Article

# Using Ensemble Convolutional Neural Network to Detect Deepfakes Using Periocular Data

David Johnson [*,†,‡] , Xiaohonh Yuan [†,‡] , Kaushik Roy [†,‡]

Posted Date: 17 February 2023

doi: 10.20944/preprints202302.0299.v1

Keywords: deepfake detection; CNN; deep neural network; computer vision; scale invariant feature transform; histogram of oriented gradients

*Article*

# Using Ensemble Convolutional Neural Network to Detect Deepfakes Using Periocular Data

**David Johnson [1,*,†], Xiaohong Yuan [2,†] and Kaushik Roy [3,†]**

North Carolina Agricultural and Technical State University, 1601 E Market St, Greensboro, North Carolina 27411, USA; xhyuan@ncat.edu (X.Y.); kroy@ncat.edu (K.R.)

* Correspondence: dmjohns8@aggies.ncat.edu; Tel.: +1-706-731-4242

† These authors contributed equally to this work.

**Abstract:** Deepfakes are manipulated or altered images, or video, that are created using deep learning models with high levels of photorealism. The two popular methods of producing a deepfake are based on either convolutional neural networks (CNN), or autoencoders. Deepfakes created using CNN comparatively show higher qualities of realism, yet oftentimes leave artifacts and distortions in the generated media that can be detected using machine learning and deep learning algorithms. In recent years, there has been an influx of periocular image and video data because of the increase usage of face masks. By wearing masks, much of what is used for facial recognition is hidden, leaving only the periocular region visible to an observer. This loss of vital information leads to easier misidentification of media, allowing deepfakes to less likely be identified as fake. In this work, feature extraction methods, such as Scale-Invariant Feature Transform (SIFT), Histogram of Oriented Gradients (HOG), and CNN, are used to train an ensemble deep learning model to detect deepfakes in videos on a frame-by-frame level based on the periocular region. Our proposed model is able to distinguish original and manipulated images with accuracies around 98.9 percent, which is an improvement to previous works by combining SIFT and HOG for deepfake detection in convolutional neural networks.

**Keywords:** deepfake detection; CNN, deep neural network; computer vision; scale invariant feature transform; histogram of oriented gradients

## 1. Introduction

Deepfakes are manipulated images or videos that usually are an attempt to affect the image or reputation of an individual, or deepfakes are used to spread disinformation. There are times when the creation of deepfakes is for good-faith intentions, such as when MIT created a deepfake video of a speech by President Nixon alleging a failed Apollo-11 mission [1], or when American director Jordan Peele impersonated President Obama [2] to present a speech that seemed out of character. The altered, hyper-realistic images and videos are enough to convince people of their legitimacy. Nicolas Cage appears in a collection of videos [3] where the actor appears in movies that he was never cast in. Deepfake creation stems from machine learning methods [4] that have been improved upon to now using advanced deep learning techniques involving convolutional neural networks (CNN) [5], which are deep learning models that consist of convolutional layers. These layers are able to learn patterns, or features, in images based on sub-groupings of pixels within a larger image. These feature detectors are called filters, or kernels, and are "activated" when the layer receives input from an image containing similar pixel structures. This essentially is a form of learning the contents of images, and these filters can be used with other images to detect if they contain similar pixel patterns. Since these filters are learned by the deep learning model, these models then have the ability to recreate images that would activate the filters, thus creating a deepfake [5–7].

In recent years, there has been an increase in image and video data that only shows the periocular region instead of the entire face. This is largely due to the increase in people wearing masks because of the COVID-19 pandemic. The less-revealing faces already inhibit facial inter-personal facial recognition, but this lowered recognition also makes it easier for a deepfake to pass as realistic to a human observer. Deepfake creators that rely on face swapping and mimicry focus on prominent facial features to manipulate, such as the eyes, nose, and mouth [5]. These features along with their expected, artifact-free observation, allow people to discern a manipulated image yet when most of the face is covered or hidden, people have less information in front of them to make that decision. Although deepfake periocular image data can be created, the process to create them still is believed to generate artifacts and inconsistencies within the created images that can be detected not only by human eyes, but also with deep learning models.

Deep learning methods have been used before to detect deepfakes, such as CNN and long short-term memory (LSTM) but there has been a lack of combining various computer vision algorithms with deep learning models. Deep learning models alone tend to perform well, that including other feature extraction methods can incur unneeded overhead without increasing performance. For example, feature extraction using scale-invariant feature transform takes a long time to extract a smaller set of features than compared to histogram of oriented gradients. Both computer vision methods can be used for object and edge detection, although the quality of the extracted features may differ significantly.

Compared to CNN, many of these computer vision methods are able to perform feature extraction at a much faster rate while using fewer resources, although classification performance is poorer. This research intends to combine feature extraction and CNN for deepfake detection, and to compare the performance of these combined models. As of this paper, there have not been significant strides made in the deep learning field for periocular-based deepfake detection. This paper will show that deep learning models using different computer vision feature extraction methods, along with combining these various models, can detect deepfakes at a significantly higher accuracy focusing only on the periocular region. Models designed around smaller smaller amounts of information are useful when only these smaller pools of data are available.

A simple example of a CNN is a model that can distinguish pictures containing cats from pictures containing dogs, or even being able to count the number of cats and dogs in a picture, along with their breed. More robust CNN are able to generate new images containing a cat or dog, to further the example. Common methods of deepfake generation involve cutting and pasting [5], or called swapping [8–10], the face of one person onto another. This swapping leaves artifacts in the produced image that may be noticeable to the human eye but smaller details can be detected by a machine [9,11]. In order to increase realism in the manipulated images, smoothing and brushing techniques are used along the bounding box of the swapped face, with key facial landmarks mapped to match the movement of the actor on the original image. The bounding box has been a focus of research [5,8] in detecting deepfakes since using a mask to swap a face naturally creates edges, making edge detection possible [12]. Alterations within the swapped face are less likely to produce consistent edges since their creation is based on image augmentation [13], as opposed to image masking.

In this paper, the dataset DeepFakeDetection [14,15] is used for performance evaluation. These models may then be used on other datasets from the FaceForensics++ [14] collection to compare their results for a more general evaluation. DeepFakeDetection consists of one thousand unaltered videos, and three thousand manipulated videos based on the original videos. Since the source is video data, and frames will need to be extracted to have image data, a substantially large portion of the dataset would look similar, in that each frame of a video is only slightly different from the neighboring frame in most cases, in some cases, the neighboring frame is the exact same.

The use of CNNs for image, or image-like, learning is widely employed since the convolutional process creates smaller, stacked regions of learnable features while preserving input data. This process is performed within layers called *Convolutional Layers*. Between convolutions, the feature map, or activation map, will become more dense, or deeper, while the two-dimensional nature of the image data at most remains the same, while in many cases, decreases as the number of filters increase across layers. These filters activate when a learned feature is passed through them. This means that we can have many trainable filters, trained on smaller, deeper dimensions of image data to detect features that would indicate if an image is original, unaltered media, or if it has been altered and could be a deepfake.

Along with using convolutional layers, we explore some other methods that can be used to extract features from image data for deepfake classification. These methods are Scale-Invariant Feature Transform [11,16] (SIFT) and Histogram of Oriented Gradients [17] (HOG). SIFT features are useful in that the features, when clustered together, are used to represent objects and biological markers within an image. These objects and markers are able to be mapped across various images, regardless of changes in scale or orientation. This presents a useful tool for object detection that can be used to detect distinguishing features in deepfake images. HOG features, when seen as in image, is more akin to edge detection and contour detection. HOG features represent changes in angle gradients, or slopes, and contrast across pixel data using Gaussian derivatives. When the image data is viewed, it is shown as a collection of angled vectors that outline the various objects in the image.

Ensemble models offer a relatively simple way to include multiple different features as input for the same process. Typically, deep neural networks have a single set of features, or a batch of similar features, for learning. These more complex models allow a single input, such as an image, to be represented by different types of features. In the case of this article, the different type of features are 1) convolutional features, 2) SIFT features, and 3) HOG features. Since these features are extracted using different processes, one set of features may contain unique data that can be used effectively to distinguish original media from deepfakes. In this article, we describe how an ensemble model can be used with various feature extraction tools to accurately detect deepfakes with high performance. We will also explain the construction of the model and their performance on different datasets.

## 2. Proposed Methods

This section will discuss the methods used to build a CNN model for feature extraction and classification, and also the use of machine learning algorithms for feature extraction. This section will also include the performance of the models along with a comparison of their metrics.

### 2.1. Dataset

The primary dataset comes from the *FaceForensics++* [14] collection of deepfakes. The current collection consists of six sets of deepfakes ccreated using different methods of deepfake creation. A few of the datasets within *FaceForensics++* are *FaceSwap*, *Face2Face*, *DeepFakeDetection*, and *Deepfakes*. The DeepFakeDetection [14,15], created by Google and Jigsaw, is used as the experimental dataset. This dataset consists of 1000 original, unaltered videos containing twenty-eight actors. There are also 3000 manipulated videos, called deepfakes. The deepfakes are produced from the original videos using face swapping methods. To balance original and manipulated videos, every third manipulated video per actor is used for every original video per actor. The videos from the datasets are also labeled as either "real" or "fake," along with the extracted frames from each video being labeled based on their respective source. For example, frames extracted from the video named "fake_000.mp4" will have the name "fake_xxxx.jpg," where "xxxx" is an incremental number based on the number of total extracted frames, and the beginning of the file name, such as "fake" corresponds to the frame's label. Since the proposed model is a binary classifier, there are only two target labels for prediction: real and fake. Real would suggest the image is unaltered, unmanipulated, while fake would suggest the image is a deepfake.

### 2.2. Dataset Frame Extraction

Using the *Python* [18] library *OpenCV* [19], videos are loaded and frames saved. Since image data is coming from videos, the majority of frames tend to be nearly identical to the previous and successive frame. Having a large amount of similar images where the key features may not change within each frame can lead to the dataset having increased homogeneity. To mitigate this, a frame is saved every $100ms$, and 100 frames are saved per loaded video. After extracting all of the necessary frames, there is a total of $113,572$ images

### 2.3. Periocular Region Extraction

Face detection and facial landmark localizing was done using the trained deep neural network Multitask Cascaded Neural Network (MTCNN) [20]. By using a trained neural network, we can retain detected faces with a confidence of 99%, and ignore faces with lower confidence. MTCNN also provides the location of detected facial landmarks, such as the left eye, right eye, nose, and mouth corners. Using the locations of the eyes, we create a bounding box with extra padded space to include the areas surrounding the eyes, such as the eyebrows, and top of the nose region. An example of this process can be seen in Figure 1. The returned face by MTCNN is not truly a face, but a dictionary. This dictionary contains the keys $box$, $confidence$, and $keypoints$. The value of $keypoints$ is a dictionary containing coordinates for facial landmark locations $left\_eye$, $right\_eye$, $mouth\_left$, $mouth\_right$, and $nose$, where $box$ is the bounding box around the detected face, $confidence$ is the confidence of the model in the detected face, $left\_eye$ is the eye on the left side from the perspective of the viewer, $mouth\_left$ is the corner of the mouth on the left side from the perspective of the viewer, and $nose$ is the tip of the nose. To prevent having two separate eyes to manage, the pixel locations of the bottom right- and top rightmost pixels can be used with the bottom left- and top leftmost pixels to create a single bounding box containing both eyes. This bounded box is then padded with extra pixels since the landmark locations provided by MTCNN are centered on the eyes. This box is then used to crop the eye region from the face image and save the eye region as an image. The images are then resized to height and width $(20,100)$. After extraction, the image database consists of $113,572$ color images of shape $(20,100,3)$, where exactly half, $(56,786)$, come from original, unaltered videos, and the other half come from the deepfake videos. Sample input images for the model after frame extraction are shown in Figures 2 and 3.

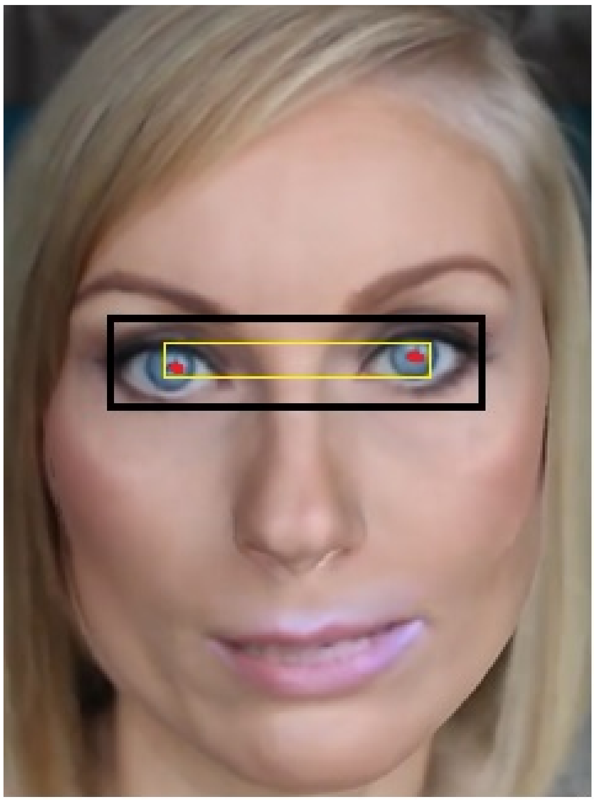


**Figure 1.** Visualization of periocular region extraction. Red: MTCNN landmark location. Yellow: Bounding box based on Red. Black: Padded bounding box based on Yellow.

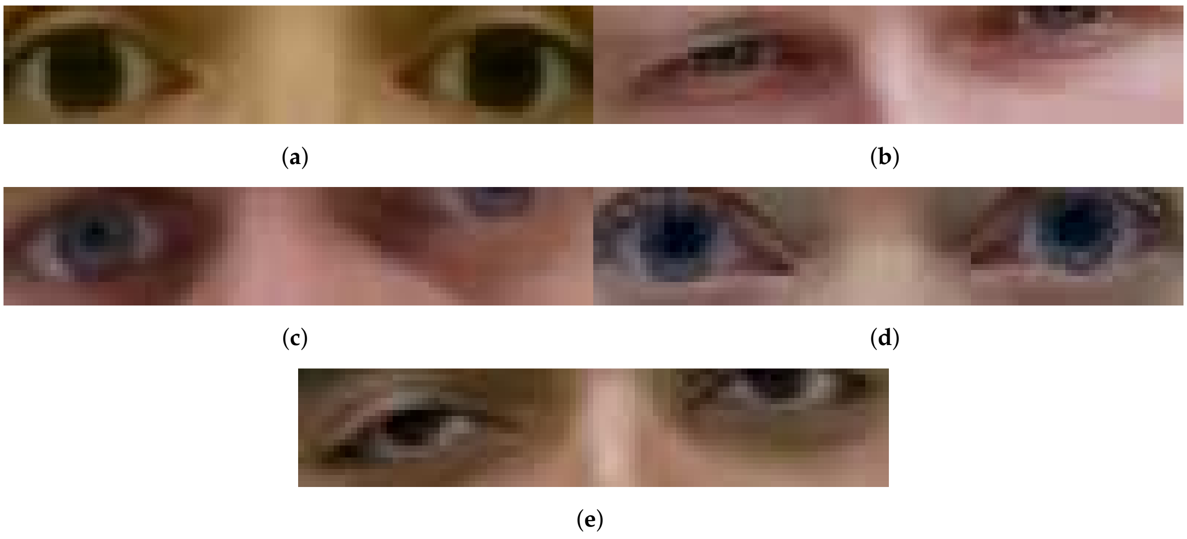


**Figure 2.** Sample original, unaltered periocular images used as source for the *DeepFakeDetection* dataset.

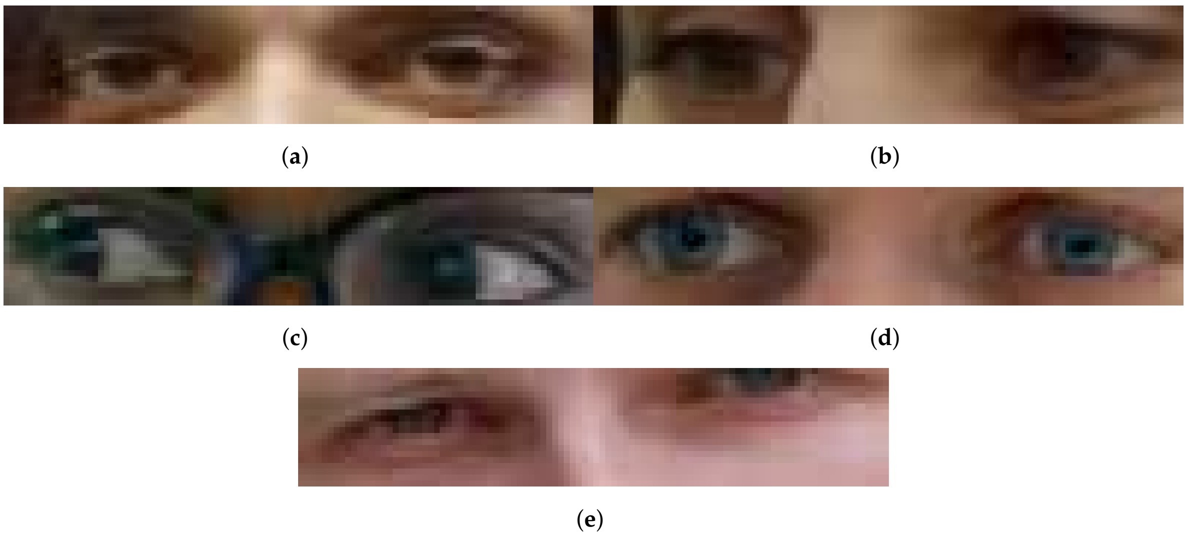


**Figure 3.** Sample altered, deepfake periocular images used as source for the *DeepFakeDetection* dataset.

Before using the data with a model, the input dataset must be split into training and testing sets. The dataset will be split with a ratio of 90/10 to match Table 1, where exactly half of the dataset belongs to the *Real* class and the other half belongs to the *Fake* class:

**Table 1.** Training and Testing split

| | **Training/Test Split** | |
|---|---|---|
| | **90%** | **10%** |
| **Real** | 51107 | 5679 |
| **Fake** | 51107 | 5679 |
| | **Total:** | 113572 |

## 2.4. *Convolutional Neural Network Model*

Convolutional neural networks (CNN) use *Convolutional Layers* to create feature maps of images. These feature maps consist of filters that activate when they "see" a similarly learned feature from the training data in images, such as edges or color, or a pattern. CNN have been a popular choice for object recognition in images and deepfake detection [8,10,21–23] because of their ability to learn features in images and create high dimensionality generalizations from images without sacrificing information from the input images. As images are passed through convolutional layers, a kernel of a fixed size, typically $3x3$, $5x5$, or other sizes, the kernel learns features about pixels and their surrounding data points, which would be other pixels, thus preserving the relationship between pixels [24]. Building a CNN requires building upon convolutional layers. We design a CNN model containing four convolutional "blocks," where each block contains a *Convolutional Layer*, a *MaxPooling Layer*, and a *Dropout Layer*. The first two convolutional layers have 32 filters, and the last two have 64 filters. All convolutional layers use the *RectifiedLinearUnit* (ReLU) activation function and a stride of 3. ReLU improves the generalization ability of convolutional layers with an effect similar to Batch Normalization [25]. Using a dropout of 0.3 provided enough random, turned-off neurons to reduce overfitting. Finally, two *Dense* layers are used, with the first having 128 neurons and relu activation function, and the second, classifying layer having one neuron and the *sigmoid* activation function. Since the labels refer to the classes, and there are only two possible classes for this binary classification problem, we use the *BinaryCrossEntropy* loss function. The equation for calculating the loss function is shown in Equation (1), where $y$ is the binary class value, either 0 or 1, and $p$ is the predicted probability that the predicted element belongs to class 1. The optimizer *Adam* is used with the learning rate set to 0.00001. Figure 4 shows the architecture of the standalone CNN model used for feature extraction and classification.

$$loss = -(y \log(p) + (1 - y) \log(1 - p)) \tag{1}$$

A problem we faced when training and adjusting the model was overfitting. When the model overfits on the dataset, the model learns the features of the training data too well, yet cannot distinguish the class of images in the validation data with as good of a performance as the training data. Our model overfitting on the *DeepFakeDetection* dataset is shown in Figure 5. When the training accuracy is significantly higher than the validation accuracy, and when training loss is signiicantly lower than validation loss, this is a glaring symptom of overfitting.

*Dropout* can be used to be attempt to control overfitting [26]. This is a process performed before hidden layers where neurons are randomly removed, or dropped, along with their connections. This random neuron dropping forces the model to adapt how it learns features from inputs [27]. The number of dropped neurons is determined by the user and is a fraction of the neurons. To be more precise, this is not a definite ratio of neurons to be dropped, this is a calculation performed on each neuron, but given a large enough number of neurons, the total number of dropped neurons should be close to the number set by the model creator. A neuron's dropout chance is simply calculated:

$$1 - p \tag{2}$$

where p is the set dropout value. Usually, this number is below 0.5, which would be roughly half of the neurons in a layer, because the maximum regularization achievable by the regularization parameter $p(1 - p)$ is $p = 0.5$. When $p > 0.5$, more neurons are dropped, while regularization does not increase [26]. Dropout can also affect the graph of training and validation, where validation and testing performance can be better than training. An example of this is shown in Figure 6. Since Dropout is only utilized during training, where only a portion of features are used, all features are used during validation and testing.

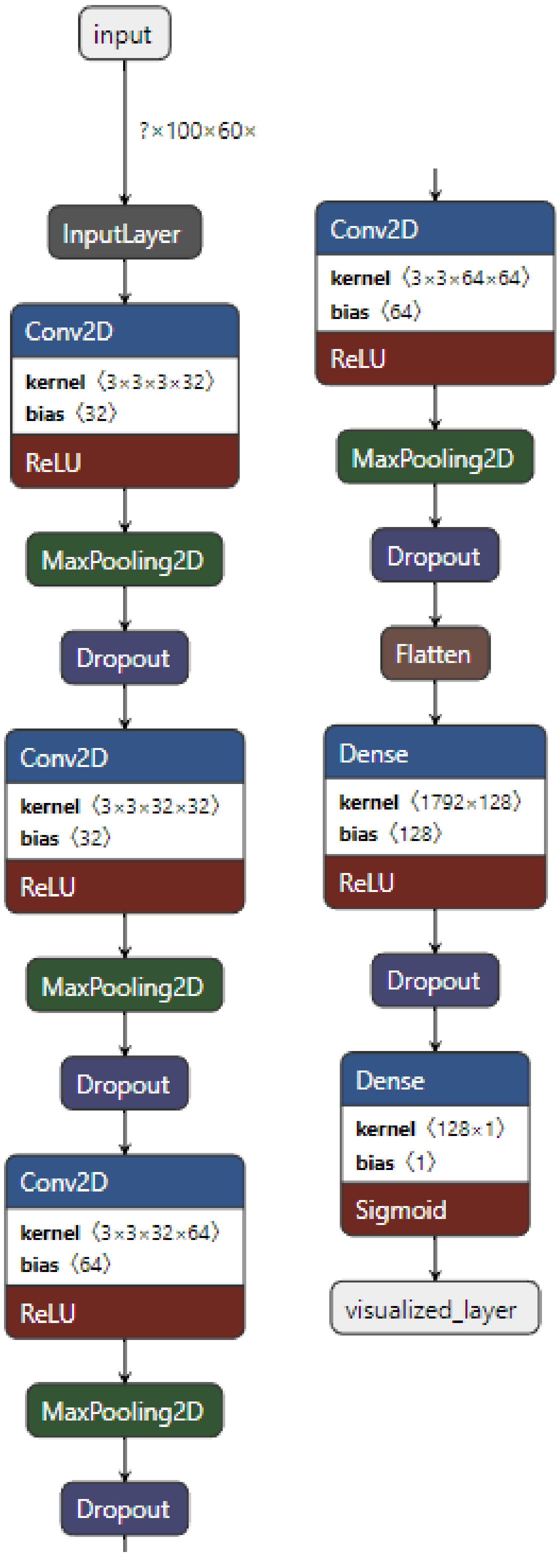


**Figure 4.** The standalone CNN model used to generate metrics for comparison against SIFT and HOG feature extraction methods.

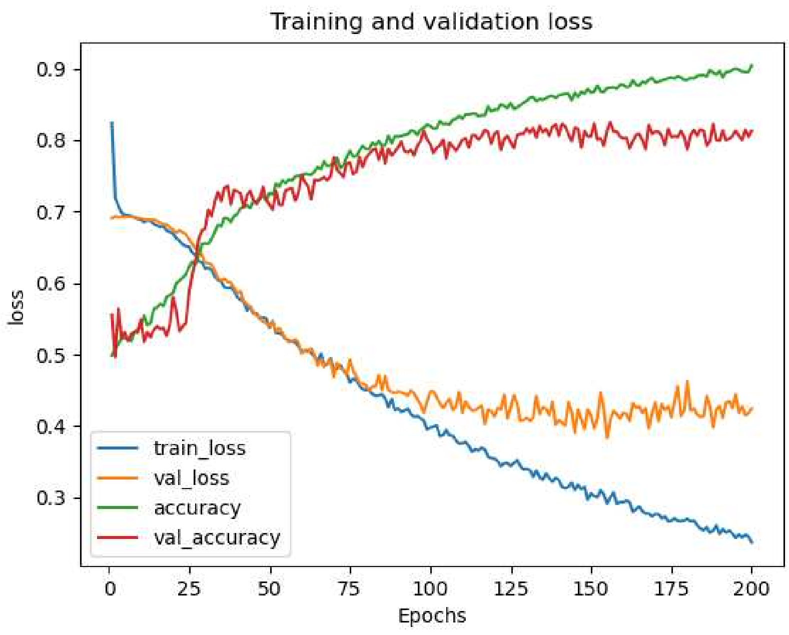


**Figure 5.** An example of our model overfitting on the *DeepFakeDetection* dataset.

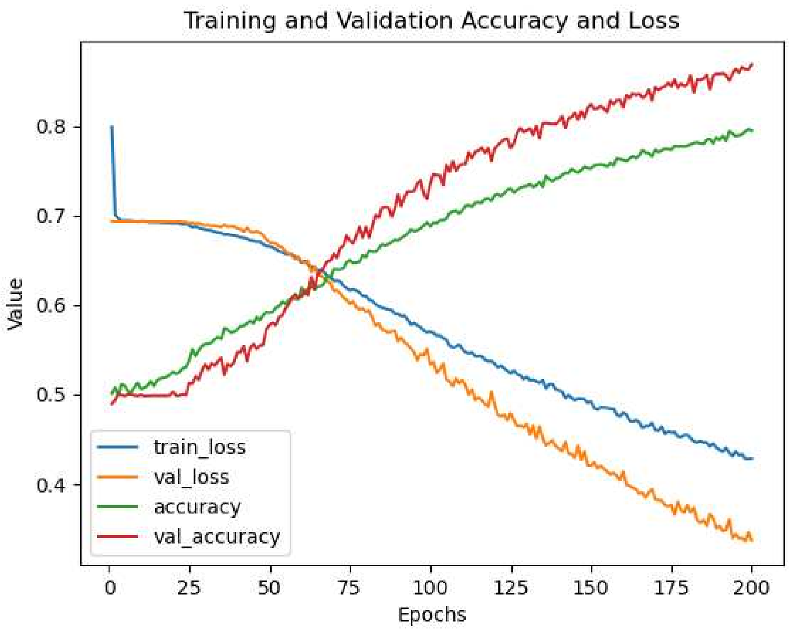


**Figure 6.** An example of our model having significantly better validation performance than training performance on the *DeepFakeDetection* dataset.

Another way to mitigate overfitting involved the use of *BatchNormalization*. Batch normalization is a process performed on a layer that adjusts each "mini batch" against its mean and standard deviation. This will ideally bring each iteration closer to the convergence, where the training and validation metrics of the model are close. Batch normalization is a fairly simply function [28] shown in Equation (3).

$$BN(x) = \gamma * \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta \tag{3}$$

where $\gamma$ and $\beta$ are scale coefficient and scale offset, respectively, $x$ is the input, $\hat{\mu}$ is the mean of the mini batch, and $\hat{\sigma}_B$ is the standard deviation of the mini batch [28].

### *2.5. Histogram of Oriented Gradients Model*

The Histogram of Oriented Gradients (HOG) model uses HOG features along with convolutional layers to classify deepfake images. HOG is a computer vision method originally used to detect upright, human poses. The algorithm utilizes binning, similarly to SIFT, to create a histogram of directed orientation, or gradients which are normalized into blocks, which allow for overlapping keypoints. Compared to SIFT, this is a much simpler process that essentially just creates an image of contours and edge data that can be used as feature data.

*Scikit-image* is an open library that contains the function *hog*, which returns HOG features and the matching HOG image. Sample HOG images that correspond to their matching features are shown in Figure 7. The HOG image is a gradient-slope representation of the original image. The initial research [17] on the HOG algorithm relied on image data of humans standing walking, or in other similar, upright poses - these images were resized to width and height $(64x128)$ since the performance of feature extraction is impacted by the input image dimension ratio. The $1:2$ ratio is not strictly retained, but the orientation is retained, and the ratio loosely retained. Our eye data, which is width and height $(20, 100)$ is rotated 90°clockwise before using it as input for the $hog()$ function. The features are then returned as a $1D$-array along with corresponding images with focus on the gradients. Only the array is used. The feature data is used as input to convolutional layers. We use convolutional blocks without pooling layers since the feature data only represents itself, without much bearing on neighboring data. The HOG model consists of only two convolutional blocks, where the first convolutional layer has 32 filters and the second has 64 filters. We, again, use two final dense layers with 128 neurons in the first layer and one neuron in the final, classifying layer. The HOG features are saved into a *Numpy ndaarray* and saved onto the hard drive for later use with the ensemble model, and also to save preprocessing time during debugging and multiple training iterations with the same data.

(**a**)

(**b**)

**Figure 7.** *Cont.*

(**c**)

(**d**)

(**e**)

(**f**)

(**g**)

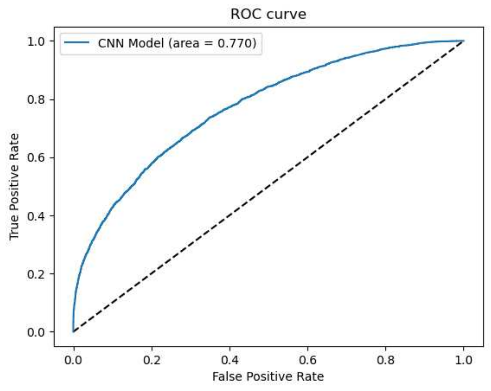
(**h**)

**Figure 7.** Sample HOG eye images. Images (**a**–**d**) are original, real images. Images (**e**–**h**) are deepfake images. Samples from DeepFakeDetection dataset.

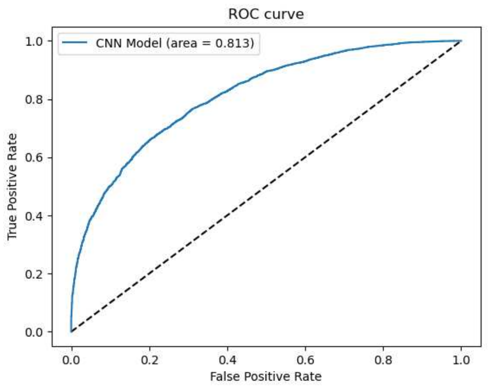
### 2.6. Scale-Invariant Feature Transform Model

The Scale-Invariant Feature Transform (SIFT) model uses SIFT features along with convolutional layers to classify deepfake images. SIFT is a computer vision method used for object- and biological marker detection, object tracking, and edge detection. SIFT uses a scale space to generate keypoints using the Difference of Gaussian. These keypoints are then taken, along with its neighborhood, to create a gradient magnitude and direction. From the magnitude and direction, the SIFT function then places these keypoints into bins, forming a histogram of gradients, similar to Histogram of Oriented Gradients. We now have a location, scale, and orientation for the keypoints, they can be detected across images regardless of changes in the location, size, or rotation. This essentially provides object detection features. Comparatively, SIFT is an extension of HOG, where normalized HOG features are generated based on sections within the input image.

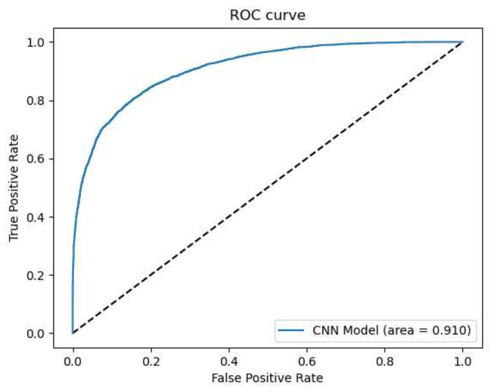
*OpenCV* contains the library *SIFT_create*() which returns keypoints and descriptors using the SIFT algorithm [16] in a three step process: 1. extracts keypoints and descriptors to store in two stacked arrays, 2. use $k$-means clustering to focus on the centers of the features which will be be used for, 3. a bag-of-features approach to represent the features and descriptors. After the features are extracted, these are used as input to convolutional layers. Similar to the previous CNN model, this model consists of convolutional blocks, but without pooling layers. Pooling layers are useful to generalize image data and reducing spatial resources and requirements, but it comes at the cost of fine data. For example, averaging sections of pixel data will still result in an image that is recognizable, but not as sharp. This cannot be done with SIFT feature data because extracted feature data does not have much of a relation to the previous or successive data. A comparison of Area under Curve values from the SIFT model using pooling layers is shown in Figure 8. Max- and Average Pooling layers correspond with a reduction in classification performance, with Max Pooling negatively affecting performance more than average. Max Pooling produced an AUC value of 0.766, Average Pooling produced an AUC value of 0.813, and finally, without pooling layers, the AUC value was 0.910. The SIFT model consists of seven convolutional blocks, where the first two convolutional layers have 32 filters, the next four layers have 64 filters, and the last convolutional layer has 128 filters. The model then has two final dense layers, one with 128 neurons and the final classifying layer with one neuron. The SIFT features are saved into a *Numpy ndaarray* and saved onto the hard drive for later use with the ensemble model, and also to save preprocessing time during debugging and multiple training iterations with the same data.



(a)



(b)



(c)

**Figure 8.** Figures from SIFT model. (**a**) ROC AUC with *MaxPooling*, (**b**) ROC AUC with *AveragePooling*, (**c**) ROC AUC without Pooling.

## 2.7. Ensemble Model

Finally, we create an ensemble model consisting of the previously created models as submodels, or branches, to combine their predictions before further training. Ensemble models typically take one of three forms: bagging, stacking, or boosting. Bagging involves training many models of the same algorithm on a smaller portion of the total dataset, and performing some decision strategy on the many outputs. Stacking involves using different model algorithms on the same input data, then performing some decision strategy on the outputs. Boosting sequentially places models that try to correct the previous model's prediction errors. In this paper, we discuss our stacked ensemble model. Previously, the models were classifying images based on features specific to that computer vision method of feature extraction, CNN, SIFT, and HOG. With the ensemble, the model will train on the collective extracted features of CNN, SIFT, and HOG per image to increase complexity of the input data. By providing more types of different features, the model would have a larger amount of information to aid in classification. Figure 9 shows the architecture of the ensemble model with the stacked submodels.

The previously created models share the architecture of the final two dense layers after a flatten layer which makes concatenating their outputs easier. The output of each branch's flatten layer is concatenated before a dense layer of 128 neurons and a final, classifying layer with one neuron. Also the learning rate was reduced to 0.00001 to help prevent the model from getting stuck at local minima. When this happens, the model would stop learning, loss and accuracy would remain the same. The architecture of the ensemble model is shown in Figure 9.

## 2.8. Training Parameters

The models were trained for 200 epochs with batch size 64. The model was also trained again for a longer period of 400 epochs on the $NeuralTextures$ dataset. We also use $k$-fold cross validation, with a $k$ value of 5 and 10. Ideally, $k = 10$ is more robust to verify training and prediction metrics, but memory restraints caused the ensemble model to have to use $k = 5$.

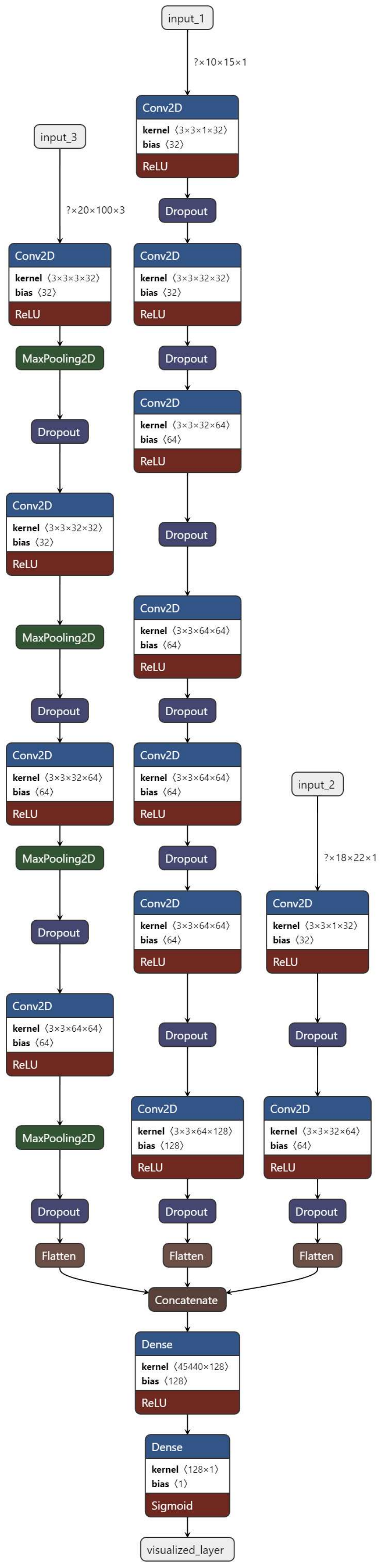
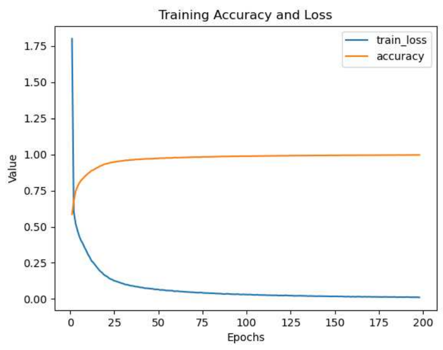


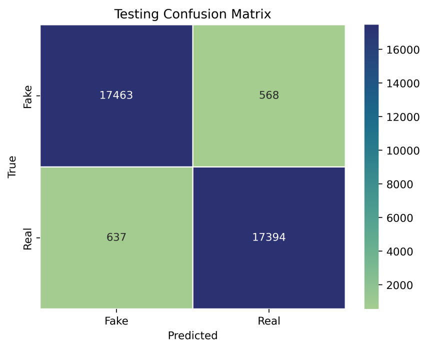
**Figure 9.** Layered visualization of the multichannel ensemble model.

## 3. Results

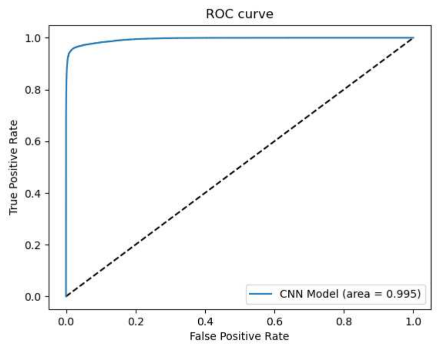
The library *Scikit-learn* contains various ways to visualize training and prediction results. This library contains $confusion\_matrix$, which outputs a confusion matrix based on the predicted classes with respect to the true classes, and $precision\_recall\_fscore\_support$ which outputs the $precision$, $recall$, $f\beta$-*score*, $f1$-*score*, and $support$, per class where applicable. *Scikit-learn* also contains the function $matthews\_corrcoef$ which returns the $\phi$-$coefficient$. For better visualization of the confusion matrix, we use the library *Seaborn* to create a confusion matrix from the $confusion\_matrix$ function with a colored heatmap. A sample set of results of training and testing the ensemble model is shown in Figure 10. The model performs with an average *k-fold* accuracy of 98.9%. The model has an ROC area under the curve value of about 0.995, meaning the ratio of true positive predictions to false positive predictions is significantly large, which is what we want.



(**a**)



(**b**)



(**c**)

**Figure 10.** (**a**) Training curve, (**b**) Confusion Matrix, (**c**) ROC.

## 3.1. Additional Metrics

Using the labels from the test set and the array of predictions, some other useful metrics can be retrieved, such as *Precision*, *Sensitivity*, $F_1$-*Score*, $F_\beta$-*Score*, and $\varphi$-$Coefficient$ [29]. *Precision* is the ability of the model to positively predict a class, with the function in Equation (4) [30].

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{4}$$

*Sensitivity* is the ability of the model to positively predict all of a certain class, with the function in Equation (5) [30].

$$Sensitivity = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{5}$$

$F_1$-*Score* is a metric that takes the harmonic mean of *Precision* and *Sensitivity*, with the function in Equation (6) [30].

$$F_1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

$F_\beta$-*Score* is the a weighted $F_1$-*Score* that shows the ability of the model to predict classes while minimizing false predictions. The function in Equation (7) [30] shows how to calculate this.

$$F_\beta - Score = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 Precision + Recall} \tag{7}$$

where the value of $\beta$ determines which metric to minimize, with $\beta < 1$ minimizing false positives, and $\beta > 1$ minimizing false negatives. When $\beta = 1$, $F_1$-*Score* $=$ $F_\beta$-*Score*. $\varphi$-$Coefficient$ is a binary classification metric that shows the relationship between the two classes. The function for $\varphi$-$Coefficient$ is defined in Equation (8).

$$\varphi - Coefficient = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)\,(TP + FN)\,(TN + FP)\,(TN + FN)}} \tag{8}$$

where $TP$, $TN$, $FP$, and $FN$ are *True Positive*, *True Negative*, *False Positive*, and *False Negative*, respectively. The returned value is between -1 and 1, where a negative value means the two classes are inversely related and a positive value means the two classes are positively related. When the values are positively related, as one increases, so does the other.

These additional metrics are shown in Table 2 when performing on the test set from *DeepFakeDetection* dataset.

**Table 2.** Predictions on a test set from *DeepFakeDetection*.

| | **Real** | **Fake** |
|---|---|---|
| *Precision* | 0.968 | 0.964 |
| *Sensitivity* | 0.964 | 0.968 |
| $F_\beta$-*Score* | 0.966 | 0.966 |
| $F_1$-*Score* | 0.966 | |
| $\phi$-$Coefficient$ | 0.966 | |

## 3.2. Additional Datasets

The previously constructed models were trained, validated, and tested on additional datasets. Along with DeepFakeDetection (DFD), these datasets also come from the deepfake collection *FaceForensics++* [14]: *Face2Face* [14] (F2F), *FaceSwap* [14,31] (FS), and *NeuralTextures* [14] (NT). The methods for generating these Deepfake datasets consist of face swapping and reenactment. In each of the additional datasets, we use $70,002$ images, where exactly half $(35,001)$ are classified as *Real* and the other half $(35,001)$ are classified as *Fake*. With the inclusion of SIFT and HOG data, the total number of inputs triple to 210, 006 elements, where each individual image from the original frames dataset is represented by 1) a periocular version, 2) its SIFT features, and 3) its HOG features.

Table 3 shows the number of images and sizes of feature data used as input for the model. The metrics of the additional datasets are shown below in Tables 4–6.

**Table 3.** Sizes of the various image and feature datasets used to compare the performance of the ensemble model.

| | Type | # of Images | Image Data | SIFT | HOG |
|---|---|---|---|---|---|
| **DFD** | Eyes<br>Face | 113, 572 | 2.6GB<br>8.2GB | 129.8MB<br>136.3MB | 171.4MB<br>1.71GB |
| **F2F** | Eyes<br>Face | 70, 002 | 1.7GB<br>5.0GB | 84.0MB | 110.9MB<br>1.10GB |
| **FS** | Eyes<br>Face | 70, 002 | 1.7GB<br>5.0GB | 84.0MB | 110.9MB<br>1.10GB |
| **NT** | Eyes<br>Face | 70, 002 | 1.7GB<br>5.0GB | 84.0MB | 110.9MB<br>1.10GB |

**Table 4.** Predictions on a test set from *Face2Face*.

| | **Real** | **Fake** |
|---|---|---|
| *Precision* | 0.961 | 0.828 |
| *Sensitivity* | 0.799 | 0.967 |
| $F_\beta$-*Score* | 0.872 | 0.892 |
| $F_1$-*Score* | 0.872 | |
| $\phi$-*Coefficient* | 0.778 | |

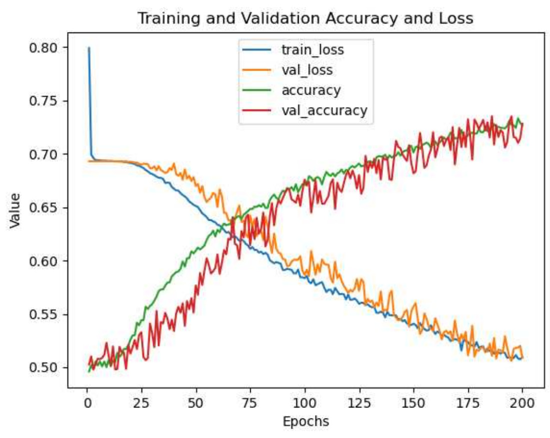
**Table 5.** Predictions on a test set from *FaceSwap*.

| | **Real** | **Fake** |
|---|---|---|
| *Precision* | 0.938 | 0.927 |
| *Sensitivity* | 0.927 | 0.931 |
| $F_\beta$-*Score* | 0.929 | 0.929 |
| $F_1$-*Score* | 0.929 | |
| $\phi$-*Coefficient* | 0.858 | |

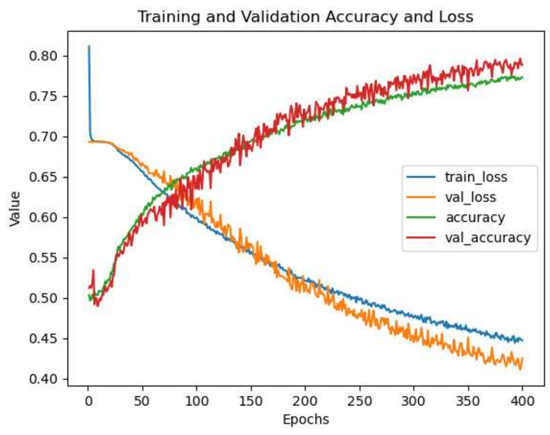
**Table 6.** Predictions on a test set from *NeuralTextures*.

| | **Real** | **Fake** |
|---|---|---|
| *Precision* | 0.878 | 0.918 |
| *Sensitivity* | 0.923 | 0.864 |
| $F_\beta$-*Score* | 0.896 | 0.889 |
| $F_1$-*Score* | 0.896 | |
| $\phi$-*Coefficient* | 0.787 | |

The graph produced from training on the *NeuralTextures* dataset shows the model could possibly perform better with longer training times. In Figure 11, Figure 11a shows how the model performs on the *NeuralTextures* dataset over 200 epochs, with a testing accuracy of 64.9%. Figure 11b shows how the model performs over 400 epochs, resulting in an accuracy increase to 77.6%.



(**a**)



(**b**)

**Figure 11.** Figures from training on the *NeuralTextures* dataset. (**a**) at 200 epochs, (**b**) at 400 epochs.

## 4. Discussion, Conclusion, and Future Direction

The prevelance of deepfakes is increasing and it is important that there are tools that can quickly discern a deepfake from original media. By using deep learning models such as convolutional neural networks, we have the ability to detect deepfakes from the DeepFakeDetection dataset with very high accuracy, approaching 98.9%. We were able to use both machine learning and deep learning feature extraction methods on periocular data to train an ensemble deep learning model and provide better classification results than previous works, such as [8,11,32].

The previously shown model in this work displays high accuracy results when classifying original and deepfake images. By including three feature extraction methods for the ensemble model to learn from, increasing the complexity of the training data, provide a more robust set of data for the model to provide a confident deepfake detection model. This model was created using time-insensitive data, where the model learned generalized features that would distinguish images into classes, yet this does not address how a future deepfake could look based on the training data at any given point in time. By using, at the comparatively most complex, convolutional layers, the model does not track time. This means the model weights have an "understanding" of what a deepfake from the trained dataset should look if a new deepfake was generated from that data. If the training data were changing over time, this would present a problem for a convolutional neural network. Videos are images displayed with respect to time. A model that can also predict on what a future deepfake may look like from previous video would serve to be useful in future deepfake detection models. A type of model to accomplish this is the Long Short-Term Memory model, or LSTM. These models are trained on feature data with respect to time to allow predictions on future data based on past data. LSTM models are becoming more popular with deep learning methods because the element of time provides invaluable information that brings more context to features. Vision Transformers [9] have also had use recently in deepfake detection. Initially, the transformer was used for natural language processing, but the idea behind encoding features as words was applied to pixel data and shows to produce an accuracy of about 91.5% and ROC AUC value of 0.91.

**Author Contributions:** Conceptualization, D.J. and K.R.; Data curation, D.J.; Formal analysis, K.R.; Funding acquisition, X.Y. and K.R.; Investigation, D.J. and Kaushik Roy; Methodology, D.J.; Project administration, K.R.; Software, D.J.; Supervision, K.R.; Validation, K.R.; Writing—original draft, D.J.; Writing—review & editing, X.Y. and K.R. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by National Science Foundation (NSF) grant number 1900187 Collaborative Research: Excellence In Research: Computational Framework and Data Science for Identification.

**Data Availability Statement:** The data presented in this study are openly available in GitHub at https://github.com/ondyari/FaceForensics and at https://doi.org/10.48550/arXiv.1901.08971.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| CNN | Convolutional Neural Network |
| SIFT | Scale-Invariant Feature Transform |
| HOG | Histogram of Oriented Gradients |
| LSTM | Long Short-Term Memory |
| MTCNN | Multitask Cascaded Neural Network |
| AUC | Area Under Curve |
| DFD | DeepFake Detection |
| F2F | Face2Face |
| FS | FaceSwap |
| NT | Neural Textures |
| MIT | Massachusetts Institute of Technology |

## References

1. Learning, M.O. Tackling the misinformation epidemic with “In Event of Moon Disaster”. https://news.mit.edu/2020/mit-tackles-misinformation-in-event-of-moon-disaster-0720.
2. Buzzfeed.; Peele, J. You Won’t Believe What Obama Says In This Video! https://news.mit.edu/2020/mit-tackles-misinformation-in-event-of-moon-disaster-0720, 2018.
3. derpfakes. Nick Cage DeepFakes Movie Compilation. https://www.youtube.com/c/derpfakes/, 2018.
4. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks **2014**. [arXiv:stat.ML/1406.2661]. https://doi.org/10.48550/ARXIV.1406.2661.
5. Mirsky, Y.; Lee, W. The Creation and Detection of Deepfakes: A Survey. *ACM Computing Surveys (CSUR), 2020, preprint* **2020**, [arXiv:cs.CV/2004.11138]. https://doi.org/10.1145/3425780.
6. Paul, O.A. Deepfakes Generated by Generative Adversarial Networks. *Georgia Southern University Honors College Theses* **2021**.
7. Shen, T.; Liu, R.; Bai, J.; Li, Z. "Deep Fakes" using Generative Adversarial Networks (GAN). *NoiseLab University of California San Diego* **2018**.
8. Karandikar, A. Deepfake Video Detection Using Convolutional Neural Network. *International Journal of Advanced Trends in Computer Science and Engineering* **2020**, *9*, 1311–1315. https://doi.org/10.30534/ijatcse/2020/62922020.
9. Wodajo, D.; Atnafu, S. Deepfake Video Detection Using Convolutional Vision Transformer. *ArXiv* **2021**, [arXiv:cs.CV/2102.11126].

10. Tran, V.N.; Lee, S.H.; Le, H.S.; Kwon, K.R. High Performance DeepFake Video Detection on CNN-Based with Attention Target-Specific Regions and Manual Distillation Extraction. *Applied Sciences* **2021**, *11*, 7678. https://doi.org/10.3390/app11167678.
11. Burroughs, S.; Roy, K.; Gokaraju, B.; Luu, K. Detection Analysis of DeepFake Technology by Reverse Engineering Approach (DREA) of Feature Matching, 2021. https://doi.org/10.1007/978-981-33-4893-6_36.
12. Kim, D.K.; Kim, K. Generalized Facial Manipulation Detection with Edge Region Feature Extraction. *arXiv* **2021**, [arXiv:cs.CV/2102.01381].
13. Suwajanakorn, S.; Seitz, S.M.; Kemelmacher-Shlizerman, I. Synthesizing Obama: learning lip sync from audio. *ACM Trans. Graph.* **2017**, p. 1–13. https://doi.org/10.1145/3072959.3073640.
14. Rössler, A.; Cozzolino, D.; Verdoliva, L.; Riess, C.; Thies, J.; Nießner, M. FaceForensics++: Learning to Detect Manipulated Facial Images. *arXiv* **2019**, [arXiv:cs.CV/1901.08971].
15. Dufour, N.; Gully, A.; Karlsson, P.; Vorbyov, A.V.; Leung, T.; Childs, J.; Bregler, C. DeepFakes Detection Dataset by Google & JigSaw. https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html.
16. Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* **2004**, *60*, 91–110. https://doi.org/10.1023/b:visi.0000029664.99615.94.
17. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA. IEEE Computer Society, 2005, pp. 886–893. https://doi.org/10.1109/CVPR.2005.177.
18. Foundation, P.S. Python. https://www.python.org/.
19. Itseez. OpenCV. https://opencv.org/.
20. Zhang, K.; Zhang, Z.; Li, Z.; Qiao, Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. *IEEE Signal Processing Letters* **2016**, [arXiv:cs.CV/1604.02878]. https://doi.org/10.1109/LSP.2016.2603342.
21. Ajoy, A.; Mahindrakar, C.U.; Gowrish, D.; A, V. DeepFake Detection using a frame based approach involving CNN; IEEE: Coimbatore, India, 2021; pp. 1329–1333. https://doi.org/10.1109/ICIRCA51532.2021.9544734.
22. Shad, H.S.; Rizvee, M.M.; Roza, N.T.; Hoq, S.M.A.; Khan, M.M.; Singh, A.; Zaguia, A.; Bourouis, S. Comparative Analysis of Deepfake Image Detection Method Using Convolutional Neural Network. *Computational Intelligence and Neuroscience* **2021**, *2021*. https://doi.org/10.1155/2021/3111676.
23. Al-Dhabi, Y.; Zhang, S. Deepfake Video Detection by Combining Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN); IEEE: SC, USA, 2021; pp. 236–241. https://doi.org/10.1109/CSAIEE54046.2021.9543264.
24. Hossain, M.A.; Sajib, M.S.A. Classification of Image using Convolutional Neural Network (CNN). *Global Journal of Computer Science and Technology* **2019**.
25. Ide, H.; Kurita, T. Improvement of learning for CNN with ReLU activation by sparse regularization; IEEE: Anchorage, AK, USA, 2017; pp. 2684–2691. https://doi.org/10.1109/IJCNN.2017.7966185.
26. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting **2014**. p. 1929–1958.
27. M., B.C. Training with Noise Is Equivalent to Tikhonov Regularization, 1994. https://doi.org/10.1.1.38.3008.
28. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift **2015**. [arXiv:cs.LG/1502.03167].
29. Fawcett, T. An introduction to ROC analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. https://doi.org/10.1016/j.patrec.2005.10.010.
30. Vakili, M.; Ghamsari, M.; Rezaei, M. Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification **2020**. [arXiv:cs.LG/2001.09636].

31. Kowalski, M. FaceSwap. https://github.com/MarekKowalski/FaceSwap/.
32. St, S.; Ayoobkhan, M.U.A.; Krishna Kumar, V.; Bacanin, N.; Venkatachalam, K.; Štěpán, H.; Pavel, T. Deep learning model for deep fake face recognition and detection. *PeerJ. Computer science* **2022**, *8*, e881. https://doi.org/10.7717/peerj-cs.881.