Deepfake Detector

Marc Laugharn

Abstract

I attempted to develop a classifier that took videos as inputs, and predicted whether or not a given video was a deepfake or not. The videos were preprocessed to focus on the faces. The data was augmented at training time, including in a way that exploited the geometry of faces. Various convolutional neural network models were explored as candidates for the classifier backbone.

1 Introduction

The general public is gradually becoming more aware of the existence and potential dangers of deepfake videos. Deepfake videos are convincing forgeries that create the possibility of easy creation of synthetic video in which any person could be made to say anything. There exists a concern that in the future, deepfake videos could be used to spread fake news, or manipulate public opinion. It is crucial for deepfake videos to be detectable.

My Math 189 Big Data final project was to participate in the Kaggle "Deepfake Detection Challenge" competition. [1]

2 Dataset

The competition dataset was downloaded via the Kaggle API. It consists of 400 training and 400 test set videos. Each video is approximately 10 seconds long, at 30 frames per second.

Each video has the label 'REAL' or 'FAKE'. There are about 5 times as many fake videos as there are real videos. Each fake video was created from one of the real videos in the dataset.

The data was split into a train set and a validation set. In the end, my final choice of the split was 80% train, 20% test. The competition creators announced that some videos had synthetic audio, but I only analyzed the audio content of the videos.

3 Insights

3.1 Failed approaches

I initially wanted to use a pre-existing face detection classifier (MTCNN [4]) and extract the embedding vectors it generated from each face, and then perform a test to check if the embedding vector appeared to be nearer to a 'fake' embedding cluster or a 'real' one using a Support Vector Machine. This approach failed because embeddings are supposed to be resilient to small changes in the data.

In the end I decided to train convolutional networks to directly classify the images from the videos.

Initially, when sampling a video's frames, I used the entire frame to predict its class. This method failed because the network was not focused on learning the most crucial aspects of problem, namely whether the faces seemed real or not.

3.2 Final strategy

It was crucial to keep the focus of the problem on the actual relevant information, namely the appearance of faces. Exploiting the geometry of the face would allow for better focus during training.

I decided to use cropped face regions as the input to my convolutional neural network.

Then I tried using several pre-existing, pre-trained convolutional neural networks as the backbone for my model. I tried ResNet18, ResNet34, EfficientNet-B3, before finally using EfficientNet-B5 as the backbone for my model.

4 Preprocessing

I extracted 15 frames from each video. I used MTCNN to locate faces in each frame. If there was greater than 90% confidence that a region contained a face, I included it. I expanded the margin of the bounding box around the face by 30%, and resized the faces to 300 by 300 pixels.

I also removed some clearly incorrect faces from the faces dataset (e.g. faces on tshirts, drawings of faces, hands that were misclassified as faces) to try to clean the resultant auto-generated dataset.

There were also instances where fake videos featured 2 people, one of whom was real. I tried my best to identify cases where I could determine which person was the real person, and to remove them from the fake person set.

5 Augmentations

Because image classifiers need a huge amount of training examples, I incorporated training time augmentations to increase the number of examples. [2]

I made sure to use augmentations that did not make classification more difficult, eg. no warp, as it would reduce the ability to detect a face in a frame.

I also developed an augmentation that allows for placing greater attention on the face.

5.1 Face Feature Mask

Face Feature Mask is a custom data augmentation I developed that exploits the geometry of facial structure. I used the dlib [5] machine learning library along with a pre-trained facial feature landmark extractor dataset. It detects 68 standard landmarks in faces: the mouth, the right eyebrow, the left eyebrow, the right eye, the left eye, the nose, and the outline of the jaw. Each facial feature corresponds to an interval of indices of the computed points, e.g. the mouth is given by the set of points in the returned data between indices 48 to 68.

My Face Feature Mask augmentation algorithm works as follows:

1. From one of 3 choices chosen at random with equal probability, create an image mask from:
   (a) the convex hull of the entirety of facial features (i.e., indices 0 to 68)
   (b) the union of the 3 convex hull masks from a set of 3 randomly chosen facial features
   (c) the convex hull of a random interval of the landmarks, where the first index lies in [0, 34) and the second index lies between [34, 68)
   (d) We produce a stencil by expanding the mask via a dilation of random radius (between 10 and 60 pixels) applied a random number (between 1 and 3) times
   (e) Take the bitwise AND of the original image with the dilated stencil

This augmentation creates images that are black everywhere except for on key facial landmarks. Compared to merely training on bounded boxes that contain faces, this augmentation makes the classifier place much more importance on facial features.

The triple of facial features mode allows for the contrasting of facial regions that appear different. The random interval mode makes sure that areas between facial features are also queried. The dilation allows for the inclusion of the facial boundary, and greater ease of localization of cropped facial features.

The augmentations I used were applied at random, and together. They were

- Crop 0-100% of width and height, with reflection, with prob. 100%
- Flip left-right, with prob. 50%
- Rotate -10 to 10 degrees, with prob. 75%
- Scale width and height, independently, between 0-100%, with prob. 100%
- Change brightness by 0.4 to 0.6, where 0.5 is no change, with prob. 75%
- Change contrast by 0.8 to 1.25, where 1 is no change, with prob. 75%
- Blur with up to 3px radius, prob. 50%
- Jpeg Compression from 100% to 65% quality, with prob. 35%
- Facial Feature Mask, with prob. 33%

6 Model architecture

I settled on using the EfficientNet architecture [7] for my model base. I replaced the output layer with a small feed-forward network. The final layer of the EfficientNet-B5 architecture is a 2048 node to (number of classes, i.e.) 2 node fully connected layer.

I replaced this fully connected layer with a 2048 to 1024 fully connected layer with a ReLU activation, which was then fully-connected to a 2 node layer. I thought this nonlinearity and greater size would improve the model's ability to learn to map images to the final 2 classes better than a single linear layer.

7 Training process

I oversampled the data so that both the fake and real class would be presented with equal probability.

Before beginning training, I tried to estimate a good learning rate by using the fast.ai framework's learning rate finder. It plots the initial loss as a function of learning rate. A good rule of thumb is to use the learning rate in the middle of the highest learning rate interval where the loss appears to slope down quickly. For most trials, 1e-3 was a good initial learning rate.

I incorporated discriminative learning rates [6], which is where you use a different learning rate for different layers in the neural network. I set the learning rate for the final layer (after the EfficientNet base) to be 1e-3, and the learning rate for the rest was 1e-4. This way the pretrained base could be adapt to the deepfake detection problem without throwing away much of its pre-existing capacity for general purpose image recognition.

For the first round of training, I used a cyclic learning rate for a maximum of 20 epochs [3]. The cyclic learning rate method has been shown to increase the speed of training. The learning rate starts low, increases rapidly and reaches its maximum about 20% of the way through epochs, and then gradually reduces for the remainder of training.

I monitored the validation loss, and stopped the training early if there was no improvement in validation loss for 5 epochs. I also measured the training loss and validation accuracy. The maximum observed accuracy I achieved across all my attempts was 95%.

I then made the first layers untrainable (known as 'freezing' the early layers), and trained the final layers with a learning rate of 5e-4 for 5 epochs.

8 Evaluation

In general, the network was able to learn to classify fakes quite accurately. The confusion matrix showed 95% of predicted fakes were actual fakes, versus 78% of predicted real images were actually real images. Of the images it thought were fake, 22% of the time they were actually real, and only 5% of the time was an image predicted as real actually a fake.

The histogram of losses in the validation set also shows that 92% of the images in the validation set had a loss less than 1. The tail of the distribution went all thye way to 14, though, which suggests the loss was dominated by severely overconfident incorrect predictions.

Ultimately, the quality of competition submissions was supposed to be measured in terms of the log loss of predictions, but I was unable to create a good competition entry until after the public leaderboard's deadline. The true labels of the test set have not yet been released, so it is impossible for me to compute my log loss score at this time.

9 Conclusion

By far, the greatest improvement to my model came from crafting geometry-awareness into the classifier, by cropping to faces, and performing face-aware augmentations. I actually was surprised it ended up performing as well as it did. In the process of creating this model, I learned a lot about deep learning, image classification, and machine learning best practices, and gained a lot of practical experience in creating and working with machine learning models.

References

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

[2] The Effectiveness of Data Augmentation in Image Classification using Deep Learning. http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf

[3] Cyclic Learning Rates for Training Neural Networks. https://arxiv.org/abs/1506.01186

[4] MTCNN https://github.com/ipazc/mtcnn

[5] dlib https://github.com/davisking/dlib

[6] Discriminative layer training https://docs.fast.ai/basic_train.html#Discriminative-layer-training

[7] EfficientNet https://arxiv.org/abs/1905.11946