CS230

Video and audio deepfakes detection using Deep Learning

Dong Bing
Stanford University
bingdong@stanford.edu

Luke Kim
Stanford University
mkim14@stanford.edu

Sergei Petrov
Stanford University
spetrov@stanford.edu

Abstract. *This work addresses a deepfake video and audio recognition task using a variety of Deep Learning techniques. In this study we investigated several architectures featuring CNNs, LSTMs, Xception networks and compared their performance.*

1. INTRODUCTION

Deepfake techniques, which present realistic AI-generated videos of people doing and saying fictional things, have the potential to have a significant impact on how people determine the legitimacy of information presented online. In this project we are focusing on using machine learning techniques to detect deepfakes using data provided by Kaggle as a part of a challenge [1].

2. RELATED WORK

Deepfakes recognition is a relatively new topic and there are still no robust solutions that allow to identify either audio or video fakes with enough confidence. Still there are some works that were done in this field that we referred to. The idea of using a CNN+LSTM architecture was borrowed from [2]. We also referred [3], [4] to build Xception+SPP/DSP+FWA architecture that turned out to be the most promising one.

3. DATASET AND FEATURES

We used datasets provided on Kaggle [1]. There are 4 groups of datasets available:

- Training set: The complete dataset, containing labels for the target. There are 470 Gb of archived videos in it. There are 119,146 training videos in total, out of which 99,992 are fake and 19,154 are real.
- Sample sets: There is a small dataset of 400 labeled videos directly available from any notebook within the challenge, we used it for testing purposes. Also there is a set of 400 unlabeled videos that are used to create an output submission table.
- Public test set: This dataset is completely withheld and is what Kaggle's platform computes the public leaderboard against. When notebook is committed, the code is re-run in the background against this dataset.

All videos are of the same length, resolution and frame rate, what makes them convenient to use.

4. METHODS

As all the current neural network architectures may only work with image data and not videos, first we chose the number of frames to extract from each video. We ended up extracting 40 frames per video as it seemed not to deteriorate networks training process and at the same time was not too time consuming to extract them. We also tested different face extracting libraries and chose MTCNN from facenet-pytorch as the one featuring the best accuracy/time consumption ratio. We chose 244x244 as the exctracted face images resolution. Eventually we used only extracted faces/stacks of them for training and prediction, as it is the area of interest for our task.

In our experiments we used the following deep architectures:

- ResNet-50 is a 50-layer deep convolutional neural network that is trained on more than a million images from the ImageNet repository. Each layer follows the general pattern of convolutional layers, batch normalization and down sampling with slight variations and the output layer is a fully connected layer for the weight and the bias, which we have replaced with the updated dimensions.
- VGG16 is another traditional deep architecture based on 16 convolutional layers pretrained on ImageNet
- CNN+LSTM was build based on the code from a GitHub repository [3]. It consisted of several blocks of layers, each in its turn consisted of two convolutional layers followed by batch norm and dropout layers with max pooling at the end. The network head included an LSTM and a FC layer followed by another batch norm. Each layer preceding an LSTM was time distributed, meaning that it was applied to every time step in an input

example. Input to the network was 4-dimensional, with the 4th dimension corresponding to time. The shape of an input was (40, 244, 244, 3)

- Xception (or extreme inception) architecture implementation is based on the original paper [4]. It includes 36 convolutional layers that are structured into 14 modules all of which have linear residual connections around them, except for the first and last modules. The idea is based on depthwise separable convolutions usage. First, to each of the input channels 3x3 convolution is applied, extracting spatial information from each channel independently. Then, 1x1 convolution is applied to gain information from a cross-channel dimension [4]. Input again consisted of 40 frames per video, but it was3 dimensional with all the frames stacked on top of each other, forming an array of the shape (244, 244, 120)
- Another form of input data was used for Xception, which was to use single frames from the videos instead of stacking the frames together. We extract 5 equally spaced frames from videos that are labeled REAL and we apply facial warping artifacts (FWA) as explained in [5] for negative data generation. To summarize, given an image of a face, the FWA captures the face features using dlib CNN face detector then it blurs the captured face feature and affine warps it back to the original face image. This procedure emulates deep-fake faces and it is much simpler to generate compared to using autoencoders. Also as described in [5], training the NN on FWA generated images improved the performance of detecting deepfakes. The figure on the right roughly shows the FWA negative data generation procedure. We processed 5286 real faces and generated 5186 fake faces using FWA and we used these data for training Xception.

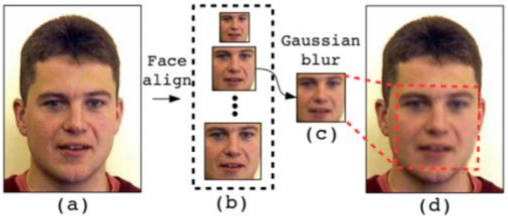


Figure 2. *Overview of negative data generation. (a) is the original image. (b) are aligned faces with different scales. We randomly pick a scale of face in (b) and apply Gaussian blur as (c), which is then affine warped back to (d).*

- Note that for both stacked and single frame Xception models, we are only training on videos that have only a single face. This is because we realized that if a video contains more than one face, and if one person has a fake face and the other person is real, the entire video will be labeled fake, but we will be feeding in both the real and fake faces into the neural network and this may cause confusions during the training phase.

5. EXPERIMENTS AND RESULTS

We started with several simple models to use th*em* for benchmarking.
First, Logistic Regression and SVM (with linear kernel) were implemented. Unlike later models that used MTCNN, OpenCV was used to detect faces from the extracted frames. The full train set was split into train and validation set with stratification to maintain an equal proportion of fake and real labels. Grid search was used to tune the hyperparameters of Logistic Regression, and the experiment was conducted with 255 epochs and a learning rate of 0.001 using binary cross-entropy evaluation metric. Logistic Regression yielded a train accuracy of 83.25% and a validation accuracy of 83.33%. SVM did worse than Logistic Regression, ending up with a 67% validation accuracy. We use flattened images for training and we likely lose information in the pixels from flattening.

Next, we tried a ResNet-50 architecture with a binary cross-entropy loss as the evaluation metric. Ideally, we wanted to run multiple experiment by freezing various different layers and comparing the result, but given the time and computational constraints, transfer learning approach was implemented in two scenarios: (1) freeze all layers prior to layer 3.0 and (2) freeze all layers prior to layer 4.0. In addition, we set the learning rate to 0.01 with zero weight decay and set the number of epochs to 10.
We also performed finetuning of VGG16 alongside ResNet-50. In this experiment we tried to change the number of hidden units from 1024 to 4096 in the first fully connected layer and change the number of trainable layers from 0 to 4 as well. The results obtained with different parameters look similar and the batch accuracy and loss graphs are shown in the reference section for finetuning with 1024 hidden units in the first fully connected layer and all the original layers of VGG16 frozen. In case of finetuning with VGG16, the training accuracy level reaches 0.8 in the beginning of the training and remains flat. Loss seems to behave similarly – decreases in the beginning of the training and then fluctuates.
We trained a simple CNN from scratch to see how it compares to VGG16. We tried two architectures shown in Figure 1, where every blue rectangle corresponds to a convolutional layer and every green one to a max pooling layer. Every convolutional layer is followed by a batch norm layer and a dropout layer with the dropout rate of 0.3. At the end of both networks there are two fully connected layers with 120 and 84 hidden units. The first

architecture performed better, and its batch accuracy and loss throughout 7 epochs of training are shown on the Diagram provided in section 8 of the references below. Again, after the 1st epoch the network seems to make no progress, and its training accuracy reaches a plateau at around 0.8.

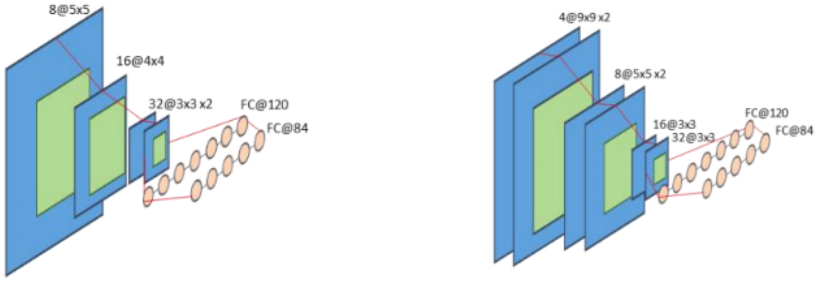


Figure 1. CNNs architectures. Parameters of convolutional layers are shown in the following format: 8@5x5 notation means we have 8 filters of size 5x5.

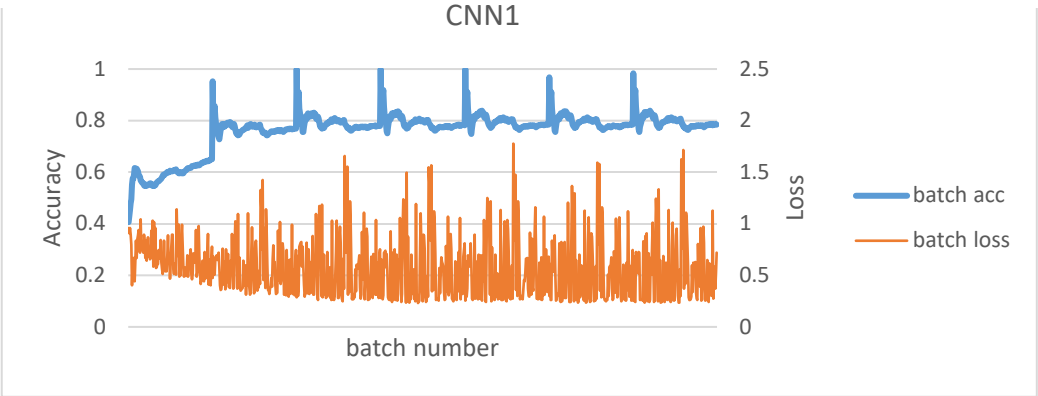


Figure 2. . History of CNN 1 batch accuracy and loss

In the Figure 2 the trining process of a CNN 1 is shown. Validation accuracy reached 75%. Next experiment involved training CNN+LSTM architecture to try to leverage temporal nature of data. In the process of tuning hyperparameters we activated and deactivated convolutional blocks described in the section 4. We also changed filter sizes in convolutional layers, number of nodes in the FC layer and dropout keep probabilities. The architecture that was chosen is shown on the

Figure 3. Each convolutional layer is followed by batch norm and dropout layers, all the layers before LSTM are time distributed.

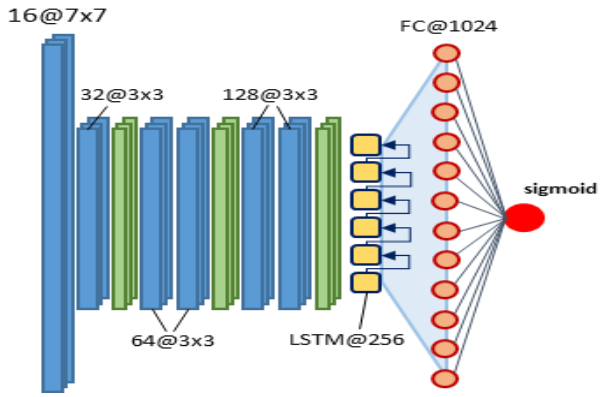


Figure 3. CNN+LSTM architecture. The same notation is used: before @ is the number of filters, after – the filter size / number of hidden units

On figure 4 below, training progress is shown. Again, training accuracy reaches plateau at around 0.87, what is better, than for CNN architectures described above, but the algorithm still cannot reach a satisfactory accuracy value. It may be related to the sequential nature of LSTM – it is good at recognizing sequences (i.e. classifying

different activities in videos), but any actions in our training examples are completely unrelated to their labels. Videos can be real or fakes regardless of whether actors move their heads, turn, walk or remain motionless. Validation accuracy reached 85% at the end of training.

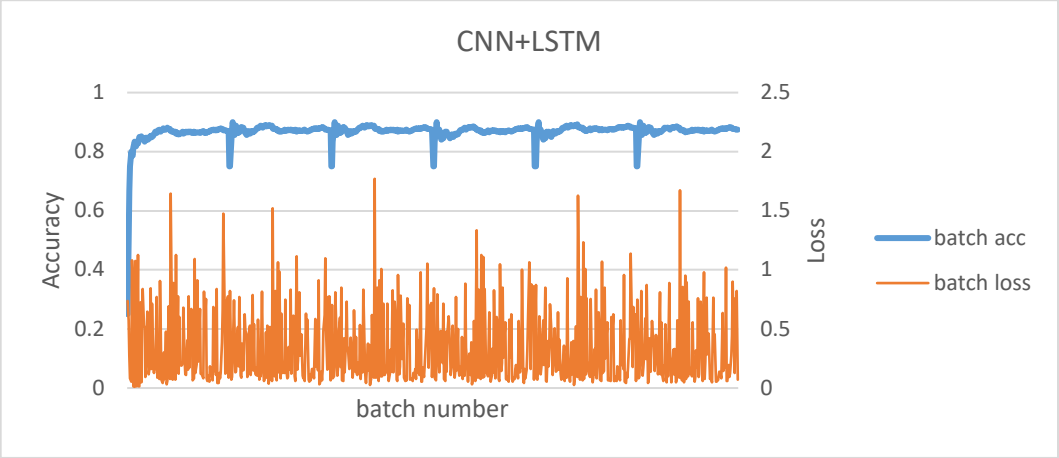


Figure 4. History of CNN+LSTM batch accuracy and loss

Our final experiment involved using an Xception architecture. We reproduced the architecture from the original paper [4] in such a way we can tune the hyperparameters and played a bit with the architecture itself. The Xception we implemented is much deeper (with skip connections) than the proposed architecture in figure 5 and it has been optimized for better training. We have implemented the Xception such that we can configure hyperparameters like the number of residual blocks, the filter size, the number of filters being used, the stride etc. There are many hyperparameters and these can be inspected from the github code. Instead of regular Xception, temporal Xception might also be a good option, but we believe using stacked faces would help capture temporal correlations in the video even with regular Xception.

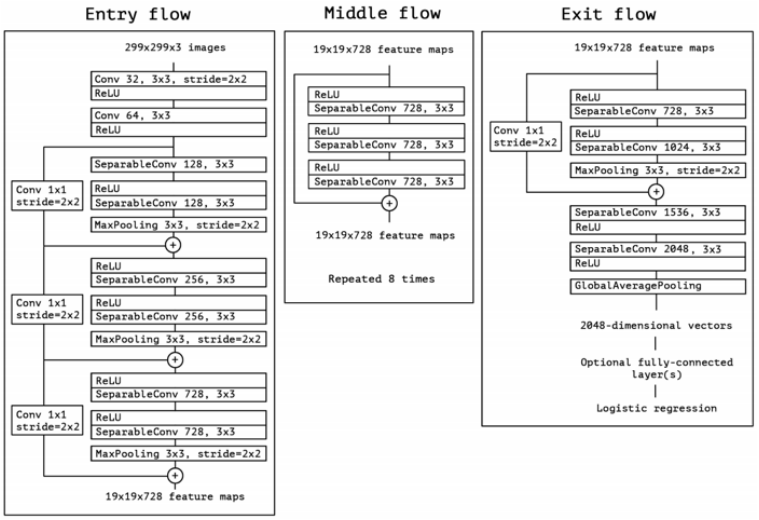
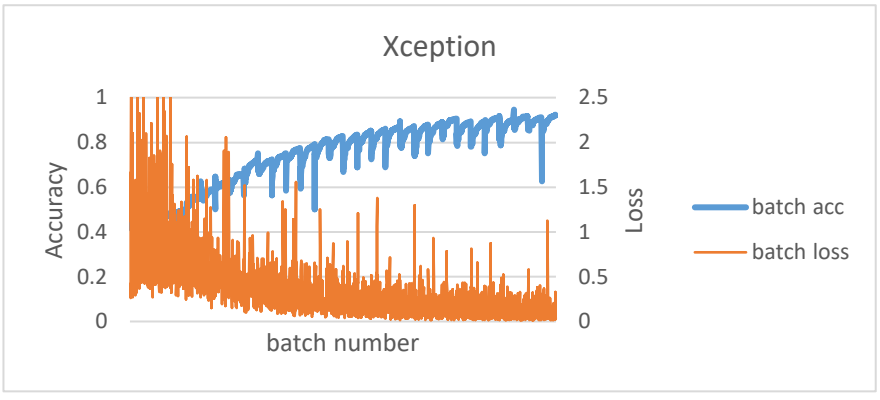


Figure 5. Xception architecture.

When tuning hyperparameters for Xception with stacked data, we changed numbers of convolutional filters in each of the block except exit flow, number of blocks, learning rate, optimizer, added fully connected layers, added and removed MaxPooling layers, removed BatchNorm layers, introduced dropout layers with different keep probs, added l2 regularization, changed activation functions in fully connected layers. Eventually we managed to fit the training set perfectly, but the maximum validation accuracy achieved was around 80%.



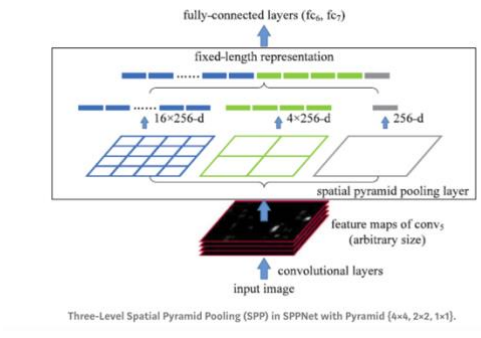
When training the Xception for single face data with FWA negative data generation, we carried out 5 experiments, each time training roughly 2000 samples on 1000 epochs using GPU using Paperspace (training the entire data set of 10472 images created an OOM error). For each experiment, after the 1000th epoch the training

and validation losses were close to zero (for the 5th experiment, the training loss was at 4.07e-05 and validation loss was at 7.47e-05 after 1000 epochs) and training and validation accuracy was 100%. This clearly shows that Xception is very good at differentiating real faces and FWA generated faces. However, FWA generated data may not be enough for covering all the different types of deepfake on the Kaggle dataset, so we may have to augment the real+FWA face data with more examples of fake from the Kaggle public data set.

6. NEXT STEPS AND THE BIG PICTURE

So far we have only submitted a pretrained Xception and a Resnet-50 ensemble on Kaggle, which resulted in a loss of 0.44 on the Kaggle test data. Our aim is to also test our Xception models on the Kaggle test data and see how it performs. These are the steps we will take:

1. For the stacked data we currently have 6000 train data. We can process more data for training. Also, as mentioned in section 4 methods, the Xception is trained on only videos with single faces, however when testing we would also have to test for videos with more than one face. In this case we would have to create multiple stacks with the same face and in order to do this we need to develop some face tracking mechanism to ensure that each stack contains the face of the same person. A few methods we thought of was to discern the peoples' faces using the distance of the detected face from the leftmost border of the frame, or by calculating the difference between the detected position of the face of the current frame and the position of the face in the previous frame.
2. For Xception with single frame we know that it does well on discerning FWA generated faces and real faces. However, before making the submission to Kaggle we would have to augment the data set by also containing more fake faces from the actual Kaggle train set, because the fake faces we have for the single frame model are the ones generated from real faces only. Also, FWA can be improved as currently it affine warps back the rectangular region of the face as shown in the figure in page 2, but it can be implemented to more closely warp around the face by computing the convex hull around the facial landmarks - shown in more detail here: https://github.com/iljimae0418/CS230-deepfake/blob/master/FWA/advanced%20fwa.png. We can also implement face alignment on some faces that appear heavily skewed.
3. We can implement a higher assessing net (HAN). Typically we would get predictions from different models and average their results, but instead of this why not create a neural network that can selectively decide which model to trust on more given the input data? Then we have HAN which is a network that takes the input video itself as well as the predictions from the models as the input and the output node of HAN would give an actual probability of the video being fake. We can use multiple Xception (both stacked and single frame model) with different hyperparameters as well as the pretrained Xception+Resnet-50 ensemble that we used to submit to Kaggle, as well as the CNN+LSTM model.
4. Some Kaggle training videos have audio deepfakes as well. In the metadata file for the training video, if the video is labeled FAKE, then its original video name is also listed next to it. However if this column is blank it means that there is audio modification. We can convert deepfake audio detection into an image classification problem: first extract the spectrogram of the audios and because the spectrograms of deepfake and real audios are different, we can pass these images into a CNN for classification. We can create an ensemble with our deepfake face and audio classifier. More analysis can be found here: https://github.com/iljimae0418/CS230-deepfake/blob/master/Audio%20Analysis.docx
5. At the moment, for both the stacked faces and the single face data, we are using an image size of 244x244. However, we can prepare our test data set such that we have variable image size for input e.g. some images may be of dimension 244x244, some images may be of dimension 300x300 and some may be of dimension 600x600. This can be done by using spatial pyramid pooling (SPP) [6]. SPP can be implemented right before the fully connected layer and it applies multiple levels of pooling to create multiple 1-dimensional vectors which are then concatenated before being fed into the fully connected layer. This way any image sizes can be inputted and it does not cause problems of losing information by cropping or resizing images to a fixed dimension.



Three-Level Spatial Pyramid Pooling (SPP) in SPPNet with Pyramid {4×4, 2×2, 1×1}.

7. ACKNOWLEDGMENT

We would like to thank Kevin Kim from the University of Toronto for his contribution and ideas that kept our project running. He was an originator of an idea of using a combination of Xception networks with spatial pooling and FWA and a Higher Assessing Network. He made huge impact on the team's overall progress.

8. REFERENCES

[1] Kaggle deepfake detection challenge https://www.kaggle.com/c/deepfake-detection-challenge

[2] Thanh Thi Nguyen, Cuong M. Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen and Saeid Nahavandi Deep Learning for Deepfakes Creation and Detection. https://arxiv.org/pdf/1909.11573.pdf

[3] https://github.com/harvitronix/five-video-classification-methods

[4] Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions https://arxiv.org/pdf/1610.02357.pdf

[5] Yuezun Li, Siwei Lyu: Exposing DeepFake Videos By Detecting Face Warping Artifacts. https://arxiv.org/pdf/1811.00656.pdf

[6] Sik-Ho Tsang: SPPNet. https://medium.com/coinmonks/review-sppnet-1st-runner-up-object-detection-2nd-runner-up-image-classification-in-ilsvrc-906da3753679

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition: https://arxiv.org/pdf/1406.4729.pdf

[8] Detecting Audio Deepfakes with AI: https://medium.com/dessa-news/detecting-audio-deepfakes-f2edfd8e2b35

Github: https://github.com/iljimae0418/CS230-deepfake

Contributions:

Dong: trained and submitted Xception+Resnet-50 model to Kaggle, analyzed deepfake audios, did kaggle data analysis, helped with final report and poster.

Luke: trained logistic regression and SVM on single frame data, trained and submitted pretrained Xception and CNN to Kaggle for testing, implemented Xception with Keras, implemented FWA and trained real+FWA faces on Xception on paperspace, implemented face alignment, helped with final report and poster.

Sergei: trained VGG-16, implemented and trained CNN+LSTM, trained Xception with stacked faces, helped pre-process stacked faces as well as single frame faces from real videos using MTCNN, implemented and experimented with deepfake audio detection, helped with final report and poster.