A Convolutional LSTM based Residual Network for Deepfake Video Detection

Shahroz Tariq, Sangyup Lee
Dept. of Computer Science and Engineering
Sungkyunkwan University
Suwon, South Korea
{shahroz,sangyup.lee}@g.skku.edu

Simon S. Woo*
Dept. of Applied Data Science
Sungkyunkwan University
Suwon, South Korea
swoo@g.skku.edu

arXiv:2009.07480v1 [cs.CV] 16 Sep 2020

ABSTRACT

In recent years, deep learning-based video manipulation methods have become widely accessible to masses. With little to no effort, people can easily learn how to generate deepfake videos with only a few victims or target images. This creates a significant social problem for everyone whose photos are publicly available on the Internet, especially on social media websites. Several deep learning-based detection methods have been developed to identify these deepfakes. However, these methods lack generalizability, because they perform well only for a specific type of deepfake method. Therefore, those methods are not transferable to detect other deepfake methods. Also, they do not take advantage of the temporal information of the video. In this paper, we addressed these limitations. We developed a Convolutional LSTM based Residual Network (CLRNet), which takes a sequence of consecutive images as an input from a video to learn the temporal information that helps in detecting unnatural looking artifacts that are present between frames of deepfake videos. We also propose a transfer learning-based approach to generalize different deepfake methods. Through rigorous experimentations using the FaceForensics++ dataset, we showed that our method outperforms five of the previously proposed state-of-the-art deepfake detection methods by better generalizing at detecting different deepfake methods using the same model.

KEYWORDS

Deepfake detection, Video forensics, Image manipulation

1 INTRODUCTION

Deep learning-based methods for synthetic image generation have sprouted tremendously in the last few years. These new methods can generate photorealistic images that can easily deceive average humans [15, 24, 25, 35, 40, 41]. Due to their ability, these methods have many applications in computer vision or graphics disciplines, such as human face generation [24] and photorealistic scenery generation [31]. However, there is also a dark side to all of this innovation. Many people with malicious intentions have used these methods to generate fake videos of celebrities and masses [11, 13, 14, 23], for which numerous approaches exist [15, 25, 40, 41]. This has started causing major social issues: a recent study claimed that 96% of the deepfakes originate from porn videos [30]. They come under the same umbrella of so-called Deepfakes. Recently, the research community has released numerous deepfake datasets to assist other researchers in developing detection mechanisms for these deepfakes. The most pioneering work is the FaceForensics++ dataset [35] developed in part by Google. Originally, the FaceForensics++ [35] dataset contained Pristine (1,000), Deepfakes (1,000), FaceSwap (1,000), Face2Face (1,000), and Neural Texture (1,000) videos. Later, Google contributed by supplementing real (363) and fake (3,000) videos. This year, Facebook launched a deepfake detection challenge with prize money of one million U.S. dollars to accelerate research in this field [6]. Lately, several deepfake detection methods with high zero-shot test accuracy on specific training deepfake datasets have emerged [3, 10, 21, 22, 29, 39]. However, they would have poor detection accuracy on new deepfake methods that were not present in the training set. This was our primary motivation to develop a generic and universal deepfake video detector, since it would be impractical to produce datasets for every new deepfake generation method. Therefore, we leverage massive deepfake datasets, such as FaceForensics++ [35], that are already available and employ transfer learning to detect other deepfake methods as well as newly generated ones.

Studies on the development of a generic deepfake detector have seen limited research activity [10]. Therefore, we aim to address these problems by developing a model that first trains on one deepfake method from the massive deepfake dataset of FaceForensics++ [35] and then uses a few-shot transfer learning method to learn about other deepfake methods. The number of sample videos required for few-shot learning is minimal, rendering the choice more practical. Our method differs from previous works in that we explored different transfer learning strategies and compared their results through rigorous experimentations on multiple datasets.

We noticed that most of the deepfake detection methods [10, 35] randomly extract frames (images) from videos for training and testing, hence a single frame-based detection method. However, after carefully observing numerous deepfake videos, we were surprised to discover tiny artifacts between consecutive frames within the deepfake videos through which we can identify these videos, as shown in Fig. 1. Therefore, we concluded that the temporal information between consecutive video frames is crucial for deepfake detection [36]. To incorporate the temporal aspect, we used a Convolutional LSTM, since it has shown to be useful for such tasks [43]. Therefore, we propose CLRNet, a Convolutional LSTM based Residual Network for Deep Fake Video Detection using Transfer Learning.

The main contributions are summarized as follows:

- **CLRNet**: We propose a novel architecture based on Convolutional LSTM and Residual Network for deepfake detection using a sequence of consecutive frames from a video.
- **Generalizability**: We provided a more generalized method than previous state-of-the-art deepfake detection approaches

*Corresponding Author

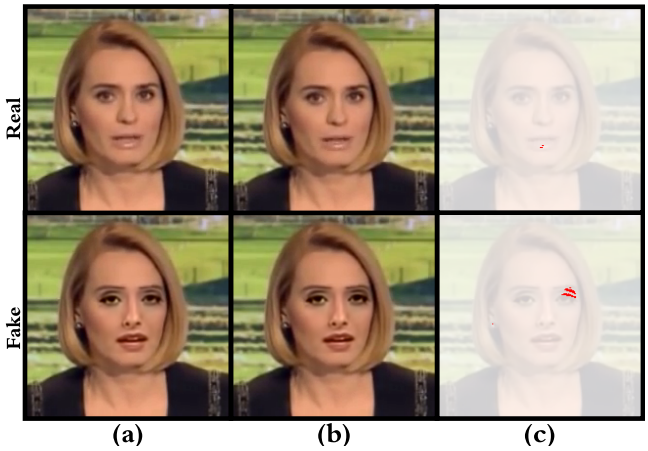


Figure 1: Difference between two consecutive frames of a deepfake video. (a) and (b) are the n^{th} and $(n+1)^{th}$ frames, respectively. For pristine videos, these frames are nearly the same, but for deepfake videos, there are inconsistencies. The difference between the n^{th} and $(n+1)^{th}$ frames highlights (red) those inconsistencies in (c). There is a small difference between real video frames, but the difference between fake video frames is more significant, as shown in (c).

with high accuracy and demonstrated it through rigorous experimentations.

2 RELATED WORK

Our work spans across different fields, such as deepfake detection, model generalization, and transfer learning. Therefore, we will briefly cover the related works in this section.

2.1 Deepfake Detection

The detection of abnormal eye blinking [26] has shown to be effective for the identification of inconsistencies in manipulated videos or images. Furthermore, image splice detection methods [4, 20, 37] aim to exploit the deviation resulting from splicing near the boundaries of manipulated regions in an image. Although inconsistencies in images generated from existing deepfake generation methods can be detected, new and more advanced generation methods are researched and developed every year. On the other hand, model-based methods, such as the measurement of features from demosaicing artifacts [16], lens aberrations [45], and JPEG artifacts resulting from the choice of different image processing methods [2], have traditionally been used to identify image manipulations. Nevertheless, they have shown to be unreliable when detecting machine-generated fake images, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), because the entire set of images are created from scratch. Deep learning-based approaches in a supervised environment have shown high detection accuracy. Specifically, Convolutional Neural Network (CNN) based approaches concentrated on automatically learning hierarchical representations from the RGB color images input [27, 33] or utilizing manipulation detection features [5], and using hand-crafted

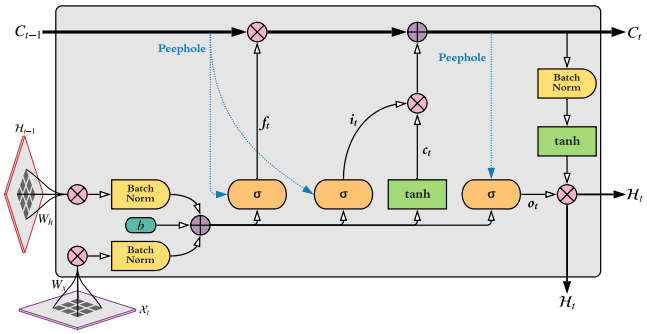


Figure 2: *Convolutional LSTM cell*: Visual representation of the Convolutional LSTM cell. The main structure is similar to that of an LSTM, but some additional components are present to incorporate the convolutional part. Here, $\mathcal{X}_t$ is the input, C_t is the cell output, and $\mathcal{H}_t$ is the hidden state. The gates are represented by i_t, f_t, and o_t.

features [9]. Tariq *et al.* [38, 39] introduced ShallowNet, a fast learning and effective CNN-based network for detecting GAN-generated images with high accuracy even at low resolution (64×64). Furthermore, Zhou *et al.* [47] applied a two-stream Faster R-CNN network, which can capture high and low-level image details. Rössler *et al.* [34, 35] presented a significantly improved performance on compressed images, which is essential for detecting deepfakes on social networking sites such as Instagram, Facebook, and Twitter. Most of the aforementioned approaches concentrate on detecting facial manipulations in a single video frame. However, as shown in Fig. 1, it is crucial to analyze the temporal information between consecutive frames in deepfake videos. In our approach, we use multiple consecutive frames to utilize this temporal information for an improved detection of deepfakes.

2.2 Detection with Consecutive Video Frames

Sabir *et al.* [36] proposed a detection method that utilizes both the CNN and Recurrent Neural Network (RNN) to capture the temporal information presented in 5 consecutive deepfake video frames. Also, Güera *et al.* [18] adopted a similar approach, extracting the features from up to 80 consecutive frames using CNN layers and feeding them to RNN layers to build a temporal information-aware deepfake detection model. Both methods [18, 36] extract features from CNN and pass it to RNN layers. At the same time, we build our CLRNet model using Convolutional LSTM cells, which can capture the spatio-temporal information directly from an input image sequence. However, most of these approaches yielded worse results when evaluated on datasets containing videos from a different deepfake generation method. Thus, we explored transfer learning for our CLRNet model to address this challenge.

2.3 Generalization via Transfer Learning

A variety of deepfake video generation techniques are constantly being developed and more sophisticated deep fake videos will arise in the future. However, collecting and producing a significant amount of new deepfake samples would be impractical. To cope with such

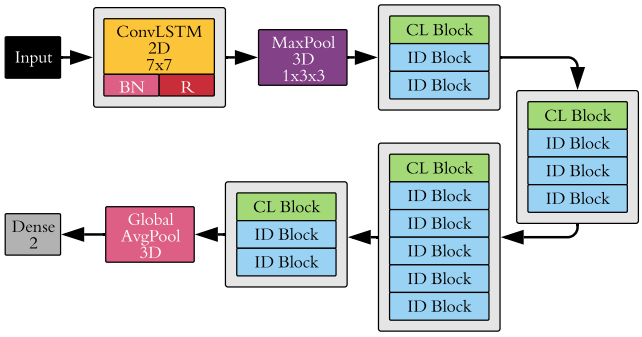


Figure 3: *CLRNet Architecture*: High-level architectural diagram of our Convolutional LSTM based Residual Network (CLRNet) model. The input to the model is a sequence of consecutive images and the output is a classification result, real or fake. We used Keras temrs to denote the layer names.

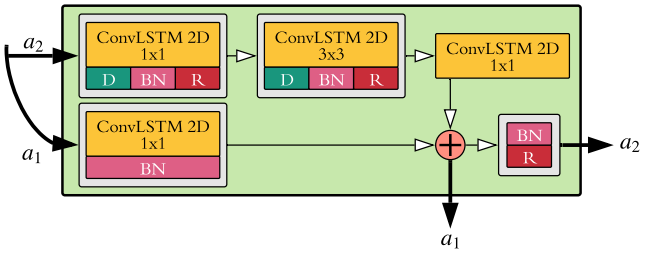


Figure 4: *CL Block*: Visual representation of the internal structure of the Convolutional LSTM block (CL Block). There are two ConvLSTM2D layers, each followed by dropout, BatchNorm, and ReLU. Afterward, we directly added the last ConvLSTM2D layer to a ConvLSTM2D with a BatchNorm layer from the shortcut connection to get a_1. Finally, a_2 results from BatchNorm followed by ReLU applied to the output of the addition operator. The input a_1 and a_2 are identical for the first CL Block, but different afterward.

situations, few-shot transfer learning (TL) is the key to the detection of deepfakes created by different methods. That is, what has been learned in one domain (e.g., FaceSwap) can be used to enhance the generalizability in another domain (e.g., Face2Face). Cozzolino *et al.* [10] have experimented the generalization of a single detection method for multiple target domains. In this work, we compare our approach against ForensicsTransfer [10] to demonstrate the enhanced generalizability and transferability.

3 OUR APPROACH

A frame-by-frame analysis of deepfake videos reveals the inconsistencies between consecutive frames in deepfake videos, which are absent in pristine videos. These inconsistencies include 1) a sudden change in brightness and contrast on a small region of the face, and 2) the size of some facial parts such as eyes, lips, and eyebrows changes between frames. These are minor inconsistencies that can be detected with a thorough examination. Figure 1 shows an example of such artifact from two consecutive frames of a deepfake video along with their sudden differences marked in red. These inconsistencies render the video somewhat unnatural. Motivated by this finding and observation, we developed a new deepfake detection method, the Convolutional LSTM Residual Network, that can account for these inconsistencies for the identification of real and fake videos.

3.1 Convolutional LSTM Cell

Shi *et al.* [43] stated that the main problem with handling spatio-temporal data in FC-LSTM [17] is the use of full connections during input-to-state and state-to-state transitions, and no spatial information is encoded. In contrast, Convolutional LSTM (ConvLSTM) overcomes this problem by introducing 3D tensors whose last two dimensions are spatial (rows and columns) for all the inputs $(\mathcal{X}_1, \ldots, \mathcal{X}_t)$, outputs $(C_1, \ldots, C_t)$, hidden states $(\mathcal{H}_1, \ldots, \mathcal{H}_t)$, and gates (i_t, f_t, o_t). In this paper, we follow the formulation of ConvLSTM by Shi *et al.* [43]. The Hadamard product and the Convolution operator are denoted by '$\circ$' and '$*$', respectively.

$$
\begin{aligned}
i_t &= \sigma\left(W_{x_i} * \mathcal{X}_t + W_{h_i} * \mathcal{H}_{t-1} + W_{c_i} \circ C_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{x_f} * \mathcal{X}_t + W_{h_f} * \mathcal{H}_{t-1} + W_{c_f} \circ C_{t-1} + b_f\right) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tanh\left(W_{x_c} * \mathcal{X}_t + W_{h_c} * \mathcal{H}_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{x_o} * \mathcal{X}_t + W_{h_o} * \mathcal{H}_{t-1} + W_{c_o} \circ C_t + b_o\right) \\
\mathcal{H}_t &= o_t \circ \tanh\left(C_t\right)
\end{aligned}
\tag{1}
$$

Furthermore, a visual representation of our ConvLSTM cell, based on Xavier [42] and implementation of Keras [8], is shown in Fig. 2.

3.2 Convolutional LSTM Residual Network

Sabir *et al.* [36] showed that a CNN based backbone encoding network, such as ResNet or DenseNet, connected to an RNN based network achieves high accuracy for deepfake detection tasks. They also find a sequence of images for an improved performance compared to using a single frame input. Furthermore, to avoid the vanishing gradient problem, we added residuals in our network. Based on previous research [28, 29, 36, 44] and our analysis of inconsistencies and/or artifacts in consecutive frames, we developed our Convolutional LSTM based Residual Network (CLRNet). Figure 3 shows a visual representation of the architecture for our CLRNet model. The input elements of our model are 3D tensors preserving the entire spatial information for consecutive frames. Therefore, we used ConvLSTM (CL) cells instead of Convolution cells,as shown in Fig. 4 and 5. Shi *et al.* [43] state that stacking ConvLSTM in this way provides a strong representational power to the model. We developed the core architecture of our CLRNet using two types of building blocks (i.e., CL block and ID block). Figures 4 and 5 provide a pictorial representation of the CL and ID blocks. These blocks can be related to the Convolution and Identity building blocks from ResNet [19], respectively. The building blocks in our CLRNet model have two outputs (i.e., a_1 and a_2), as shown in Fig. 4 and 5. In both blocks, a_1 is the output following the addition step, whereas a_2 is the output of the addition step followed by the batch normalization and ReLU layers. The output a_1 and a_2 serve as inputs for the next block. The CL block contains a ConvLSTM cell followed by a batch

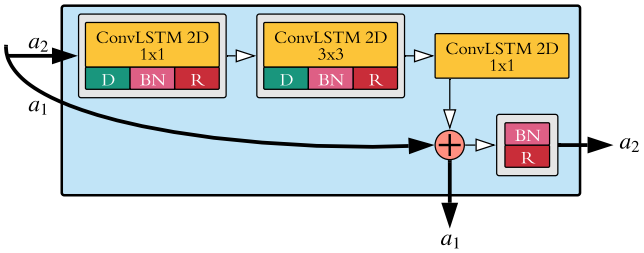


Figure 5: *ID Block*: Visual representation of the internal structure of the Identity block (ID Block). Similar to the CL Block, there are two ConvLSTM2D layers, each followed by dropout, BatchNorm, and ReLU. Afterward, we add the last ConvLSTM2D layer to the input a_1 from the shortcut connection to get output a_1. Finally, a_2 is the result of BatchNorm and ReLU applied to the output of the addition operator.

normalization layer on the shortcut path, whereas in the ID block, the shortcut path directly connects the input a_1 to the addition layer.

3.3 Transfer Learning Strategies

For transfer learning, we evaluated the following three strategies: 1) Single-source to Single-target: we train our model with one large deepfake dataset and then apply transfer learning to one target deepfake dataset (e.g., the model is first trained on the DF dataset and then transfer learned to FS), 2) Multi-source to Single-target: we train our model on multiple sources and apply transfer learning to a single target domain, and 3) Single-source to Multi-target: we train our model on a single source domain and then use a small volume of multiple target domains to apply transfer learning. We will discuss the advantages and disadvantages of each strategy in Section 4.

3.4 Implementation Details

3.4.1 Dataset description. To compare our method with different baselines, we used DeepFake (DF), FaceSwap (FS), Face2Face (FS), NeuralTextures(NT), and DeepFakeDetection (DFD) datasets in [35]. Table 1 describes all the datasets used for this paper. Each class inside the FaceForensics++ [35] dataset, except DFD, contains 1,000 videos. We used the first 750 videos out of 1,000 for training, the next 125 for validation, and the remaining of the 125 for testing. For DFD, we selected only 300 videos (250 for training, 25 for validation, and 25 for testing).

3.4.2 Preprocessing. From each real and fake video, we extracted 16 samples such that each sample contains five consecutive frames. We used multi-task CNN (MTCNN) [46] to detect the face landmark information inside the extracted frame. Afterward, we used this landmark information to crop the face from the image and aligned it to the center. Lastly, we resized all the frames to a 240 × 240 resolution.

3.4.3 Data Augmentation. We also applied data augmentation techniques to diversify the training data. We varied the following conditions: 1) Brightness (-30% to 30%), 2) Channel shift (-50 to 50), 3) Zoom (-20% to 20%), 4) Rotation (-30° degrees to 30°), and 5) Horizontal flip (50% probability).

Table 1: *Dataset details*: There are 1,000 videos for Pristine, DeepFake, FaceSwap, Face2Face, and Neural Textures, respectively: we used 750 real and 750 fake videos for training, 125 real and 125 fake videos for validation as well as for testing. There are 3,363 videos (363 real, 3,000 fake) in the DeepFakeDetection dataset: we used 250 real and 250 fake videos for training, 25 real and 25 fake for validation, as well as for testing. For transfer learning, we used 10 real and 10 fake videos.

Datasets	Total Videos	Base Training videos	Transfer Learning videos	Samples per video
Pristine (Real)	1,000	750	10	16
DeepFake (DF)	1,000	750	10	16
FaceSwap (FS)	1,000	750	10	16
Face2Face (F2F)	1,000	750	10	16
NeuralTextures (NT)	1,000	750	10	16
DeepFakeDetection (DFD)	3,363	250	10	16

3.4.4 Training. The idea behind our method is to train on one of the widely available deepfake datasets and then use transfer learning to train for the other datasets with a small amount of samples. We trained our model by taking 16 samples from each of the 750 real (Pristine) and 750 fake (DF or FS or F2F or NT) videos.

3.4.5 Transfer Learning Configuration. Once the training is complete for the base model, we performed transfer learning to other datasets by using a small subset of the target dataset (10 videos). For our CLRNet model, we freezed the first 120 layers and applied transfer learning to the model with an equal number of videos from the source and target datasets, that is, 10 videos per dataset.

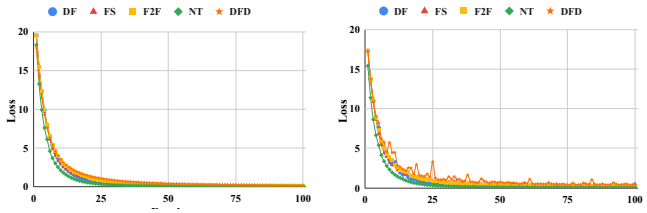
3.4.6 Machine Configuration. We used Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz with 256.0GB RAM and NVIDIA GeForce Titan RTX. We used TensorFlow v1.13.0 [1] with Keras Library [8] on Python v3.7.5 for the implementation of our CLRNet model.

3.4.7 Evaluation Metrics. We used Precision, Recall, and F1-Score for the evaluation. Due to space limitations, we are reporting only the F1-Scores in Table 2 and 3. ShallowNet, Xception, FF++ [35], and FT uses a single frame as input for training and testing sets, whereas Sabir *et al.* [36] and our CLRNet uses five consecutive frames as input. We kept the same number of real and fake images in the training, validation, and test sets to minimize the influence of data imbalance during evaluation.

3.5 Baseline Methods

We compared CLRNet with several state-of-the-art methods and tried our best to implement them according to their specifications. The following is a description of these methods.

3.5.1 Xception. The Xception Network [7] is considered as the state-of-the-art for image classification task. We used the Keras [8] implementation of Xception, which is pre-trained on the ImageNet dataset [12].



(a) Training Loss (b) Validation Loss

Figure 6: *Training vs Validation Loss of CLRNet*: Training and validation losses of CLRNet on the DeepFake (DF), FaceSwap(FS), Face2Face (F2F), NeuralTextures (NT), and DeepFakeDetection (DFD) datasets. The loss progression for both training and validation is descending very similarly, which shows that our model is learning accurately and is not overfitting to the training data.

3.5.2 ShallowNet. Tariq *et al.* [39] showed that ShallowNet [38] achieves high accuracy in detecting computer-generated images. We developed ShallowNet using Python v3.6.8 using TensorFlow v1.14.0 [1] and used the Keras v2.2.4 [8].

3.5.3 FaceForensics++ (FF++). Rössler *et al.* [35] used a modified version of the Xception Network to detect DeepFake, FaceSwap, Face2Face, and NeuralTextures. We are directly using results from FaceForensics++ [35], since they used the same dataset.

3.5.4 DenseNet with Bidirectional RNN.. Sabir *et al.* [36] used DenseNet with a bidirectional RNN to achieve high accuracy on DeepFake, FaceSwap, and Face2Face datasets. Similar to our CLRNet, this work also uses five consecutive frames for the training and testing of the model. We are directly using the results of Sabir *et al.* [36], since they used the same dataset.

3.5.5 Forensics Transfer (FT).. Cozzolino *et al.* [10] developed a weakly supervised method for domain adaptation using an autoencoder based approach. They divide the latent space into real and fake parts to achieve higher detection accuracy on the Face2Face and FaceSwap datasets. For the implementation of ForensicTransfer autoencoder [10], we used PyTorch v1.1.0 [32] on Python v3.6.8.

4 RESULTS

We have performed extensive experiments to evaluate and compare the performances of CLRNet and baseline methods. Only the most important experiments and their findings are discussed in this paper. The following sections will discuss the results from different experiments in detail.

4.1 Learning Capability of CLRNet

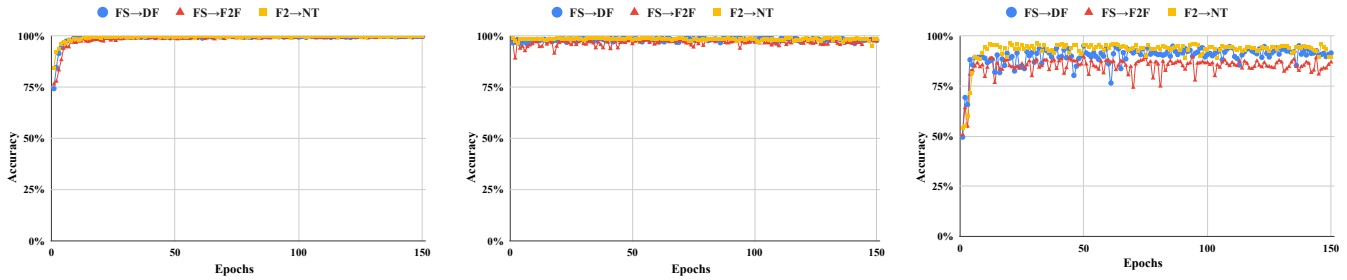
Figure 6 presents the training and validation losses of CLRNet on different datasets (DF, FS, F2F, NT, and DFD). We can observe that the training and validation lossess gradually descend in a similar fashion, which shows that our CLRNet model is not overfitting to the training dataset and is learning distinguishable features between real and fake images from the training set. As shown in Fig. 6, there are slight fluctuations in the validation loss of DFD, but it is not the case for other datasets. We believe that this is due to the smaller size of the dataset used for DFD training (250 videos) as compared to other datasets (750 videos), as shown in Table 1. Another cause is the dynamic environment in DFD videos, making it harder for the model to learn. However, our CLRNet model was able to learn from the DFD dataset, even with a smaller data size, and achieved a 96.00% F1-score, as shown in Table 2.

4.2 Performance on Base Dataset

We trained our CLRNet and other baseline models on a base training dataset (DF or FS), as shown in Table 2. Then we compared the performance of the trained model on a test set from the base dataset and also evaluated the zero-shot performance from the test set of other datasets. The baseline methods (ShallowNet, Xception, and FT) and CLRNet are trained and evaluated on the set of images from the datasets. However, the results of FF++ [35] and Sabir *et al.* [36] are directly taken from their work. Table 2 reports all the results. As we can observe from Table 2, when trained on the DeepFake base dataset, FT (99.35%) outperforms CLRNet (99.02%) by a slight margin. The performances of our vanilla implementation of Xception (86.00%) and ShallowNet (56.65%) do not bear good results. However, the modified versions of Xception from FF++ (96.36%) and DenseNet of Sabir *et al.* (96.90%) perform better than ShallowNet (56.65%) and the vanilla Xception (86.00%). The zero-shot performance is relatively low (below 85%) for all methods: FT performed the best on F2F (74.12%) and NT (83.54%), and CLRNet performed the best on DFD (60.38%). When trained with the FaceSwap (FS) base dataset, CLRNet (FS: 98.05%) is the best performer as compared to the best baseline method FT (FS: 96.17%), as shown in Table 2. Similar to the DF base dataset, Xception (FS: 85.37%), FF++ (FS: 90.29%), and Sabir el al. (FS: 94.35%) showed decent performance, and ShallowNet showed the worst performance (FS: 52.75%). We also evaluated our CLRNet method on NeuralTextures (NT) and DeepFakeDetection (DFD) base datasets. As shown in Table 2, CLRNet achieves a very high F1-score for both datasets (NT: 99.50%, DFD: 96.00%). From Table 2, we can observe that there is no best method in terms of the zero-shot performance. Therefore, we explore transfer learning in the next experiments.

4.3 Transfer Learning with Source and Target

In our preliminary experiments, we observed that when the model is trained with only the target dataset for transfer learning, its performance on the source dataset drops (as low as 50%), which is not ideal for building a generic deepfake detector. Therefore, we combine a small volume of the source (10 videos) and target (10 videos) datasets for transfer learning, so that the model can remember the features of the source dataset as well. Figure 7 shows the training accuracy (source + target), validation accuracy on the source dataset (FS), and validation accuracy on the target dataset (DF or F2F or NT). We can observe from Fig. 7 that the validation accuracy of the source dataset (FS) slightly drops (1~2%), and the validation accuracy on the target dataset (DF or F2F or NT) increases over time (reaches up to 90%). This technique allows CLRNet to achieve high performance on transfer learning for both the source and target datasets. In the next section, we will discuss the results of transfer learning.



(a) ***Training (Source + Target)*****: We used ten videos from source and ten from the target dataset. CLRNet shows high learning capability, even with such small amount of data.**

(b) ***Validation (Source)*****: We achieved high accuracy on the source dataset by providing ten videos from the source dataset during transfer learning.**

(c) ***Validation (Target)*****: We also achieved relatively high accuracy on the target dataset by using only ten videos for the target dataset with CLRNet.**

Figure 7: ***Transfer Learning Accuracy of CLRNet*****: Comparison of (a) transfer learning (TL) training accuracy with (b) the validation accuracy of the source and (c) the validation accuracy of the target. The arrow '→' represents the TL from a source dataset to a target dataset.**

4.4 Transfer Learning Performance

For transfer learning, we used only CLRNet and the best performing baseline model on the base dataset, namely FT. Cozzolino *et al.* [10] performed only the Face2Face (source) to FaceSwap (target) experiment for the FF++ dataset [35]. We performed extensive experiments with FT and compared it with our CLRNet model. We performed three different types of experiments, each consisting of multiple sub-experiments, for the transfer learning (TL) task. The following sections provide the details of these experiments and the performance evaluation of CLRNet and FT.

4.4.1 Exp 1. Single–source to Single–target. In this experiment, we used one source dataset (either FS, F2F, NT, DFD or DF) and one target dataset (either FS, F2F, NT, DFD or DF) for transfer learning and compared the performance of FT and CLRNet. Due to space limitations, we partially report this experiment in Table 3. In experiment 1, we set DeepFake (DF) as the source and FaceSwap (FS) as the target. The CLRNet accuracy increases from 50.00% to 83.08% for the target (FS) and drops from 99.02% to 90.95% for source (DF), as shown in Table 2 and 3. In contrast, FT accuracy drops from 50.00% (FS) and 99.35% (DF) to 47.72% (FS) and 62.59% (DF), respectively, as shown in Table 2 and 3. Similarly, when Face2Face (F2F) is the target, the CLRNet accuracy increases from 53.73% to 88.35% for the target, whereas the FT accuracy increases only by 5% from 74.12% to 79.31%. This low performance demonstrates the limitation of FT regarding generalization. In experiment 2, we set FaceSwap (FS) as the source, and the performance for the target datasets (DF and F2F) are even better than in experiment 1, in favor of CLRNet. When DeepFake (DF) is the target, CLRNet (FS: 96.33%, DF: 92.47%) outperforms FT (FS: 44.93%, DF:76.94%) by 50% for FS and 16% for DF, respectively. Also, when Face2Face (F2F) is the target, CLRNet (FS: 93.93%, F2F: 87.48%) outperforms FT (FS: 55.74%, F2F: 53.73%), as shown in Table 3. We also experimented with CLRNet by setting the target as NeuralTextures (NT) or DeepFakeDetection (DFD). After transfer learning, the performance on NT increases from 98.05% to 98.12% for the source (FS) and from 53.33% to 95.50% for the target (NT), as shown in Table 2 and 3. Finally, for DFD, we observed a similar trend and achieved an increase from 49.13% to 88.88%.

Based on our experiment, we observed that the accuracy of CLRNet always increases for the target as compared to its zero-shot performance with a slight decrease or sometimes an increase in the source accuracy, as shown in Table 2 and 3. On the other hand, the performance of FT is very unstable and generally worse than the single dataset performance, which shows significant limitation of FT; CLRNet overcomes this limitation and achieves better performance.

4.4.2 Exp 2. Multi–source to Single–target. In this experiment, we first trained the model with two different source datasets and then performed transfer learning to a new target dataset. In Table 3, we report the best performing model of this experiment for both FT and CLRNet, namely FS+DF (source) and F2F (target). The accuracy of CLRNet (FS: 94.37%, DF: 92.15%) on the source is significantly higher compared to that of FT (FS: 45.17%, DF: 64.08%). Similarly, CLRNet(F2F: 86.22%) outperforms FT (F2F: 55.98%) on the target dataset, as shown in Table 3. CLRNet also exceeds the performance of FT in terms of zero-shot learning on NT (CLRNet: 79.75%, FT: 62.09%) and DFD (CLRNet: 63.12%, FT: 51.38%). This experiment also validates the superiority of CLRNet over FT at deepfake detection generalization.

4.4.3 Exp. 3. Single–source to Multi–target. In this experiment, we test the generalizability of our CLRNet model, that is, its performance to detect multiple deepfake types at once with only a small amount of data. For this experiment, we first trained our CLRNet model on a source dataset (FS) and then performed three types of experiments. In experiment 1, we set the target as DF+F2F and evaluated the accuracy of the model after transfer learning. From Table 3, we can observe the performance for the source (FS: 94.55%) and the target (DF: 91.77%, F2F:87.75%). In experiment 2, we set the target as DF+F2F+NT. From Table 3, we can see the accuracy for the source (FS: 94.35%) and the target (DF: 90.67%, F2F:85.78%, NT: 91.47%). In this experiment, we set the target as DF+F2F+NT+DFD. In Table 3, we can see the accuracy for the source (FS: 93.70%)

Table 2: *The base dataset and zero-shot performance*: This table shows the performance comparison of our CLRNet with five different baseline models. First, we trained the models on a base dataset. Afterward, we tested all models on the base dataset (highlighted in gray) and on other datasets as well (zero-shot). Our CLRNet has the highest performance for all base datasets except one. However, for zero-shot performance, CLRNet performed the best on FS and FT as well as on DF and F2F. '‡' represents our implementation of the method.

Method	Base Dataset	DF (%)	FS (%)	F2F (%)	NT (%)	DFD (%)
ShallowNet‡	DF	56.65	**50.32**	54.15	35.13	41.23
Xception		86.00	49.78	57.26	61.49	**68.10**
FF++ [35]		96.36	-	-	-	-
Sabir *et al.* [36]		96.90	-	-	-	-
FT‡		**99.35**	50.00	**74.12**	**83.54**	48.45
CLRNet (Ours)		99.02	50.00	53.73	69.75	60.38
ShallowNet‡	FS	47.09	52.75	47.50	46.96	**49.58**
Xception		**49.49**	85.37	**56.61**	52.05	48.08
FF++ [35]		-	90.29	-	-	-
Sabir *et al.* [36]		-	94.35	-	-	-
FT‡		48.86	96.17	54.39	47.45	36.66
CLRNet (Ours)		48.53	**98.05**	50.15	**53.33**	49.13
FF++ [35]	NT	-	-	-	80.67	-
CLRNet (Ours)		**50.12**	**49.93**	**49.80**	**99.50**	**50.00**
CLRNet (Ours)	DFD	**65.08**	**51.92**	**60.32**	**63.10**	**96.00**

and the target (DF: 91.23%, F2F:87.50%, NT: 91.30%, DFD: 87.13%). Based on these experiments, we concluded: 1) that by increasing the number of target datasets, the source dataset accuracy slightly decreases (1~2%), and that 2) CLRNet generalizes well on different deepfake datasets, since its performance is better than the zero-shot performance on the same dataset in all of our experiments.

5 CONCLUSION

We introduced CLRNet, which is successfully applied to detect a variety of deepfake videos. Instead of using a single frame, our model uses a sequence of consecutive frames from the video as an input, which helps our CLRNet model to capture and incorporate the temporal information and detect artifacts present between consecutive frames. Through more than 5 different experiments, we have shown the superiority of our CLRNet model over the previous baselines and state-of-the-art methods. To conclude, we addressed the shortcomings of the previous state-of-the-art methods by proposing a more generalizable model with a better detection performance. From this work, we hope that our research is a stepping stone toward developing more generalized deepfake detectors and future work will continue to challenge and improve existing deepfake detection methods to more generalizable and universal approaches.

Table 3: *Transfer Learning Performance*: This table shows the performance comparison of our CLRNet model and the best baseline method from the previous experiment, namely FT. In this experiment, we first trained the model on a source dataset and then used a small target dataset (highlighted in gray) consisting of 10 videos to perform transfer learning. Afterward, we performed testing on all datasets. We performed ten experiments belonging to 3 different types: 1) Single-source to single-target, 2) Multi-source to single target, and 3) Single-source to Multi-target. Our CLRNet model performed the best in all scenarios. '‡' represents our implementation of the method.

Method	Source	Target	DF (%)	FS (%)	F2F (%)	NT (%)	DFD (%)
Single–source to Single–target							
FT‡	DF	FS	62.59	47.72	**61.34**	**68.51**	51.38
CLRNet			**90.95**	**83.08**	48.68	65.00	**53.87**
FT‡		F2F	83.70	**54.92**	79.31	**82.24**	54.92
CLRNet			**97.18**	49.28	**88.35**	78.05	**68.13**
FT‡	FS	DF	76.94	44.93	**66.42**	70.53	49.78
CLRNet			**92.47**	**96.33**	65.58	**75.80**	**59.13**
FT‡		F2F	56.06	55.74	53.73	55.38	47.63
CLRNet			**79.87**	**93.93**	**87.48**	**78.00**	**52.50**
FT‡		NT	-	-	-	-	-
CLRNet			51.82	98.12	50.90	95.50	49.13
FT‡		DFD	-	-	-	-	-
CLRNet			70.97	96.15	51.85	77.02	88.88
Multi–source to Single–target							
FT‡	FS+DF	F2F	64.08	45.17	55.98	62.09	51.38
CLRNet			**92.15**	**94.37**	**86.22**	**79.75**	**63.12**
Single–source to Multi–target							
CLRNet (best)	FS	DF+F2F	**91.77**	**94.55**	**87.75**	85.12	69.13
		DF+F2F +NT	90.67	94.35	85.78	**91.47**	69.50
		DF+F2F +NT+DFD	91.23	93.70	87.50	91.30	**87.13**

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Shruti Agarwal and Hany Farid. 2017. Photo forensics from JPEG dimples. In *2017 IEEE Workshop on Information Forensics and Security (WIFS)*. IEEE, 1–6.

[3] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. 2019. Protecting World Leaders Against Deep Fakes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 38–45.

[4] Jawadul H Bappy, Amit K Roy-Chowdhury, Jason Bunk, Lakshmanan Nataraj, and BS Manjunath. 2017. Exploiting spatial structure for localizing manipulated image regions. In *Proceedings of the IEEE international conference on computer vision*. 4970–4979.

[5] Belhassen Bayar and Matthew C Stamm. 2016. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*. ACM, 5–10.

[6] Deepfake Detection Challenge. 2019. Deepfake Detection Challenge. https://www.kaggle.com/c/deepfake-detection-challenge. Accessed: 2020-02-12.

[7] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[8] François Chollet et al. 2015. Keras. https://keras.io.

[9] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. 2017. Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*. ACM, 159–164.

[10] Davide Cozzolino, Justus Thies, Andreas Rössler, Christian Riess, Matthias Nießner, and Luisa Verdoliva. 2018. Forensictransfer: Weakly-supervised domain adaptation for forgery detection. *arXiv preprint arXiv:1812.02510* (2018).

[11] Adrian Croft. 2019. From Porn to Scams, Deepfakes Are Becoming a Big Racket-And That's Unnerving Business Leaders and Lawmakers. https://fortune.com/2019/10/07/porn-to-scams-deepfakes-big-racket-unnerving-business-leaders-and-lawmakers. Accessed: 2020-02-11.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[13] EJ Dickson. 2019. Deepfake Porn Is Still a Threat, Particularly for K-Pop Stars. https://www.rollingstone.com/culture/culture-news/deepfakes-nonconsensual-porn-study-kpop-895605. Accessed: 2020-02-11.

[14] Charlotte Edwards. 2019. Making deepfake porn could soon be as easy as using Instagram filters, according to expert. https://www.thesun.co.uk/tech/9800017/deepfake-porn-soon-easy. Accessed: 2020-02-11.

[15] FaceSwapDevs. [n.d.]. Deepfakes_faceswap - GitHub Repository. https://github.com/deepfakes/faceswap. Accessed: 2019-11-05.

[16] Pasquale Ferrara, Tiziano Bianchi, Alessia De Rosa, and Alessandro Piva. 2012. Image forgery localization via fine-grained analysis of CFA artifacts. *IEEE Transactions on Information Forensics and Security* 7, 5 (2012), 1566–1577.

[17] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[18] David Güera and Edward J Delp. 2018. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–6.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[20] Minyoung Huh, Andrew Liu, Andrew Owens, and Alexei A Efros. 2018. Fighting fake news: Image splice detection via learned self-consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 101–117.

[21] Hyeonseong Jeon, Youngoh Bang, and Simon S Woo. 2019. FakeTalkerDetect: Effective and Practical Realistic Neural Talking Head Detection with a Highly Unbalanced Dataset. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 0–0.

[22] Hyeonseong Jeon, Youngoh Bang, and Simon S Woo. 2020. FDFtNet: Facing Off Fake Images using Fake Detection Fine-tuning Network. *arXiv preprint arXiv:2001.01265* (2020).

[23] Michael Kan. 2019. Most AI-Generated Deepfake Videos Online Are Porn. https://www.pcmag.com/news/371193/most-ai-generated-deepfake-videos-online-are-porn. Accessed: 2020-02-11.

[24] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).

[25] Marek Kowalski. 2016. FaceSwap - GitHub Repository. https://github.com/MarekKowalski/FaceSwap. Accessed: 2020-02-12.

[26] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. 2018. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 1–7.

[27] Yaqi Liu, Qingxiao Guan, Xianfeng Zhao, and Yun Cao. 2018. Image forgery localization based on multi-scale convolutional neural networks. In *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*. ACM, 85–90.

[28] Falko Matern, Christian Riess, and Marc Stamminger. 2019. Exploiting visual artifacts to expose deepfakes and face manipulations. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. IEEE, 83–92.

[29] Ghazal Mazaheri, Niluthpol Chowdhury Mithun, Jawadul H Bappy, and Amit K Roy-Chowdhury. 2019. A Skip Connection Architecture for Localization of Image Manipulations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 119–129.

[30] Ivan Mehta. 2019. A new study says nearly 96 of deepfake videos are porn. https://thenextweb.com/apps/2019/10/07/a-new-study-says-nearly-96-of-deepfake-videos-are-porn. Accessed: 2020-02-11.

[31] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2337–2346.

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[33] Yuan Rao and Jiangqun Ni. 2016. A deep learning approach to detection of splicing and copy-move forgeries in images. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 1–6.

[34] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. 2018. Faceforensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179* (2018).

[35] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. 2019. FaceForensics++: Learning to Detect Manipulated Facial Images. In *ICCV 2019*.

[36] Ekraam Sabir, Jiaxin Cheng, Ayush Jaiswal, Wael AbdAlmageed, Iacopo Masi, and Prem Natarajan. 2019. Recurrent Convolutional Strategies for Face Manipulation Detection in Videos. *Interfaces (GUI)* 3 (2019), 1.

[37] Ronald Salloum, Yuzhuo Ren, and C-C Jay Kuo. 2018. Image splicing localization using a multi-task fully convolutional network (MFCN). *Journal of Visual Communication and Image Representation* 51 (2018), 201–209.

[38] Shahroz Tariq, Sangyup Lee, Hoyoung Kim, Youjin Shin, and Simon S Woo. 2018. Detecting both machine and human created fake face images in the wild. In *Proceedings of the 2nd International Workshop on Multimedia Privacy and Security*. ACM, 81–87.

[39] Shahroz Tariq, Sangyup Lee, Hoyoung Kim, Youjin Shin, and Simon S Woo. 2019. GAN is a friend or foe?: a framework to detect various fake face images. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 1296–1303.

[40] Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred Neural Rendering: Image Synthesis using Neural Textures. *arXiv preprint arXiv:1904.12356* (2019).

[41] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. 2018. Face2Face: Real-time Face Capture and Reenactment of RGB Videos. *Commun. ACM* 62, 1 (Dec. 2018), 96–104. https://doi.org/10.1145/3292039

[42] Alexandre Xavier. 2019. An introduction to ConvLSTM. https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7 Accessed: 2019-11-05.

[43] Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*. 802–810.

[44] Xin Yang, Yuezun Li, and Siwei Lyu. 2019. Exposing deep fakes using inconsistent head poses. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8261–8265.

[45] Ido Yerushalmy and Hagit Hel-Or. 2011. Digital image forgery detection based on lens and sensor aberration. *International journal of computer vision* 92, 1 (2011), 71–91.

[46] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. 2016. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters* 23, 10 (2016), 1499–1503.

[47] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. 2018. Learning rich features for image manipulation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1053–1061.

A APPENDIX ON REPRODUCIBILITY

This section describes all the necessary information required to reproduce the results from the experiments for CLRNet and all baselines methods. Here, we tried our best to cover everything to aid in reproducibility.

A.1 Hardware and software Configuration

We used Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz with 256.0GB RAM and NVIDIA GeForce Titan RTX. The following package versions were used on Python v3.7.5: TensorFlow-gpu: v1.13.1; Keras-applications: v1.0.8; Keras-preprocessing: v1.1.0; cudatoolkit: v10.0.130; numpy: v1.17.4; Pandas: v0.25.3; Opencv: 3.4.2; scikit-learn: v0.21.3; PyTorch: v1.1.0.

A.2 Deepfake Video Dataset

We downloaded the dataset (DeepFake, Face2Face, FaceSwap, NeuralTextures, and DeepFakeDetection) from the FaceForensics++ GitHub repository[1] for the training and evaluation of the baseline and our CLRNet model.

A.3 Implementation of CLRNet

We implemented our CLRNet model using TensorFlow and Keras on Python. All the layers used in the model are available in the Keras library. A detailed view of CLRNet architecture is provided in Table 6. We used the Keras convention to name the layers. The input shape for our model is (Videos, Samples, Rows, Columns, Channels). The ImageDataGenerator in Keras can load data of the form (Samples, Rows, Columns, Channels). Therefore, we could not use the ImageDataGenerator provided by the Keras library. So we built our own VideoDataGenerator with very similar functionality as that of ImageDataGenerator. However, as we planned to provide the input in a set of 5 consecutive frames, we had to implement some extra features as well. All of this is present in our source code, which we plan to release along with this paper.

A.4 Baseline Model Implementation

We compared our CLRNet model with five baseline models (Xception, ShallowNet, FF++, DenseNet with Bidirectional RNN, and FT). Below, we explain how these baselines are implemented.

A.4.1 Xception. We used the Keras implementation of Xception, which is pre-trained on the ImageNet dataset.

A.4.2 ShallowNet. We implemented ShallowNetV3 based on the specification provided by Tariq *et al.* [38] The architecture of ShallowNetV3 is shown in Table 4. ShallowNetV3 consists of 4 blocks with 33 layers. One row in Table 4 correspond to one block. We implemented this architecture using Keras and TensorFlow as backend.

A.4.3 ForensicTransfer (FT).. We implemented ShallowNetV3 based on the specification provided by Cozzolino *et al.* [10] The architecture of the ForensicsTransfer (FT) model is shown in Table 5. We implemented the FT model using PyTorch v.1.2.0. We tried our best to mimic the original performance of the baseline.

[1]https://github.com/ondyari/FaceForensics

A.5 Data preparation for CLRNet

For dataset preparation, we randomly extracted 5 consecutive frames from each video and used them as one sample. We extracted 16 samples from each video. The video whose frames are used in training has not been used for validation or testing. We cropped the faces from the frames using MTCNN [46]. We used our VideoDataGenerator to load these samples and feed them to CLRNet.

A.6 Data Augumentation

In Keras, the data augmentation feature can be used with the ImageDataGenerator. As we have implemented the VideoDataGenerator, we had to implement our own DataAugumentor as well. Therefore we used the Keras ImageDataGenerator Interface and extended it to VideoDataGenerator. The transformation setting we used for data augmentation are as follows: rotation_range=30; brightness_range = [0.7,1.0]; channel_shift_range=50.0; zoom_range=0.2; horizontal_flip=True; fill_mode='nearest'.

A.7 Training of Models

We used the Adam optimizer with the following setting: lr = 0.00005; beta_1 = 0.9; beta_2 = 0.999; epsilon = None; decay = 0.0; amsgrad = False. For the loss function, we used the Binary Cross-entropy. For every experiment, we trained our CLRNet and baseline models for 100 epochs. The best epoch based on the validation loss is used for evaluation with test datasets. We kept the same number of real and fake samples in training, validation, and test datasets. For the training of baseline methods, we used the same training and testing methods as specified in their original paper.

A.8 Transfer Learning

We freeze the first 120 layers of our CLRNet model during transfer learning. There is a small problem in Keras regarding the batch normalization layer when it comes to transfer learning. We solved this problem by using the solution provided by Vasilis Vryniotis[2] and Oleg Gusev[3]. For transfer learning, we used 150 epochs for each experiment, while keeping all other conditions the same. The number of datasets used to transfer a specific target is ten videos, and if there are two targets, ten videos from each target are used.

A.9 Source Code

We plan to release the source code for the implementations of CLRNet, our experiments, and ShallowNet and ForensicsTransfer (FT). If our paper gets accepted, we will clean up the code and share it our GitHub. The source code also contains Juypter Notebooks of all the experiments along with their results.

[2]http://blog.datumbox.com/the-batch-normalization-layer-of-keras-is-broken/
[3]https://github.com/keras-team/keras/pull/9965#issuecomment-549064001

Table 4: ShallowNet Architecture. We used the following ShallowNetV3 architecture to develop this baseline model. Each row represents a block in the architecture. An L2 kernel regularizer of 0.0001 is used in each Conv2D layer.

ShallowNetV3
Conv2D – ReLU – Dropout – Conv2D – ReLU – Dropout – Conv2D – ReLU – MaxPooling – Dropout
Conv2D – ReLU – Dropout – Conv2D – ReLU – Dropout – Conv2D – ReLU – MaxPooling – Dropout
Conv2D – ReLU – Dropout – Conv2D – ReLU – Dropout
Flatten – Dense – ReLU – BatchNormalization – Dropout – Dense – Sigmoid

Table 5: ForensicsTransfer (FT) Architecture. We used the following architecture to develop the FT baseline model. Each Conv2d inside the Encoder part has a stride of 2, whereas each Conv2d in the Decoder part has a stride of 1. We used up-sampling before each Conv2d layer inside the Decoder.

ForensicsTransfer (FT)	
Encoder	Conv2d – ReLU – Conv2d – BatchNorm2d – ReLU – Conv2d – BatchNorm2d – ReLU – Conv2d – BatchNorm2d – ReLU – Conv2d – BatchNorm2d – LReLU
Decoder	Upsample – Conv2d – BatchNorm2d – ReLU – Upsample – Conv2d – BatchNorm2d – ReLU – Upsample – Conv2d – BatchNorm2d – ReLU – Upsample – Conv2d – BatchNorm2d – ReLU – ConvTranspose2d – Tanh

Table 6: This is the detailed architecture of the CLRNet model that we used in this paper. All of these layers are available in the Keras library. The input size of the image is 240x240.

Convolutional LSTM based Residual Network (CLRNet)
ConvLSTM2D – BatchNorm – ReLU – MaxPooling3D
ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ConvLSTM2D – ReLU – BatchNorm – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ConvLSTM2D – ReLU – BatchNorm – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ConvLSTM2D – ReLU – BatchNorm – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ConvLSTM2D – ReLU – BatchNorm – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – Dropout – BatchNorm – ReLU – ConvLSTM2D – **Add**
BatchNorm – GlobalAveragePooling3D – Dropout – Dense