relevant portions of the input videos, but also provides insight on what image parts the network viewed as most informative. Figure 3 shows the architecture of this model. The second is a Siamese training strategy, which helps to learn more about the data by comparing different examples. Their model was evaluated on the Face Forensics (FF++) and DFDC datasets. It performed well on the FF++ with an area under the curve (AUC) of 94.44%, but it was 87.82% for the DFDC. Their proposed method is efficient in terms of computational complexity, works on multiple datasets, and outperforms existing methods.

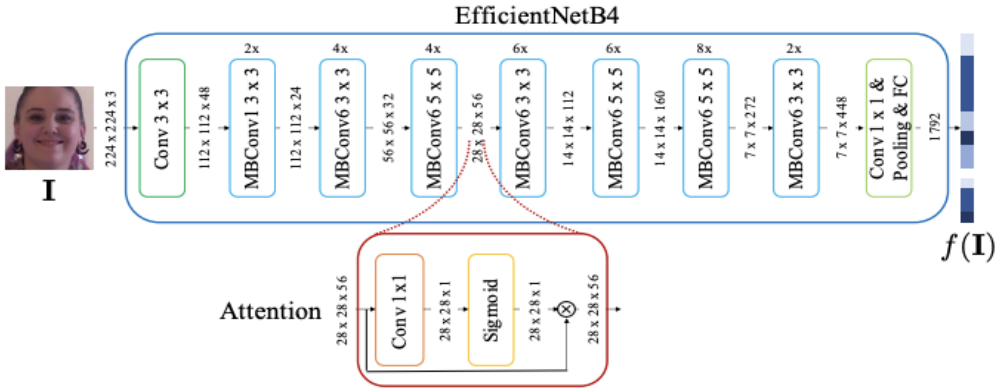

Figure 3: Architecture of EfficientNetB4Att (image from [3]).

Barni et al. [2] discuss a method to detect if an image is created using a GAN. The method proposed two deep learning algorithms, the CoNet and the Cross-CoNet. The CoNet analyzes the co-occurrence matrices of the color channels (red, blue, and green) in an image to distinguish between the real and GAN-generated images. The scheme for this can be viewed in Figure 4. In contrast, the Cross-CoNet modifies the CoNet by taking the relationships *between* the color bands into account. This allows it to compute the color and spatial co-occurrences on each color band separately. These models performed well on the StyleGAN2 dataset, with a CoNet accuracy of 98.15% and a Cross-CoNet accuracy of 99.70%. The main advantage of the Cross-CoNet appeared to be increased robustness against post-processing. In machine learning, "robustness" refers to a model's ability to maintain its performance even when the input data is modified. This points to better performance of the model in real-world applications. The authors suggest future work should attempt to use intentional attacks to confuse their model as well as expose it to unknown datasets.

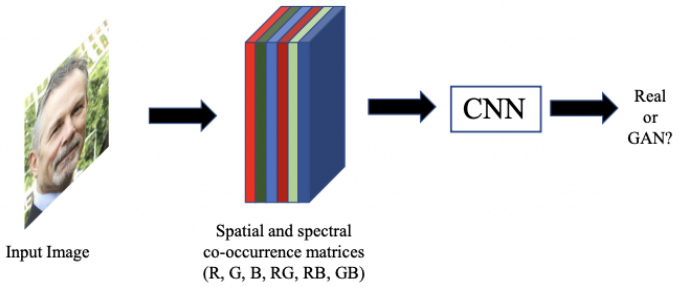

Figure 4: Scheme of CNN based detectors (CoNet and Cross-CoNet) (image from [2]).

A slightly different approach is used by Tariq et al. in [31] where they introduced a convolutional long short-term memory (LSTM) based residual network (CLRNet). A LSTM model is a is a type of recurrent neural network (RNN) that is useful for processing and predicting sequences of data. It is able to selectively keep or discard information from previous time steps, which allows it to remember information and use it to make accurate predictions. Tariq et al. wanted to capture both temporal data (which can be accomplished by LSTM) and spatial data (which can be accomplished by CNN), so they initially tried stacking these models, but this proved unstable and did not transfer learning. So, they turned to Convolutional LSTM cells, which do the same actions with more stability. From here, they developed three training strategies including single domain learning, merge learning, and transfer learning. Single learning trains the detector on a single dataset, merge learning trains the detector on a combined dataset with all known generation methods, and transfer learning trains the detector on one dataset and then learns new domains from a sample of a separate dataset. Finally, Tariq et al. discuss different defense strategies to protect against attacks. The authors did not compute accuracy scores for their model against any datasets but instead computed F1-scores for different attack styles. They found that CLRNet was the best performer across the board with a 98.61% average for a single domain, 87.58% for merge learning, and 97.57% for transfer learning, and it

did generalize well for the open-domain attack with a score of 93.86%. Some limitations or suggestions for future work in this research are utilizing different levels of video compression and experimenting with talking videos [31].

Overall, previous methods show impressive results in terms of improving upon previous models and exploring new ways to detect deepfakes. The literature reviewed in this study has demonstrated that there are various approaches that can be employed for deepfake detection, including the use of advanced deep learning techniques such as CNNs and LSTMs. Additionally, researchers have experimented with a variety of datasets and training strategies, which have resulted in improved performance in models and better generalization across various attacks. Despite this progress, there are still many challenges. One of the main challenges is the constant evolution of deepfake techniques, which requires researchers to continuously adapt and update their detection methods. In addition, the lack of standardized datasets and evaluation metrics adds to the difficulty of comparing results across different studies.

In conclusion, deepfake detection is an active area of research requiring ongoing exploring and experimenting. The results of this literature review demonstrate promising approaches to detect deepfakes. However, there is still more that can be done to improve the accuracy and generalizability of deepfake models. In addition, it is necessary to have a complete understanding of the range of methods used for deepfake detection. Thus, instead of creating a new advanced model, this research focuses on better understanding a lesser-known dataset using basic machine learning techniques.

Chapter 3

Dataset

For this project, the WildDeepfake dataset was carefully selected because it provides a rich source of authentic examples that offer a broad spectrum of quality and angles [34]. The dataset has a challenging reputation and has not been used in many earlier studies. It is comprised of 7,314 face sequences derived from 707 videos. Of these, 3,805 images are real and 3,509 are fake. All of them were sourced from unknown internet locations with no information on creation methods. While the WildDeepfake dataset may be smaller in comparison to some popular sets, like DFDC, its real-world examples make it an invaluable resource for this project. Additionally, the dataset's diversity, various levels of quality, angles, scenes, backgrounds, lighting conditions, resolutions, compression rates, and unknown deepfake methods produce significant difficulties for deepfake detection. For these reasons, it is even more necessary to test these difficulties using a variety of models.

The images within the WildDeepfake dataset consisted of snapshots of faces captured at different frames of a video. Zi et al. [34] used the MTCNN face detector to identify the face regions in each video frame and an ImageNet-pre-trained MobileNetV2 network to extract the face features. Finally, facial landmark extraction by the dlib landmark detector aligned all the faces in a face sequence and created images that were all the same size. The images included a variety of subjects, demographics, lighting, angles, positions, and more. Many of the images were highly convincing and difficult to distinguish by a human. An example of a fake image and a real image from this dataset are included in Figure 4. Because the images were so convincing, it was exciting to see how well a machine would perform on identification.

Figure 5: An example of fake and real images from the WildDeepfake dataset.

Acquiring the WildDeepfake dataset was a challenging process, as it is not publicly available. Nevertheless, by reaching out to the creators directly, access was eventually granted. The images were shared via a Google Drive link, but the task of organizing them was not a simple one. The images were stored in .tar files, which required downloading and unzipping. Moreover, the images were organized within multiple folders, making it a time-consuming task to extract the data. As an example of what one package contained, one .tar file first needed to be downloaded to the computer. It contained 10-30 folders, which each led to a set of nested folders, containing 1-30 folders with hundreds of images. To reorganize the data, the images were moved from the separate nested folders into folders labeled "fake-data" and "real-data." This was a tedious and meticulous process requiring careful organization. In total, the images took up 20.38 GB of storage, with over 1,300 folders containing hundreds of thousands of images.

After the images were sorted, the data was organized. To ensure clean data, images were originally scanned to ensure no filling-in of mixed values or deleting images needed to occur. The format was suitable for this project. In order to not create erroneous values, the code opened the folder location inputted and only read files ending with ".png" to guarantee only images were input into the data table. A function was created to open the images, attach their labels, and extract all the features. It took in the path of the folder containing all the fake or real data and the

label attached to the images in the folder, “fake” or “real,” in order to assign all the images and create a table that the models can understand and train on. The code, which can be viewed in Appendix A.1, functioned like this:

1. Create seven empty lists for the six features and one label.
2. For every subfolder in the input folder of data:
    2a. Go into the subfolder and find files ending with “.png”
    2b. Read in the image
    2c. Extract all the features using the extract_features function
    2d. Add each feature to their own list
    2e. Add the label input (fake/real) for each image
3. Return the lists of all the features and labels.

In addition to this, the data was again scanned to ensure no outliers or erroneous values. To preprocess the data, after reading each file, six features were extracted and added to individual lists. The function “extract_features” is called in step 2c of the data organization script, and it can be viewed in Appendix A.2. Using an image as input, the feature extraction process is:

1. The image is put into grayscale, as some of the features require that filter
2. Each feature is extracted through various manipulations of the image
3. A dictionary of features is created to store each feature and its value
4. The dictionary of features is returned, so it can be assigned in the other function.

More information about each of the extracted features can be found in the next chapter. The open and extract function was called by inserting a path containing all the fake data and the label “fake.” Then, the same was done with the real data. Finally, a dataframe, which is similar to

a spreadsheet, was created to combine the fake and real data and all of their features. After processing the images, the final dataframe contained 84,980 different rows of image data. An example of the dataframe is shown in Table 1. This ensured the data was in an appropriate format to train and test the machine learning models.

Table 1: Example features in the dataframe

Entropy	**Wrapped**	**Noise**	**Blur**	**Keypoints**	**Blobs**	**Label**
7.636494	6.221255	0.077251	151.995305	5	0	fake
6.966655	6.203553	0.072950	49.895874	0	2	real

Chapter 4

Methodology

This chapter covers the methodology used to build the machine learning models. After describing the computing resources necessary for this work, the first half presents the feature extraction process used to create a csv table from the image data. The second half reviews each of the machine learning models chosen and the features that were most effective.

4.1 Computing and Software Resources

For computing resources, Minnesota State University, Mankato's lab machines had 32GB RAM with an Intel Core i7-7700K CPU. They also had NVIDIA GeForce GTX 980 GPUs with 4GB of memory. For software resources, PyCharm with Python version 3.8 was used. All libraries were included. The models were all created with scikit-learn library's resources. This includes sklearn.ensemble.RandomForestClassifier, sklearn.neighbors.KNeighborsClassifier, and sklearn.svm.SVC [29].

4.2 Feature Extraction

The feature selection process involved some trial and error. After reviewing previous literature, the best features to extract and simple extraction methods were not readily apparent. Rather, many previous models used advanced pre-made techniques, such as ResNet, MesoNet, and Xception models, for extracting both images and features [17]. These approaches use deep learning algorithms to extract only the frames with face data from a longer video and then perform extraction on those faces. However, since the dataset used in this project was already a collection of facial images, these advanced techniques were not necessary

Despite having prior knowledge of how to set up machine learning models, a problem still arose: the need to extract features from the dataset. In order for the machine learning models to work, the data must be formatted in tables. Therefore, the goal was to create a table of usable features from the images so that the models could identify patterns and make accurate predictions. In addition, all the features used were numerical, with some continuous and some discrete values.

To start, a basic decision tree model was created as a framework to verify the model development process would work and the machine could classify images. This model acted as a blueprint, so when features were extracted, they could immediately go into a model to assure correct formatting and results. From here, the plan was to extract a few features to verify the model could classify the image data. Initially, the first features found were selected and extracted from a subset of images to try on the model for training and recognition. However, compatibility issues arose due to differences in data formats. For example, one feature that was thought to be beneficial was the Histogram of Oriented Gradients, which can be used to detect objects within an image. While this feature could be extracted from all of the images, it could not be used effectively to train the basic models used in this work. This was because the output was a long-dimensional array that could not be interpreted by a decision tree model, which requires individual features.

To ensure that the machine learning model training would work, a simpler feature extraction code was implemented as a proof-of-concept. The code extracted the mean and standard deviation of the red, blue, and green colors from the facial images, which would generate single float values that the models could use to make decisions. After testing the new code, it was confirmed that the models were built correctly. However, detection with this model

did not return a good accuracy score, so in order for the models to be useful, better features must first be extracted.

To identify useful features to extract and to ensure correct formatting, an approach was taken where a singular image was selected and tested with the potential features offered by the scikit-image library [28]. This library was chosen due to previous exposure in classes and exceptional documentation. By examining the various options, reviewing examples, and analyzing the code, features that appeared useful were selected. Initially, ten numerical features were picked based on their potential for value. The resulting model's accuracy greatly improved when compared to previous tests done on just color data, which was a positive indication that progress was being made.

To better refine the chosen features, the models were evaluated by systematically removing one feature at a time and comparing the accuracy. This approach helped determine which features were contributing to better accuracy and which were not. For instance, the features initially selected, "Dense Daisy," a local image descriptor based on gradient orientation histograms, which allows for fast dense extraction of features, and “Local Maxima,” a function used to find the coordinates of peaks in an image, proved to not affect the accuracy while testing. After several rounds of feature selection, a final set of six features was extracted from the images. These features included entropy, phase unwrapping, noise, key points, blur, and blobs. In Table 2, the features are compared based on how much accuracy decreased when the feature was removed from each model. The numbers in Table 2 represent accuracy from a model trained on all six features minus accuracy from a model trained on the remaining five features. For example, when entropy was removed from random forest, it lost 1.64% accuracy and had a new accuracy score of 97.21%. Thus, bigger numbers represent a more valuable feature for the

model. Blur proved to be the most valuable feature for each model, as the drop in accuracy was highest in each model when it was removed. Entropy was the second most valuable across the board. Noise and phase unwrapping has the smallest drops in accuracy when they were removed. Still, every feature proved to be helpful in detecting deepfakes. In the following paragraphs, each of these features will be examined in more detail.

Table 2: The amount of accuracy decreased after removing the features from each models. Bigger numbers represent a more valuable feature for the model.

Feature	Random Forest	KNN	SVM
Entropy	1.64	5.44	3.46
Phase Unwrapping	0.16	0.84	0.82
Noise	0.01	0.34	3.78
Keypoints	0.23	0.91	1.44
Blur	**1.73**	**6.36**	**5.22**
Blobs	0.42	1.92	3.65

4.2.1 Final Features

Entropy is related to the complexity of its surroundings and can measure subtle differences in gray-level distributions. A visualization of entropy is included in Figure 5. Using the function “skimage.measure.shannon_entropy(img),” the Shannon entropy is computed with $S = -\sum(p(k) * \log(p(k)))$. In this case, p(k) is the frequency/probability of pixels with the value k. This returns a continuous float value for entropy, which can easily be interpreted by machine learning models [28].

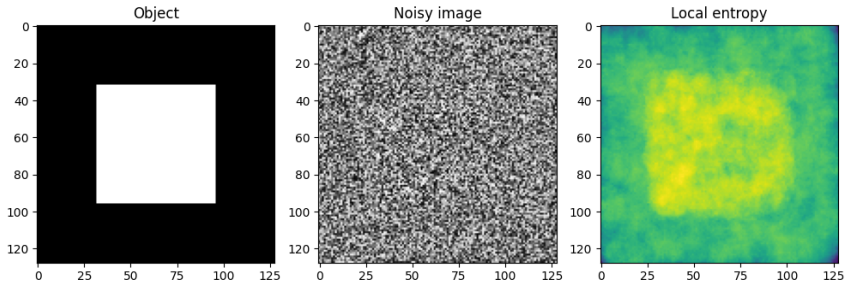

Figure 6: A visualization of Entropy, provided by scikit-image library [28].

Phase unwrapping is used to recover the underlying, unwrapped signal of an image that can only be observed modulo 2π. This provides quantitative information from the image, which is necessary for the models. The unwrapping process, as shown in Figure 6, returns a long array of values in the image, so the range of values was extracted by taking the maximum subtracted by the minimum to receive a continuous float that the models could interpret. The hope was that the range would provide more information about the image details and would vary from real to fake images [28].

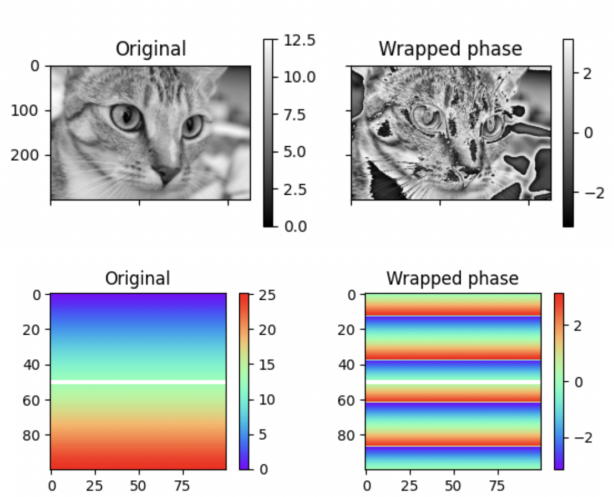

Figure 7: A visualization of phase unwrapping images, provided by scikit-image library [28].

The “noise” feature actually came from “non-local means denoising for preserving textures” according to scikit-image [28]. In Figure 7, the astronaut image is denoised using the non-local means filter. The non-local means algorithm can replace the value of a pixel with an average from the selection of other pixel values. This allows textures to be restored. However, instead of denoising the image, the standard deviation of the noise was measured using the estimate_sigma function. It was hypothesized that highly variable noise values would correlate to false images, whereas less variability would correlate to real images. This provided a continuous float value to help train and test the machine learning models [28].

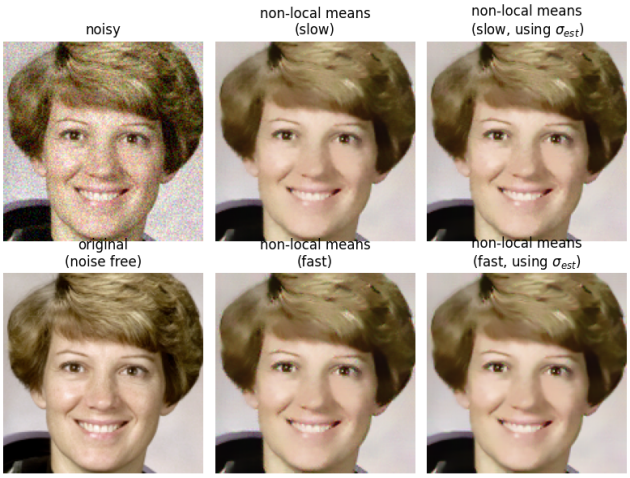

Figure 8: A visualization of noise and denoising, provided by the scikit-image library [28].

The keypoints feature uses a scale-invariant center-surround detector (CENSURE). Scikit-image library claims it can outperform other detectors in image registration and visual odometry [28]. The CENSURE detector provides a list of keypoints from details in the image, from which the length was found to have a value our models can use. This provided a discrete value to train the machine learning models. The hope was that the number of keypoints would

differ in fake images compared to real ones. Keypoints can be viewed as red dots in Figure 8 [28].

Figure 9: A visualization of keypoint detection shown by the red dots, provided by scikit-image library [28].

Another feature extracted was the estimated strength of the blur. Figure 9 shows a blurred image. After applying a filter to the image, the average of the image's blur information was calculated. This provided a continuous value to train the models. Then, it was added to the list of extracted features. It was hypothesized that fake images would return a higher blur value [28].

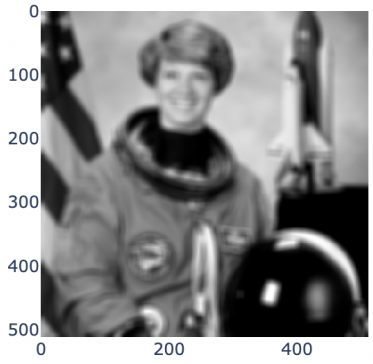

Figure 10: A visualization of blur, provided by scikit-image library [28].

Finally, the feature of “blobs” was selected. The approach uses the difference of Gaussian (DoG) to find blobs, which are either bright spots on dark or dark spots on bright regions in an image. In Figure 10, blobs are detected of stars in space. A simple, discrete count of the blobs was collected from the images, which were all the same size. This feature was included in the models in hopes that manipulated images would contain a varying amount of blobs [28].

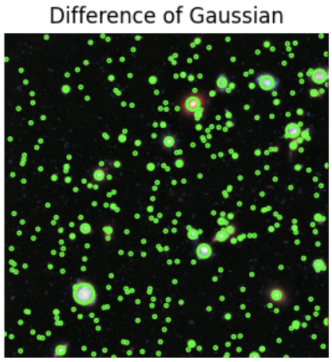

Figure 11: A visualization of blobs using the Difference of Gaussian method, provided by scikit-image library [28].

4.2 Model Selection

After successfully extracting features from the images, these features were used to train the machine learning models. The final models chosen were random forest, k-nearest neighbors (KNN), and support vector machine (SVM). All of the functions used to create the models were sourced from scikit-learn library [29]. These were chosen due to their powerful abilities, popularity, simplicity, and prior work. In addition, they should show a variety of results. Each of these methods uses supervised learning, meaning they were trained with labeled data (“fake” and “real”) to predict the outcomes. Each model will be further explained and examined in the following paragraphs.

Random forest is a powerful ensemble learning algorithm that uses multiple decision trees to classify data. Figure 11 helps visualize the random forest architecture. This approach was

chosen because of its versatile algorithm that can handle complex datasets with many features and types. In the random forest algorithm, individual decision trees are first trained on a subset of the data, in which each datapoint has an equal probability of being selected for a data subset. Each tree in the ensemble can randomly select a subset of observations with full range of the features from the dataset. Then, a final classification is determined by combining and averaging the outputs of all the decision trees, hence a random forest of trees. This approach helps to reduce overfitting and improve generalization performance, making it an ideal choice for deepfake detection [32]. My model used 100 estimators or decision trees as this produced the highest accuracy after running different iterations, as shown in Table 3.

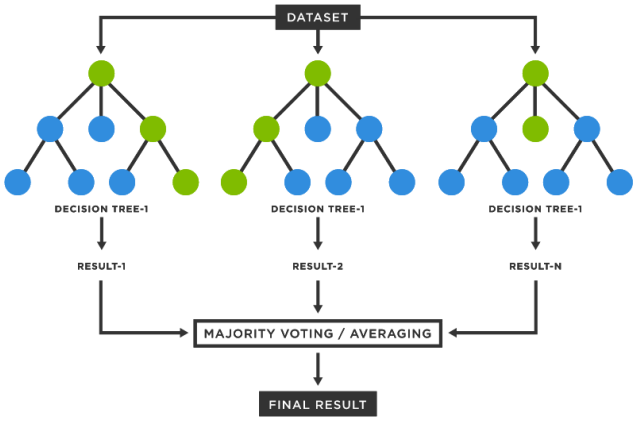

Figure 12: Example of random forest model (image from [32]).

KNN, or K-Nearest Neighbors, is a non-parametric algorithm that classifies data based on its proximity to other data points. Figure 12 provides an example KNN model to show how a point is assigned. It was chosen due to its simplicity, speed, and its ability to be effective in many classification tasks. For KNN classification, all the training points are placed on a plot. Then, new points are classified based on the number of K nearest data points to the test point. The most common label among those neighbors is the predicted label for the test point. This

approach works well when the decision boundary between classes is nonlinear or complex, making it a good choice for deepfake detection [18]. My model used k=5 neighbors, as Table 4 shows that this proved to return the highest accuracy.

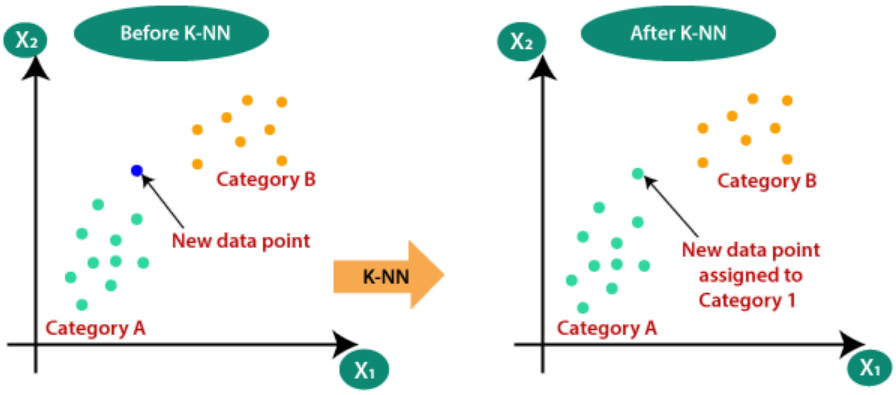

Figure 13: Example of KNN model (image from [18]).

SVM, or Support Vector Machine, is another powerful algorithm that separates data into classes using a hyperplane with the maximum margin. It was chosen because it is known to be effective in handling high-dimensional data. The basic idea behind SVM is to find the hyperplane that separates the data with the maximum margin. The margin is the distance between the hyperplane and the closest data points from each class. Figure 13 aids in viewing the hyperplane for a 2-dimensional classification task. This approach helps to maximize the separation between classes and improve the generalization performance, a goal of this work [27]. My model used the default radial basis function (RBF) kernel, as this best separated the data.

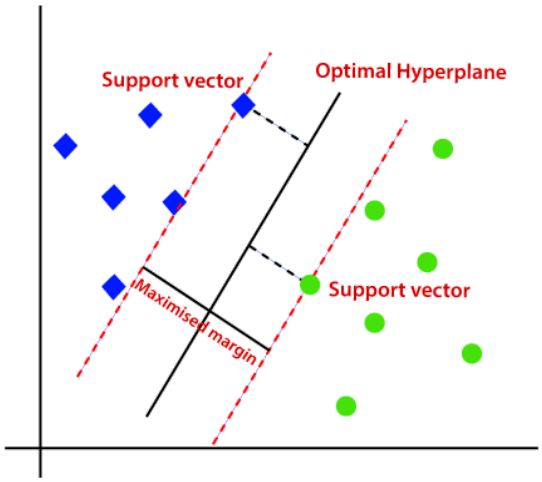

Figure 14: Example of SVM model (image from [27]).

Table 3: Accuracy results from a variety of estimator numbers for the random forest model.

Number of Estimators	Accuracy
10	0.9854672
50	0.9878207
100	**0.9884679**
150	0.9879972
200	0.9878795
250	0.9881149
300	0.9884679

Table 4: Accuracy from different numbers of neighbors used in KNN.

Number of Neighbors	Accuracy
3	0.9309838
5	**0.9443398**
7	0.9311014
9	0.9328666
11	0.9289833

With the simplicity of Python, data is separated before being processed and trained by the model. First, the X and Y variables are assigned. X is set to all of the features, and Y is set to the label: real or fake. Then, the training/testing split is chosen. The models were trained on 80% of the data and tested on the remaining 20% of the data. This is a common split in machine learning and proved to be most effective through a series of tests. It performed better than 70/30, 75/25, and 90/10. The testing and training sets were randomly chosen each run and did not remain consistent through each model. Results from the testing data return immediately after training. Then, the models were compared with evaluation metrics of accuracy, precision, recall, and F1 Score. From here, the model's features were adjusted to favor those that provided higher scores in these categories. This final step helped determine the features mentioned above, such as 100 estimators, 5 neighbors, and the choice of kernel. Figure 14 provides an overall architecture of the entire machine learning process. It includes separating the real and fake images for training, processing the features, structuring the data, inputting the data into the models, and retrieving results. This quick summary provides the overall methodology of this work.

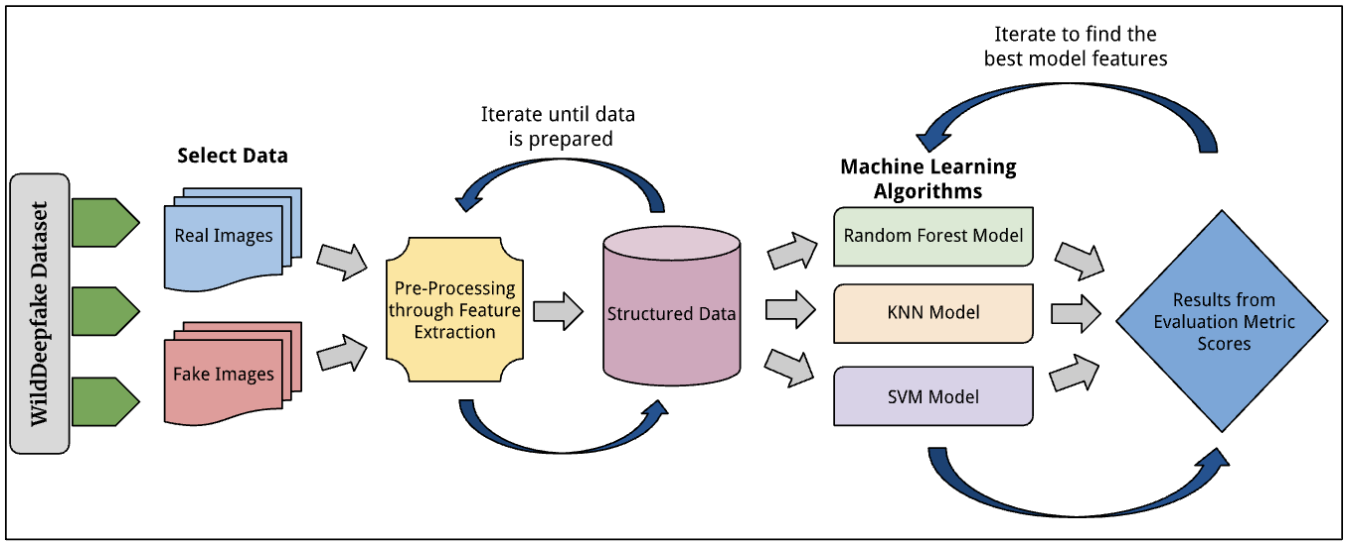

Figure 15: Depiction of the machine learning architecture for this thesis.

Chapter 5

Results

This chapter covers the detection results using three machine learning models. To review, the objective of this thesis is to develop machine learning models for deepfake detection. These models were trained and tested on a dataset of over 7,000 photo samples. The photos were preprocessed to extract image features for analysis. Then, three different machine learning models were evaluated for deepfake detection: random forest, KNN, and SVM. The models were trained on random samples of 80% of the dataset and tested on the remaining 20%. The performance of each model was evaluated using metrics such as accuracy, precision, recall, and F1 score. These are visualized with a confusion matrix and an ROC curve.

For each model, the ROC curve was computed using the roc_curve function from sklearn.metrics [29]. This function takes two inputs: the true binary labels and the target scores, such as probability estimates or confidence values. Then, the positive class label was specified to the "real" images. The outputs of the function include increasing false positive rates, increasing true positive rates, and decreasing decision thresholds. The points (0,0) and (1,0) correspond to the false positive rate (FPR) and true positive rate (TPR) when the threshold is set at the extreme ends. At (0,0), no instances are predicted as positive, while at (1,0), all instances are predicted as positive. Understanding these points helps to interpret the performance of the classifier across different threshold settings [29].

The random forest model achieved an accuracy of 98.85%, precision of 98.90%, recall of 98.75%, and F1 score of 98.82%. Figure 15 provides the confusion matrix, and Figure 16 provides the ROC curve for the random forest model's results.

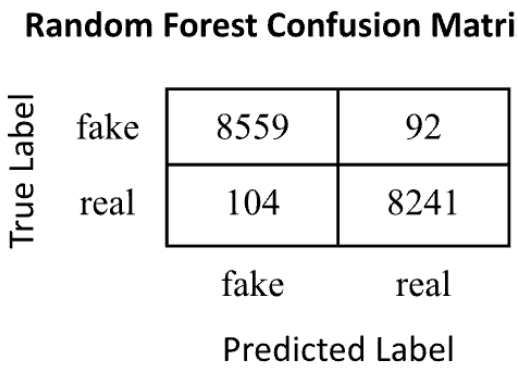

Figure 16: Random forest confusion matrix.

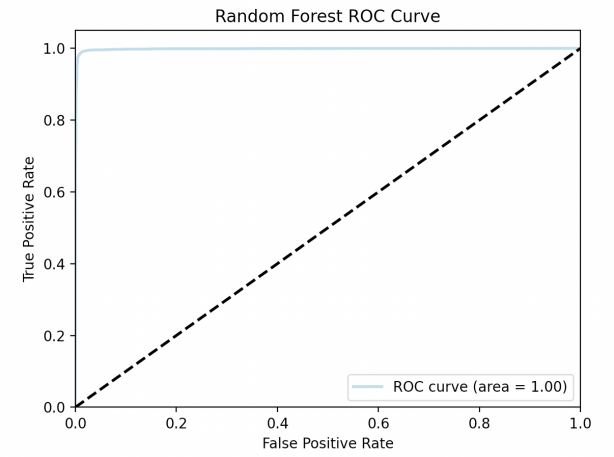

Figure 17: Random forest ROC curve. Top left corner represents a perfect classifier.

The KNN model achieved an accuracy of 94.43%, precision of 94.18%, recall of 94.50%, and F1 score of 94.34%. Figure 17 shows the confusion matrix, and Figure 18 shows the ROC curve for KNN results.

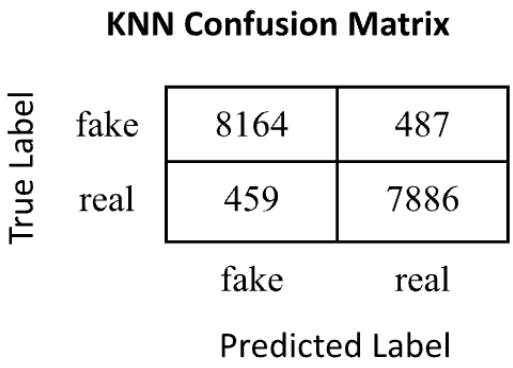

Figure 18: KNN confusion matrix.

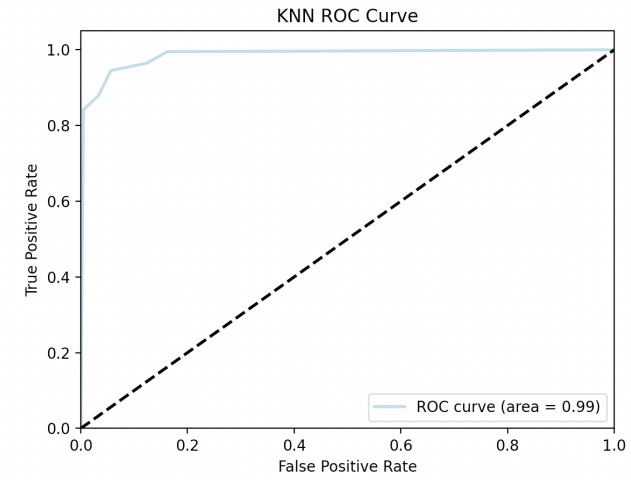

Figure 19: KNN ROC curve. Top left corner represents a perfect classifier.

The SVM model achieved an accuracy of 84.54%, precision of 84.15%, recall of 84.40%, and F1 score of 84.27%. Figure 19 highlights the confusion matrix, and Figure 20 gives an ROC curve for the SVM model's results. Additionally, Table 5 provides a comparison for every evaluation metric for each of the models.

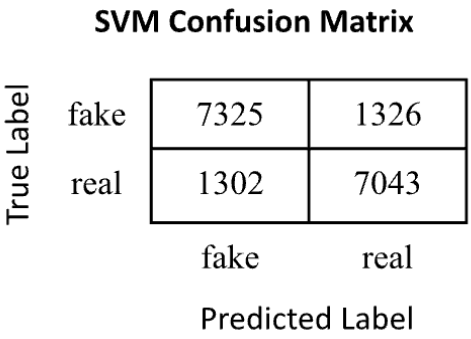

Figure 20: SVM confusion matrix

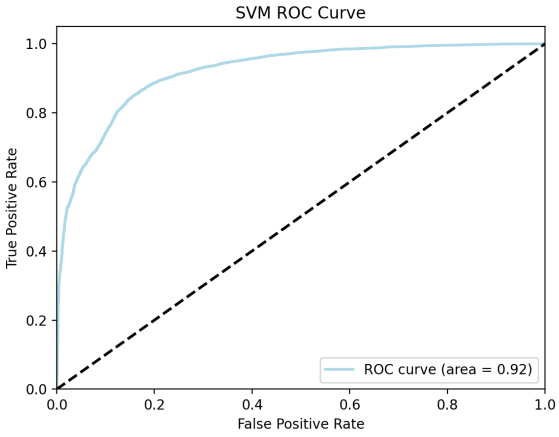

Figure 21: SVM ROC curve. Top left corner represents a perfect classifier.

Table 5: Results from each model for each evaluation metric are compared.

	Random Forest	**KNN**	**SVM**
Accuracy	98.85%	94.43%	84.54%
Precision	98.90%	94.18%	84.15%
Recall	98.75%	94.50%	84.40%
F1 Score	98.82%	94.34%	84.27%

Chapter 6

Analysis

Interpreting the results shown in chapter 5, the models performed very well at the task of deepfake detection. It should first be noted that all three models achieved relatively high accuracy rates in detecting deepfakes. These accuracies are competitive with other datasets that have been claimed to be easier to detect. In addition, they show improvements compared to state-of-the-art models on this specific dataset. As shown in Tables 6 and 7, these models performed better than previous models that use more complex methods to analyze this dataset.

Table 6: Comparison table of model accuracy on the deepfake dataset. Paper title, dataset, method, and results in terms of accuracy are provided.

Title	Dataset	Method	Accuracy
FedForgery: Generalized Face Forgery Detection with Residual Federated Learning [22]	WildDeepfake	FedForgery	68.03%
Improved Xception with Dual Attention Mechanism and Feature Fusion for Face Forgery Detection [21]	WildDeepfake	Dual Attention Mech Xception	83.32%
Hierarchical Forgery Classifier On Multi-modality Face Forgery Clues [11]	WildDeepfake	HFC-MFFD	86.84%
Spatiotemporal Inconsistency Learning for DeepFake Video Detection [14]	WildDeepfake	STIL	84.12%
Exploiting Fine-grained Face Forgery Clues via Progressive Enhancement Learning [13]	WildDeepfake	Progressive Enhancement Learning	84.14%
Fighting Deepfake by Exposing the Convolutional Traces on Images [15]	WildDeepfake	RECCE	83.25%
Identity Mappings in Deep Residual Networks [16]	WildDeepfake	ResNetV2-50	63.99%
MesoNet: a Compact Facial Video Forgery Detection Network [1]	WildDeepfake	MesoNet-4	64.47%
Xception: Deep Learning with Depthwise Separable Convolutions [6]	WildDeepfake	XceptionNet	69.25%
Dual Contrastive Learning for General Face Forgery Detection [30]	WildDeepfake	DeepfakeMAE	81.80%
WildDeepfake: A Challenging Real-World Dataset for Deepfake Detection [34]	WildDeepfake	ADDNet-2D	76.25%

Table 7: Results from current work with random forest, KNN, and SVM.

Dataset	Method	Accuracy
WildDeepfake	Random Forest	98.85%
WildDeepfake	KNN	94.43%
WildDeepfake	SVM	84.54%

A comparative analysis of results from the random forest, KNN, and SVM models indicate the random forest model outperforms the other two models in terms of accuracy, precision, recall, and F1 score. This high performance can be attributed to its ability to handle complex datasets with high-dimensional feature spaces and interactions between features. It provides a probability of belonging to a class, which is useful for a variety of classification problems. In contrast, the KNN model performed relatively well due to its simplicity and ability to handle non-linear relationships between the features. However, the SVM model's lower performance can be attributed to the sensitivity to hyperparameters and its potential for overfitting with large numbers of features. In addition, SVM tries to compute a distance to a boundary, so it is best with sparse amounts of linearly separable data. Even with adjusting the kernel for non-linear data, it is not optimized to separate large amounts of complex data fully.

Overall, the results highlight that the choice of model significantly affects classification performance. The random forest model's ability to handle complex data and the interactions between features make it a viable option for many classification tasks. However, models like KNN may be more appropriate for datasets with fewer features and non-linear structures. The SVM model can also be effective with small, linearly separable datasets. Still, the results suggest that all of the models were proficient in detecting deepfakes. Though more improvements can be made, the results are impressive and improved compared to previously researched methods.

Chapter 7

Conclusion

The machine learning models were successful in detecting deepfakes. With competitive accuracy and other evaluation metrics, they showed strength as a viable model for deepfake detection. In this chapter, it is necessary to understand the limitations faced by deepfake detection and these models. In addition, suggestions for future research are discussed.

7.1 Limitations

According to Juefei-Xu et al., the main limitations faced by models are the ability to generalize to unseen synthesized techniques, remain robust against attacks, and provide explainable detection results [17]. Many models struggle when exposed to real-world deepfakes, especially those with unknown generation techniques.

One of the main limitations specific to this study was that only one dataset was used. While the WildDeepfake dataset used was intentionally selected to represent a diverse range of real-world deepfake techniques, more datasets would help verify the model's reproducibility.

Another limitation is the number of machine learning methods used. Although the three models used in this study were selected based on their popularity and effectiveness in classification tasks, there are many other machine learning methods that could be explored to improve the accuracy of deepfake detection.

Moreover, the study was constrained by time limitations which prevented a more extensive exploration of different models and datasets. While efforts were made to optimize the models and datasets used, the results of the study may not be generalizable to all deepfake scenarios. In the future work section, further exploration is discussed.

Despite these limitations, machine learning methods have shown great potential in detecting deepfakes. Compared with the common deep learning methods used, these basic machine learning models are generally easier to interpret and require fewer computational resources. By understanding the limitations of the current study and continuing to explore and refine machine learning methods, deepfake detection will continue to advance and help to mitigate the negative consequences of this emerging technology.

7.2 Future Work

While this work has demonstrated promising results in detecting deepfakes on a high-quality dataset, there is still much more to be explored in this area. To further advance the field, it is crucial to address the current limitations and explore areas of future research.

One area that requires attention is the application of machine learning methods to other datasets. For example, the DFDC dataset could be used since it is the largest dataset and contains a variety of actors, qualities, photos, videos, and generation methods. Cross-dataset evaluation could be conducted by training the model on one dataset and testing it against another, which will help ensure the robustness and generalization of the models. Using the most diverse data to train the model will result in a model best equipped to detect real deepfakes. Overall, continuing to experiment with new models, parameters, and a variety of data is vital since there are many factors that can affect model performance.

In addition to exploring new datasets, more time and exploration could be conducted on feature extraction. The scikit-image library has hundreds of features, providing a wide range of possibilities for extracting information from images. These features should be further analyzed and understood to better interpret the value of these for deepfake detection. Exploring additional aspects of spatial, frequency, biological, and other features will ensure the model is capable of

detecting a variety of deepfakes. Along with this, ablation studies could be conducted by removing one feature at a time to test for the highest accuracy. Though this occurred with a small number of features in this work, it should be conducted on a larger scale. This will ensure robustness against a variety of generation methods and produce improved models.

It is crucial to keep up with research efforts as fake content proliferates on the internet. With the rise in AI-generated media, such as images, videos, and artwork, it could soon be impossible to discern the truth from anything viewed online. With continued research, we can, hopefully, remain at the forefront of AI development and continue to detect fake content effectively. Otherwise, we may be forced to accept an unsettling “reality” in which artificially created content regularly shapes our perceptions, beliefs, and lives.

Overall, it is important to acknowledge the ongoing race between deepfake generation and detection methods. While deepfakes are becoming more convincing, it is crucial not to rely solely on complex methods to detect them. Instead, research should first aim to better understand the models by utilizing simple tactics to detect deepfakes. This research was a step in that direction, but additional steps will be needed to continue the fight against deepfakes and fake content as a whole.

Bibliography

[1] Afchar, Darius, et al. “Mesonet: A Compact Facial Video Forgery Detection Network.” *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, 4 Sept. 2018, arxiv.org/abs/1809.00888.

[2] Barni, Mauro, et al. “CNN Detection of Gan-Generated Face Images Based on Cross-Band Co-Occurrences Analysis.” *IEEE Workshop on Information Forensics and Security*, 2 Oct. 2020, https://arxiv.org/abs/2007.12909.

[3] Bonettini, Nicolò, et al. “Video Face Manipulation Detection through Ensemble of CNNS.” 2020 25th International Conference on Pattern Recognition (ICPR), 16 Apr. 2020, https://arxiv.org/abs/2004.07676.

[4] Brown, Sara. “Machine Learning, Explained.” *MIT Sloan*, 21 Apr. 2021, mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[5] Chesney, Robert and Danielle K. Citron, “Deep Fakes: A looming challenge for privacy, democracy, and national security,” *107 California Law Review (2019)*, 2018. https://scholarship.law.bu.edu/faculty_scholarship/640

[6] Chollet, François. “Xception: Deep Learning with Depthwise Separable Convolutions.” *CVPR*, 4 Apr. 2017, arxiv.org/abs/1610.02357.

[7] “Cool, but Scary… Deepfakes Are Here.” *Digital, Intercultural and Real-Life Marketing*, 5 Nov. 2019, https://interculturaltalk.com/2019/11/05/cool-but-scarydeepfakes-are-here/.

[8] “Deep Learning vs. Machine Learning: Beginner's Guide.” *Coursera*, https://www.coursera.org/articles/ai-vs-deep-learning-vs-machine-learning-beginners-guide.

[9] “Deepfake Detection Challenge.” *Kaggle*, www.kaggle.com/competitions/deepfake-detection-challenge. Accessed 16 May 2023.

[10] Dolhansky, Brian, et al. “The Deepfake Detection Challenge (DFDC) Dataset.” *Facebook AI*, 28 Oct. 2020, https://arxiv.org/abs/2006.07397.

[11] Dufour, Nick, and Andrew Gully. “Contributing Data to Deepfake Detection Research.” *Google AI Blog*, 24 Sept. 2019, ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html.

[12] *Google Scholar*, scholar.google.com/.

[13] Gu, Qiqi, et al. “Exploiting Fine-Grained Face Forgery Clues via Progressive Enhancement Learning.” *Proceedings of the AAAI Conference on Artificial Intelligence*, 28 Dec. 2021, https://arxiv.org/abs/2112.13977.

[14] Gu, Zhihao, et al. “Spatiotemporal Inconsistency Learning for Deepfake Video Detection.” *Proceedings of the 29th ACM International Conference on Multimedia (MM ’21),* 11 Oct. 2021, https://arxiv.org/abs/2109.01860v3.

[15] Guarnera, Luca, et al. “Fighting Deepfake by Exposing the Convolutional Traces on Images.” *IEEE Access*, 7 Aug. 2020, https://arxiv.org/abs/2008.04095.

[16] He, Kaiming, et al. “Identity Mappings in Deep Residual Networks.” *NASA/ADS*, ui.adsabs.harvard.edu/abs/2016arXiv160305027H/abstract.

[17] Juefei-Xu, Felix, et al. “Countering Malicious Deepfakes: Survey, Battleground, and Horizon.” *International Journal of Computer Vision*, 23 Mar. 2022, https://arxiv.org/abs/2103.00218v3.

[18] “K-Nearest Neighbor(KNN) Algorithm for Machine Learning .” *JavaTPoint*, https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning.

[19] Korshunov, Pavel, and Sebastien Marcel. “DeepFakes: a New Threat to Face Recognition?

Assessment and Detection." *arXiv: Computer Vision and Pattern Recognition*, 20 Dec 2018. https://arxiv.org/abs/1812.08685.

[20] Li, Yuezun, et al. "Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics." *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 27 Sept. 2019, https://arxiv.org/abs/1909.12962.

[21] Lin, Hao, et al. "Improved Xception with Dual Attention Mechanism and Feature Fusion for Face Forgery Detection." *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, 29 Sept. 2021, https://arxiv.org/abs/2109.14136v1.

[22] Liu, Decheng, et al. "Fedforgery: Generalized Face Forgery Detection with Residual Federated Learning." Conference on Artificial Intelligence, 18 Oct. 2022, https://arxiv.org/abs/2210.09563.

[23] Liu, Decheng, et al. "Hierarchical Forgery Classifier on Multi-Modality Face Forgery Clues." *IEEE*, 30 Dec. 2022, https://arxiv.org/abs/2212.14629.

[24] "Machine Learning: A Quick Introduction and Five Core Steps." *Centric Consulting*, 8 Dec. 2022, https://centricconsulting.com/blog/machine-learning-a-quick-introduction-and-five-core-steps/.

[25] *Papers with Code*, https://paperswithcode.com/.

[26] Rossler, Andreas, et al. "FaceForensics++: Learning to detect manipulated facial images." *IEEE International Conference on Computer Vision (ICCV)*, 2019. https://arxiv.org/abs/1901.08971

[27] Saini, Anshul. "Support Vector Machine(SVM): A Complete Guide for Beginners." *Analytics Vidhya*, 12 Nov. 2021, https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/.

[28] “Scikit-Image’s Image Processing in Python.” *Scikit-Image*, https://scikit-image.org/.

[29] “Scikit-Learn Machine Learning in Python.” *Scikit-Learn*, https://scikit-learn.org/stable/index.html

[30] Sun, Ke, et al. “Dual Contrastive Learning for General Face Forgery Detection.” *Proceedings of the AAAI Conference on Artificial Intelligence*, 27 Dec. 2021, https://arxiv.org/abs/2112.13522.

[31] Tariq, Shahroz, et al. “One Detector to Rule Them All: Towards a General Deepfake Attack Detection Framework.” *WWW '21: The Web Conference 2021*, 1 May 2021, https://arxiv.org/abs/2105.00187v1.

[32] “What Is a Random Forest?” *TIBCO Software*, https://www.tibco.com/reference-center/what-is-a-random-forest.

[33] Yang, Xin, et al. “Exposing Deep Fakes Using Inconsistent Head Poses.” *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019. https://arxiv.org/abs/1811.00661.

[34] Zi, Bojia, et al. “WildDeepfake: A Challenging Real-World Dataset for Deepfake Detection.” *MM '20: The 28th ACM International Conference on Multimedia*, 5 Jan. 2021, https://arxiv.org/abs/2101.01456v1.

Appendix A

Code for Data Processing and Feature Extraction

The code in this appendix organizes the data (A.1) and extracts features from the images (A.2), so that it is prepared to train the machine learning models (B.1, B.2, and B.3).

A.1 Data Organization

This function takes in a folder path to find an image, calls the previous functions, and adds each feature from the dictionary to a list. It also takes in the respective label for each image. It returns lists of all the image's feature data.

```
def get_images_and_labels(folder, label):

    entropy = []
    wrapped = []
    noise = []
    blur = []
    keypoints = []
    blobs = []
    labels = []

    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        if not os.path.isdir(subfolder_path):
            continue
        for filename in os.listdir(subfolder_path):
            if not filename.endswith(".png"):
                continue
            try:
                img = io.imread(os.path.join(subfolder_path, filename))
                features = extract_features(img)
                entropy.append(features['entropy'])
                wrapped.append(features['wrapped'])
                noise.append(features['noise'])
                blur.append(features['blur'])
                keypoints.append(features['keypoints'])
                blobs.append(features['blobs'])
                labels.append(label)
            except Exception as e
```

```
            print("Skipping file", filename)
            print("Error:", e)
    return entropy, wrapped, noise, blur, keypoints, blobs, labels
```

This code calls the previous function to assign all of the fake and real images. Then creates a data frame using the lists of features.

```
# Assigning all the data
fake_entropy, fake_wrapped, fake_noise, fake_blur, fake_keypoints, fake_blobs, fake_labels =
get_images_and_labels(r"C:\Users\Student\DeepfakeResearch\WildDeepfakeData\fake-data", "fake")

real_entropy, real_wrapped, real_noise, real_blur, real_keypoints, real_blobs, real_labels =
get_images_and_labels(r"C:\Users\Student\DeepfakeResearch\WildDeepfakeData\real-data", "real")

# Creating data frame
all_data = pd.DataFrame({
    "Entropy": fake_test_entropy + real_test_entropy,
    "Wrapped": fake_test_wrapped + real_test_wrapped,
    "Noise": fake_test_noise + real_test_noise,
    "Blur": fake_test_blur + real_test_blur,
    "Keypoints": fake_test_keypoints + real_test_keypoints,
    "Blobs": fake_test_blobs + real_test_blobs,
    "Label": fake_test_labels + real_test_labels
})

# Save the data to a csv file
all_data.to_csv(all_data.csv")
print(all_data.head())
```

A.2 Feature Extraction

This function takes in an image, extracts all the features, and adds the data to a dictionary:

```
def extract_features(img):
    gray_img = color.rgb2gray(img)

    # Entropy Feature Extraction
    entropy = skimage.measure.shannon_entropy(img)

    # Wrapped Feature Extraction
    image_wrapped = np.angle(np.exp(1j * img))
    max_val = np.max(image_wrapped)
    min_val = np.min(image_wrapped)
    wrapped_range = max_val - min_val
```

```
# Noise Feature Extraction
astro = img_as_float(img)
astro = astro[30:180, 150:300]
sigma = 0.08
noisy = random_noise(astro, var=sigma ** 2)
sigma_est = np.mean(estimate_sigma(noisy, multichannel=True))

# Blur Feature Extraction
blurred_images = [ndi.uniform_filter(img, size=k) for k in range(2, 32, 2)]
img_stack = np.stack(blurred_images)

# Keypoints Feature Extraction
detector = CENSURE()
detector.detect(gray_img)

# Blob Dog Feature Extraction
blobs_dog = blob_dog(gray_img, max_sigma=1, threshold=.1)


features = {
   'entropy': entropy,
   'wrapped': wrapped_range
   'noise': sigma_est,
   'blur': np.mean(img_stack),
   'keypoints': len(detector.keypoints),
   'blobs': len(blobs_dog)
}
return features
```

Appendix B

Code for Model Training and Testing

This python code is used for splitting the data into training and test sets, creating models for the random forest, KNN, and SVM algorithms, and using these models to make predictions on the test set.

B.1 Random Forest Model

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

# accessing the csv to make the model
my_data = pd.read_csv("all_data.csv")

X = my_data[["Entropy", "Wrapped", "Noise", "Blur", "Keypoints", "Blobs"]]
y = my_data["Label"]

# training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# standardize the range of values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Random Forest classifier
forest_model = RandomForestClassifier(n_estimators=100, random_state=1)
forest_model.fit(X_train, y_train)
forest_y_pred = forest_model.predict(X_test)

# evaluate performance
forest_accuracy = accuracy_score(y_test, forest_y_pred)
forest_precision = precision_score(y_test, forest_y_pred, pos_label='real'
```

```
forest_recall = recall_score(y_test, forest_y_pred, pos_label='real')

forest_f1 = f1_score(y_test, forest_y_pred, pos_label='real')

print("Random Forest Accuracy:", forest_accuracy)
print("Random Forest Precision:", forest_precision)
print("Random Forest Recall:", forest_recall)
print("Random Forest F1 Score:", forest_f1)

# confusion matrix
forest_cm = confusion_matrix(y_test, forest_y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(forest_cm, annot=True, cmap="Blues", fmt='g', xticklabels=['fake', 'real'], yticklabels=['fake', 'real'])
plt.title("Random Forest Confusion Matrix")
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.show()

# ROC curve
forest_probs = forest_model.predict_proba(X_test)
forest_probs = forest_probs[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, forest_probs, pos_label='real')
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='lightblue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

B.2 KNN Model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix, roc_curve,
auc
import matplotlib.pyplot as plt
import seaborn as sns


# accessing the csv to make the model
```

```
my_data = pd.read_csv("all_data.csv")

X = my_data[["Entropy", "Wrapped", "Noise", "Blur", "Keypoints", "Blobs"]]
y = my_data["Label"]

# training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# standardize the range of values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate the performance
knn_accuracy = accuracy_score(y_test, y_pred)
knn_precision = precision_score(y_test, y_pred, pos_label='real')
knn_recall = recall_score(y_test, y_pred, pos_label='real')
knn_f1 = f1_score(y_test, y_pred, pos_label='real')
print("KNN Accuracy:", knn_accuracy)
print("KNN Precision:", knn_precision)
print("KNN Recall:", knn_recall)
print("KNN F1 Score:", knn_f1)

# Confusion matrix
knn_cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(knn_cm, annot=True, cmap="Blues", fmt='g', xticklabels=['fake', 'real'], yticklabels=['fake', 'real'])
plt.title("KNN Confusion Matrix")
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.show()

# ROC curve
knn_probs = knn.predict_proba(X_test)
knn_probs = knn_probs[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, knn_probs, pos_label='real')
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='lightblue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN ROC Curve')
```

```
plt.legend(loc="lower right")
plt.show()
```

B.3 SVM Model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix, roc_curve,
auc
import matplotlib.pyplot as plt
import seaborn as sns

# accessing the csv to make the model
my_data = pd.read_csv("all_data.csv")

X = my_data[["Entropy", "Wrapped", "Noise", "Blur", "Keypoints", "Blobs"]]
y = my_data["Label"]

# training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# standardize the range of values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# linear kernel, since data is linearly separable
svm_model = SVC(kernel='rbf', random_state=1)
svm_model.fit(X_train, y_train)
svm_y_pred = svm_model.predict(X_test)

# Evaluation of performance
svm_accuracy = accuracy_score(y_test, svm_y_pred)
svm_precision = precision_score(y_test, svm_y_pred, pos_label='real')
svm_recall = recall_score(y_test, svm_y_pred, pos_label='real')
svm_f1 = f1_score(y_test, svm_y_pred, pos_label='real')
print("SVM Accuracy:", svm_accuracy)
print("SVM Precision:", svm_precision)
print("SVM Recall:", svm_recall)
print("SVM F1 Score:", svm_f1)

# Confusion matrix
svm_cm = confusion_matrix(y_test, svm_y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(svm_cm, annot=True, cmap="Blues", fmt='g', xticklabels=['fake', 'real'], yticklabels=['fake', 'real'])
```

```
plt.title("SVM Confusion Matrix")
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.show()

# ROC curve
svm_probs = svm_model.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, svm_probs, pos_label='real')
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='lightblue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM ROC Curve')
plt.legend(loc="lower right")
plt.show()
```