Deepfake detection challenge
Biometric Systems Project - Sapienza University of Rome
Academic Year 2019-2020

Shadi Andishmand

Abstract

The Deepfake Detection Challenge is a Kaggle competion hosted by AWS, Facebook, Microsoft and the Partnership on AI's Media Integrity Steering Committee with the purpose to enhance the research in the detection of deepfakes and manipulated media (Kaggle, 2019). Started in December 2019 and lasted three months, it saw the participation of more than 2000 teams from all over the world both from academia and industry. One of the main contributions of this competition to the media manipulation detection research area is the large dataset of real and fake videos that is now available to the research community. This dataset is an improvement over other recently released ones for its data diversity, agreement from persons depicted in the videos and size of dataset, among others. During the course of the competition our team tried several solutions and run many experiments that led us to the the top 13% in the public leaderboard, but, due to the severe overfitting that almost all the top teams faced over the public test set, we only reached the top 42% in the private leaderboard. Our final solution is an ensemble of three EfficientNetB7 and one Xception net that detect the presence of manipulation in single frames of the videos, without any temporal information. We leave the explanation of our design choices to the following sections.

1 Introduction

The term "DeepFake" refers to a deep learning algorithm able to create fake videos by swapping the face of a person by the face of another person. It consists in two autoencoders with a shared encoder that are trained to reconstruct training images of the source and the target face, respectively. A face detector is used to crop and to align the images. To create a fake image, the trained encoder and decoder of the source face are applied to the target face (Rossler et al., 2019). This algorithm called Faceswap was published in 2017 on Reddit and it gave life to a new wave of face manipulation techniques. In the following months and years many other algorithms emerged making use of more sophisticated deep learning techniques like GANs, and the face manipulation extended to areas like face2face expression transfer and face generation. Along with fake generation techniques that can be used for malicious purposes, we saw the spreading in the cultural mainstream of easy to use apps that do not require any knowledge of the underlying algorithms.

The worry is that deepfakes might become a powerful weapon for political misinformation, hate speech, or harassment, spreading virally on social media platforms. Companies like Facebook and Google have already started to receive widespread pressure to tackle these new emerging problem. Along with fake voice generation and fake news generation, this one represents a new emerging frontier that we are only starting to see applied for malicious purposes and, just like with any other cybersecurity challenges, the fight will go on for many years. Face manipulation becomes better every day and its going to get harder and harder to detect fake face in not only images but also in videos. This challenge represents one of the most remarkable efforts on the industry side to show how seriously they are taking the concerns of their users who recently saw the spreading on many social networks of videos of politicians giving fake speeches and the use in many revenge porn cases. (Tolosana et al., 2020)

1.1 Challenges of the problem

Some of the challenges that we face in a problem like this are the same that we find in the face detection and recognition realm, so we have to consider that, even if over the last decade computer

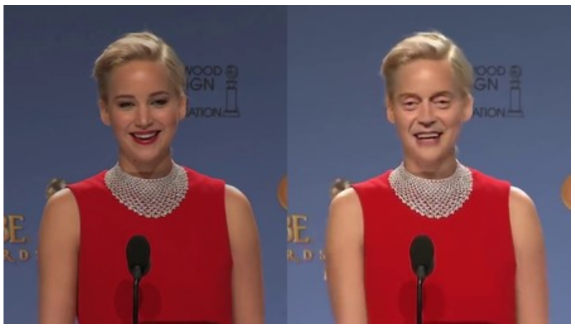


Figure 1: Example of deepfake.

vision improved the performance of automated face recognition algorithms on many tasks, it is unclear whether the use of algorithms improves security or puts it at greater risk, given the radical shift in the approach taken. If in face detection and recognition we have to handle all those scenarios where subjects are non-cooperative, for now in deepfake detection we only have to handle cooperative scenarios. We only inherit from them the usual problems like the causes that make the appearance of a face to vary like ageing, facial expression, cosmetics, glasses, and so on, (for differences in the same person), or ethnicity and gender for differences between two individuals. Beyond these we also have to handle changes in the illumination, pose, scale, resolution, noise and a few others.
Varying the illumination can result in larger image differences than varying either the identity or the viewpoint of a face. Pictures of the same person with the same camera and seen with the same facial expression and pose may appear really different with changes in the lighting conditions. In our dataset this issue was really important since we are extracting from videos in different conditions and lightning. As you can see from the frames in Figure 2, which are from a same person, some videos have low illumination and consequently the captured frames are very dark so detection is a little harder.
The pose of a face changes with the angle of the camera and rotation in the head position so, given that often the actors are moving, the many different poses also affect the face detection in many videos. As for facial expression there isn't much variety for the same individual given that the actors are either reading from a script or talking to other persons and all the videos are taken in the same session.

2 Dateset

The dataset was presented two months before the starting of the competition (Dolhansky et al., 2019). From this paper we know that:

- Videos are generated using two facial modification algorithms that are not disclosed in order to encourage the participants to come up with solutions that are independent of the manipulation algorithm
- The real videos where collected from actors that agreed to participate in the creation of the dataset which uses and modifies their likeness. They vary in gender, skin tone and age.
- The videos include different lighting conditions and head poses, and participants were able to record their videos with any background they desired.
- The rough approximation of the general distribution of gender and race across this preview dataset is 74% female and 26% male; and 68% Caucasian, 20% African-American, 9% east-Asian, and 3% south-Asian.
- The first algorithm used does not produce sharp or believable face swaps if the subject's face is too close to the camera, so selfie videos or other close-up recordings resulted in easyto-spot fakes.
- The second method generally produces lower-quality swaps, but is similar to other off-the-shelf face swap algorithms.
- After the manipulation, every video was subjected to at most one of the following augmentation: reduce the FPS of the video to 15; reduce the resolution of the video to 1/4 of its original size; reduce the overall encoding quality.

The dataset was split in 4 groups of videos. The first two are publicly available and are made of videos 15 seconds long.

- Training Set of 119154 videos (470 GB).
- Public Validation Set of 400 videos
- Public Test Set of 4000 videos completely withheld and is what Kaggle's platform computes the public leaderboard against

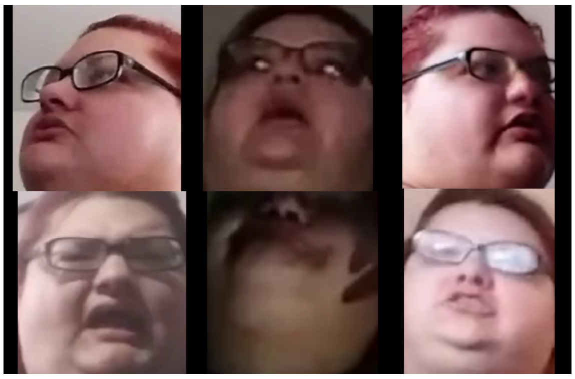


Figure 2: Actors took videos under very differnt conditions.

- Private Test Set privately held outside of Kaggle's platform, and is used to compute the private leaderboard. It contains videos with a similar format and nature as the Training and Public Validation/Test Sets, but are real, organic videos with and without deepfakes.

2.1 Evaluation

Submissions are scored on negative log loss:

$$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

The use of the logarithm punishes for being both confident and wrong. In the worst possible case, a prediction that something is true when it is actually false will add infinite to the loss. Consequently, predictions are bounded from the extremes by a small value.

3 Deepfake Detection Pipeline

The approach that we followed from the very beginning was to establish a working pipeline made of a few almost fully independent blocks that can be replaced and improved at will. Being our first experience with a very large dataset we give a few details on how to handle this kind of problem in every aspect.

3.1 Downloading the dataset

In December the Amazon Web Services team (one of the sponsors) gave each participating team 800$ in AWS credits to start processing the training dataset and train the models. Being a few weeks late to the competition we were not able to receive this first tranche of credits, so we bought an SSD Hard Disk of 2 TB to download the entire dataset and avoid any possible problem related to space. Refer to Appendix A for the source code. We decided to extract the frames on our local machine and used the second tranche of credits received in January to do the face detection which required access to a powerful GPU in order to speed up the process.

3.2 Frames extraction

During the first few weeks we decided to run a few experiments with several machine learning and deep learning techniques in order to learn new algorithms and at the same time we followed the animated discussions from more experienced users. This aspect of the competition was one of the most interesting for us because it allowed us to learn from real professionals. Initially we thought that it was necessary to use temporal models like LSTMs or GRUs to analyze multiple frames, but given our inexperience with this kind of models we were not able to produce an effective solution. One of the few restrains on the submission is that the inference on the Public Test Set can be at most 9 hours, so frames extraction/face extraction/face detection are the main bottleneck. Along with the majority of the community we decided to predict over the single frames and then averaging the predictions for the frames in one video. At first we tried to extract all the frames from all the videos, which took many hours, but after a while we realized that the there is in not much variance in the frames of these 15 seconds long videos, the actors are almost always still during the video and that some actors appeared much more frequently than others, so we decided to extract only one frame from each video. We used the VideoCapture class provided by OpenCV to extract the frames from the videos.

3.3 Face Detection and Extraction

To detect the faces in the frame we used a pytorch GPU implementation (Esler, 2019) of the MTCNN algorithm (Multitask Cascaded Convolutional Networks) (Zhang et al., 2016). This is an algorithm consisting of 3 stages, which detects the bounding boxes of faces in an image along with their 5 Point Face Landmarks. Some videos contain more than one person and most of the times both of them are fake. Given that the face detection has often one or two false positives, but with a lower confidence than the actual face detected, we preferred to only keep the bounding box with the highest confidence. Given the bounding box, we resize it to 224x224

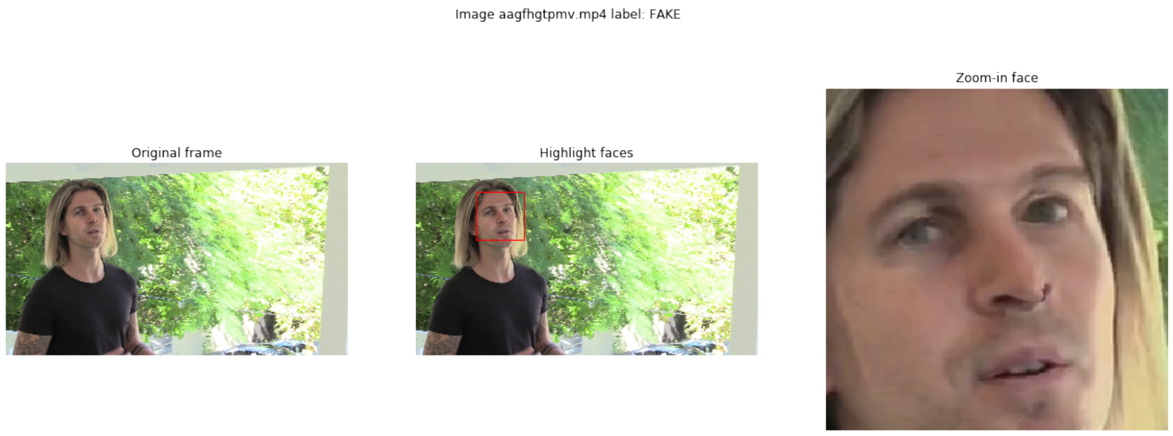


Figure 3: Face Detection and Extraction from fake video. [1]

and add a 0 border where it is needed. Given the importance of the speed in face extraction to keep our inference inside the time constraint, this one revealed itself to be the best approach compared to using an in house solution or more mainstream libraries.

3.4 Training and Validation sets

One of the first things that was required to build a pipeline that could allow us to run many experiments was to have a good validation set. Given the fact that each team was allowed only two submissions (evaluation on the public test set) each day and each of these would require a few hours, having a good correlation between the validation loss and the public test set loss was necessary to design a good model. After a few experiments the public validation set reveled itself to be inadequate mainly because it was too small and because the same actors appeared both in the training and validation sets, causing the models to overfit over the most frequent actors. Given the long training times k fold cross validation was not an option. Since the entire training set was split in 50 parts and the same actor appeared almost always in only consecutive parts and the real video and their corresponding fakes always appeared in the same parts, we decided to use the first 40 parts for the training and the remaining 10 for the validation. Given that the dataset is highly unbalanced (5 fakes for each real) we downsample the validation set by only taking for each real video only one originated fake.

[1] `https://www.kaggle.com/aleksandradeis/deepfake-challenge-eda`

3.5 Data Augmentation

In order to increase the diversity of data available to train the models without actually collecting new data we used a few data augmentation techniques that help us to avoid overfitting and help the generalization. We used the Albumentations library (Buslaev et al., 2020). We do the data augmentation at training times by calling the augmentation routine every time a new batch of images is requested by the training procedure. Each batch contains both the real and its fake and we apply the same augmentation on both of them with the purpose to make the model invariant to this kind of different settings. We apply each of these transformations with a 10% probability (Shift, Scale, Rotate, HorizontalFlip, Cutout, RandomContrast, RandomBrightness, JpegCompression, Downscale, GaussNoise) and then we normalize. We decided to use this library mainly because it is one of the fastest and wide available.

3.6 Transfer learning

One of the most important hyperparameters for the training of a deep learning model is the weight initialization mainly because of the fact that during the stochastic gradient descent (and its derivatives) algorithm we may end up in a local minima or it may take a very long time to find good parameters for our task, so starting from a model that is already able to extract the main features from an image gives a significant speed up to almost all the computer vision tasks. This is why transfer learning using pretrained models over the Imagenet dataset has become the standard in any computer vision task. This is then followed by finetuning over the dataset provided to solve the specific task.

4 Models

Given that the problem involves handling manipulated videos, at the beginning of the competition we decided to try to use recurrent models to spot the fake frames. Only after several days that we were not able to get a working model we decided to switch to single frames prediction using CNN and averaging across a few extractions from each video to get a working solution. Given that we saw the same behavior from the rest of the community, we decided to stick with this approach for the rest of the challenge.

4.1 Experiments with Recurrent Models

Even if Recurrent Neural Networks are mostly used in natural language processing, in the last few years we saw emerging a new interesting model called Convolutional LSTM that we thought could be successfully applied to deepfake detection. Given that Convolutional LSTM are expected to predict frames of a video, a recurrent deepfake detection system might be able to detect a manipulation if the next frame significantly deviates from its prediction. The architecture is composed by two essential components: 1. CNN for frame feature extraction. 2. LSTM for temporal sequence analysis. We obtain a set of features for each frame that is generated by the CNN and then we concatenate the features of multiple consecutive frames in order to pass them to the LSTM for analysis. We finally produce an estimate of the likelihood of the sequence being either a deepfake or a non manipulated video. We tried to implement this architecture using Keras by first extracting the frames from a video, then we use a Conv2DLSTM, which is a variant of LSTM containing a convolution operation inside the LSTM cell, where the matrix multiplication is replaced with the convolution operation at each gate in the LSTM cell. The output is a 2D CNN with one by one kernel which actually optionally take output of just last layer also optionally take the output of complicated model which is CONV2DLSTM output. Then the data which is fed into the final model to train would be the input which are frames and output which we made by CONV2DLSTM so the model will evaluate the next predicting frames based on this two.

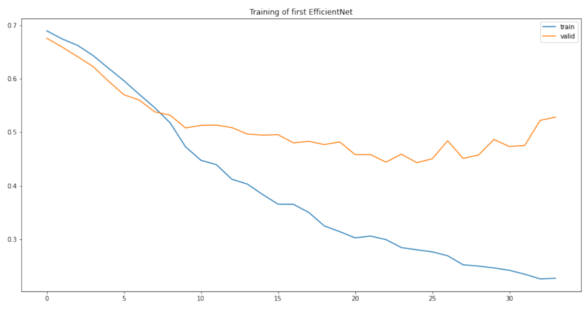


Figure 4: Training of the first EfficientNet.

4.2 Experiments with Convolutional Neural Networks

After being done with recurrent models, we tried to use the two models that we were most familiar with: ResNet and Inception. With this models we were able to make our first submissions. Both of them gave similar results over the public test set getting a log loss at most of 0.52. After a few weeks that we were not able to significantly improve our score, we decided to start making use of some of the techniques that are needed to succeed in this kind of competitions. We started by switching to one of the most recent models scoring the highest accuracy over the Imagenet dataset called EfficientNet (Tan and Le, 2019) pretrained over imagenet. After a few test over the different scales of this architecture we ended up choosing efficientnet-b7. After this we moved on to build an ensemble of multiple models, both more EfficientNets and one XceptionNet. The hard constraint of 1 GB on the total size of the models submitted led us build an ensemble of 3 EfficientNet-B7 and one XceptionNet. In all of them we removed the top layer, added at the top one linear layer with one output followed by a sigmoid activation function to produce the prediction.

4.3 Training

Before starting the training we unfreeze all the parameters of the network in order to finetune across the entire architecture and not only the last layer. The final training dataset is made of 79501 fakes and 14820 reals. We used Minibatch Gradient Descent with a learning rate of 0.001, weight decay of 1r-6 and momentum set to 0.8. We tested different optimizers like Adam and RmsProp, but, even if they were faster to find good solutions, SGD was able to find better ones. We used binary cross entropy as loss function without any label smoothing which, in hindsight, might have helped to avoid

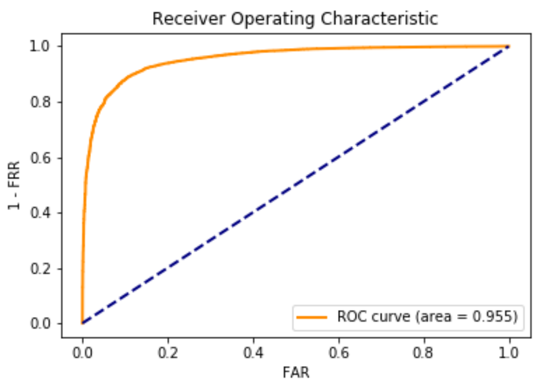


Figure 5: ROC over the validation set.

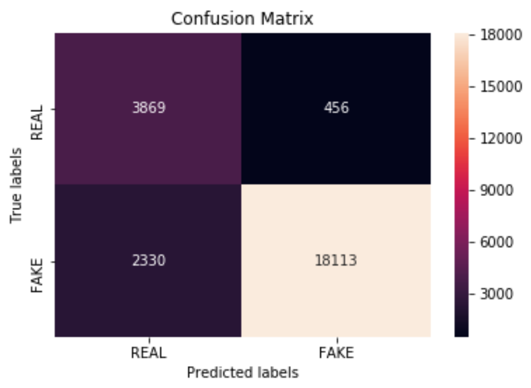


Figure 6: Confusion Matrix over the validation set.

overfitting and spared us from using predictions clipping.
We set the batch size to 16, which is quite small but useful to add some regularization effect since the actors that are less frequent in the dataset are not overshadowed in the single batches by the ones who appear in more videos. In order to balance the the dataset, each batch contains 8 fakes and their respective 8 reals, so one epoch is a run over all the fake images. We monitor the validation loss each 1000 batches and apply early stopping on these intervals rather than on the epochs.
Given the size of the dataset and the extensive us of many data augmentation technique we decided to train all the models over the entire dataset. After a few runs of the training procedure we set a cap to the number of epochs to 10 since the training routine always stops before because of early stopping which has patience of 10 epochs and monitors the validation binary cross entropy. Both during validation and testing we used prediction clipping which made over confident wrong predictions less severe over the loss by restricting the 0-1 range of predictions to a smaller one like 0.1-0.9.

4.4 Validation

As we said before we could not use k fold cross validation, so we relied on the fact that we took a good chunk of images out of our training set in order to have a solid validation set and a good validation strategy overall. During the training routine we undersample the fake class to balance the validation set and reduce the validation time. The reported results instead are over the entire validation set

$$FAR = \frac{FP}{FP + TN} = \frac{456}{456 + 3869} = 0.105$$

$$FRR = \frac{FN}{FN + TP} = \frac{2330}{2330 + 18113} = 0.114$$

The metric that better summarizes the performance of our solution is the Negative Predictive Value, which is the precision over the negative class.

$$NPV = \frac{TN}{TN + FN} = \frac{3869}{3869 + 2330} = 0.624$$

From this we can see that one out of three negative predictions (real) is incorrect.

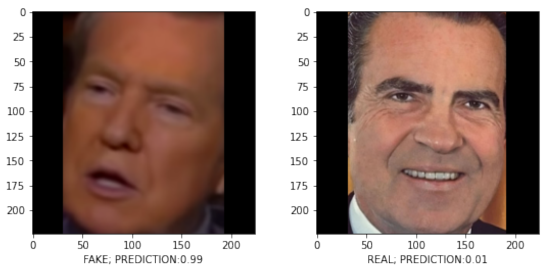


Figure 7: Perfect predictions on high quality reals and low quality fakes.

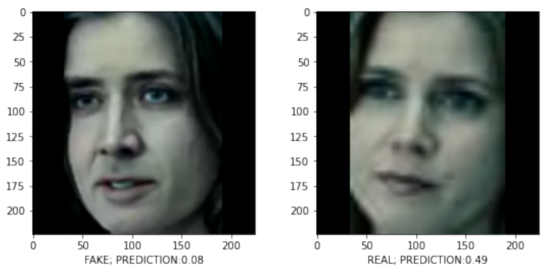


Figure 8: Not very good prediction on good quality fake and bad quality real.

Dataset	Log Loss	Position
Validation	0.29647	——
Public Test	0.42855	278th
Private Test	0.58670	933rd

Table 1: Public and private test set binary cross entropy and leaderboard position.

4.5 Inference

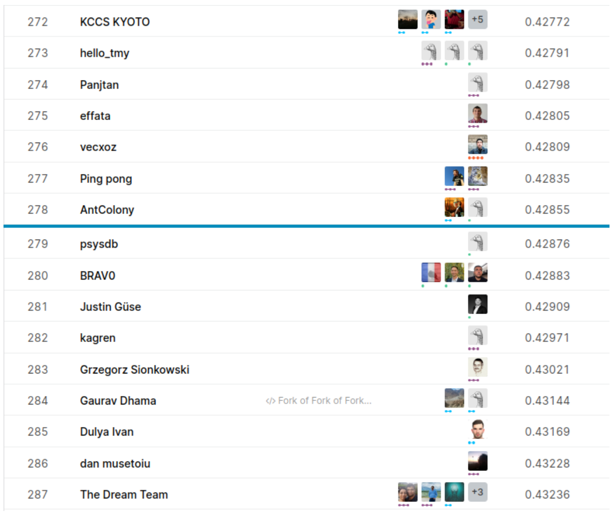
The approach that we followed for the inference over the test set is to average the prediction for each video over all the frames that we could extract by remaining inside the time limit. Averaging over circa 20 frames gave an important boost in the performance compared to only 1 frame, but increasing it even more did not give any further improvement. We use test time augmentation by applying a random illumination with 0.3 probability to each frame mainly because we noticed that it was the only one that leads our models to deeply change its prediction based on its intensity compared to other transformations. As we said before we used prediction clipping which drastically improved our score over the public test set, but it was only one more reasons for the overfitting over the private test set.

5 Results

According to the rules of the competition each participant is allowed to pick two submissions for evaluation over the private test set.We reached the top 13% in the public leaderboad (278th out of 2114 teams) with a public test loss of 0.42855. The severe overfitting to the public test set led to a remarkable reshuffling of the leaderboard which meant for us dropping to the 933rd position staying only in the top 42% with a binary cross entropy of 0.58670. Many of the top 200 solutions saw a similar drop in performance and the top 1 test loss dropped from 0.19207 to 0.42320 which is very close to what we reached in the public test set. Our solution can easily detect visual artifacts, which covers the majority of the test set, but it is not very good at separating the reals from the good deepfakes and for this reason the log loss goes up quickly.

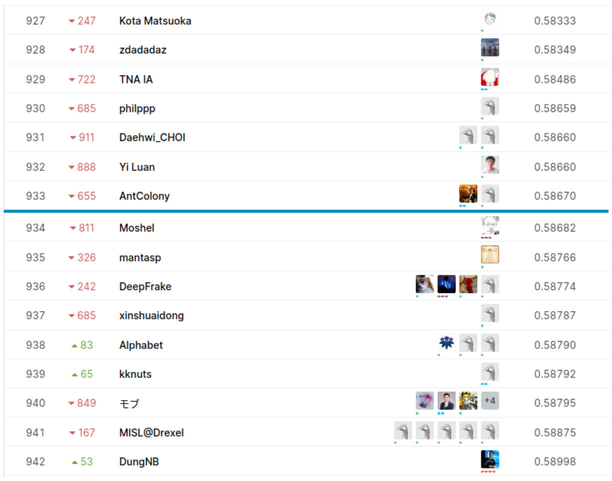
5.1 Competition Results

The winner of the competition detected deepfakes just over 82 % of the time. But when that algorithm was tested against a set of previously unseen deepfakes, its performance dropped to a little over



272	KCCS KYOTO		+5	0.42772
273	hello_tmy			0.42791
274	Panjtan			0.42798
275	effata			0.42805
276	vecxoz			0.42809
277	Ping pong			0.42835
278	AntColony			0.42855
279	psysdb			0.42876
280	BRAVO			0.42883
281	Justin Güse			0.42909
282	kagren			0.42971
283	Grzegorz Sionkowski			0.43021
284	Gaurav Dhama	Fork of Fork of Fork...		0.43144
285	Dulya Ivan			0.43169
286	dan musetoiu			0.43228
287	The Dream Team		+3	0.43236

Figure 9: Public Leaderboard.



927	▾247	Kota Matsuoka		0.58333
928	▾174	zdadadaz		0.58349
929	▾722	TNA IA		0.58486
930	▾685	philppp		0.58659
931	▾911	Daehwi_CHOI		0.58660
932	▾888	Yi Luan		0.58660
933	▾655	AntColony		0.58670
934	▾811	Moshel		0.58682
935	▾326	mantasp		0.58766
936	▾242	DeepFrake		0.58774
937	▾685	xinshuaidong		0.58787
938	▴83	Alphabet		0.58790
939	▴65	kknuts		0.58792
940	▾849	モブ	+4	0.58795
941	▾167	MISL@Drexel		0.58875
942	▴53	DungNB		0.58998

Figure 10: Private Leaderboard. The reshuffling of the leaderboard is impressive.

65 percent[2]. The key feature of his solution is the domain specific data augmentation meant to force the models to learn specific characteristics in frames by cutting out portions of the faces. Learning rate decay, parallel training on muliple GPUs, larger margins around the faces and longer training times are the main features that our pipeline did not include which, according to the best performers, significantly improved the generalization of their solutions.

6 Conclusion

At the end of the competition, the most successful solutions were published and we learned so many invaluable lessons about the mistakes that we made during this challenge that we have

[2]https://www.wired.com/story/deepfakes-not-very-good-nor-tools-detect/

already successfully applied in other projects in the following months, both by studying these notebooks and by following the discussions around the problem during the three months that it lasted. In the end many of the choices that we many in our pipeline turned out to be correct, but what separated ours from better solutions was the attention to crucial details, the use of some advanced techniques like multitask learning and, most importantly, a better understanding of the problem. Something that really slowed us down was the fact that, due to our lack of experience with the AWS platform, we quickly run out of the free credits and we were limited to train our models only on Google Colab and Kaggle.

One experiment that made us loose a lot of days was trying to build a better validation set by clustering the actors in the dataset in order to avoid any leakage between the sets, but it turned out that the Public Test score was even less correlated to the validation score and we moved back to the simple split based on the provided parts. This challenge turned out to be well beyond our skills mainly because of our inexperience with such big datasets, pipeplines with so many moving parts, conducting a rigorous program of experiments over a few months of work and keeping detailed track of all our tests that led us to follow rabbit holes that led nowhere for weeks by making us waste a lot of precious time. One of our most grave mistakes that we never thought about was that our margin in the extracted faces was set to default value of 0, whilst we learned afterwards from the top solutions that this kind of models with this dataset of images generated with this deepfake algorithms learn to detect some inconsistency between manipulated region and surrounding region. With a few adjustments in different parts of the pipeline we might have reached a way better score, that in hindsight look very silly.

References

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. 2020. Albumentations: Fast and flexible image augmentations. *Information*, 11(2).

Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Ferrer. 2019. The deepfake detection challenge (dfdc) preview dataset.

Tim Esler. 2019. Face recognition using pytorch.

Kaggle. 2019. The deepfake detection challenge.

Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Niessner. 2019. Faceforensics++: Learning to detect manipulated facial images. In *The IEEE International Conference on Computer Vision (ICCV)*.

Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA. PMLR.

Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. 2020. Deepfakes and beyond: A survey of face manipulation and fake detection. *arXiv preprint arXiv:2001.00179*.

K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. 2016. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

A Source code

```python
def extract_face_from_frame(frame, fd):
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    pilimg = Image.fromarray(frame)
    # face detection and extraction
    faces, confs = fd.detect(pilimg)
    if faces is None:
        return None
    best = confs.argmax()
    box = [max(0, int(x)) for x in faces[best].tolist()]
    img = frame[box[1]:box[3], box[0]:box[2]]

    #resize and 0 border
    sf = 224/np.max(img.shape)
    img_rs = cv2.resize(img, (int(img.shape[1]*sf), int(img.shape[0]*sf)), fx=sf, fy
=sf)
    bottom = int((224-int(img.shape[0]*sf))/2)
    top = 224 - img_rs.shape[0] - bottom
    left = int((224-int(img.shape[1]*sf))/2)
    right = 224 - img_rs.shape[1] - left
    img = cv2.copyMakeBorder(img_rs, top, bottom, left, right, 0)
    return img

def extract_faces_from_video(path, fd, t=None, n=None, transforms=None):
    output = []
    cap = cv2.VideoCapture(path)
    ret = True
    begin = time.time()
    count = 0
    while ret:
        if n is not None and count >= n:
            break
        if (not t is None) and (time.time() - begin > t):
            break
        ret = cap.grab()
        if not ret:
            break

        # next frame extraction
        ret, frame = cap.retrieve()

        img = extract_face_from_frame(frame, fd, transforms=None)
        if transforms is None:
            img = cv2.cvtColor(img_rs, cv2.COLOR_RGB2BGR)
        else:
            img = transforms(image=img_rs)['image']
        if img is None:
            continue

        output.append(img)
        count += 1
    cap.release()
    return np.asarray(output)
```

Listing 1: Face detection and extraction

```python
train_transforms = albumentations.Compose([
    ShiftScaleRotate(p=0.2, scale_limit=0.25,
                     border_mode=1, rotate_limit=25),
    HorizontalFlip(p=0.1),
    Cutout(p=.1),
    RandomContrast(p=.1),
    RandomBrightness(p=.1, limit=0.3),
    JpegCompression(p=.2, quality_lower=15, quality_upper=60),
    Downscale(scale_min=0.25, scale_max=0.25, p=0.2),
    GaussNoise(p=0.1),
    Normalize()
], additional_targets={'image2': 'image'})
```

Listing 2: Image Augmentation

```python
def train(model, name, train_set, valid_set, early_stopping,
          train_transforms, valid_transforms, root_dir,
          batch_size, epochs):
    model_name = 'checkpoint_' + name + '.h5'
    patience = 0
    best_val_loss = None
    history = list()
    for epoch_n in range(0, epochs):
        print('EPOCH:', epoch_n, '/', epochs)
        train_set = train_set.sample(frac=1).reset_index(drop=True)

        nbatches = 0
        train_loss = 0
        train_acc = 0
        for i, (X_train, Y_train) in tqdm(enumerate(generate(train_set, batch_size,
                                                             train_transforms,
    root_dir,
                                                             train=True), 1)):
            print(X_train.shape, Y_train.shape)
            batch_loss, batch_acc = model.train_on_batch(X_train, Y_train,
                                                         reset_metrics=True)
            train_loss += batch_loss
            train_acc += batch_acc
            nbatches += 1

            if i % 1000 == 0:
                train_loss /= nbatches
                train_acc /= nbatches
                nbatches = 0
                # evaluate
                valid_loss, val_acc, val_auc, val_cm, clipped_loss, _ = evaluate([
    model], valid_set, 32,

    valid_transforms, root_dir,

    [(0.1, 0.9), (0.15, 0.85)])
                if (best_val_loss is None) or (valid_loss < best_val_loss):
                    patience = 0
                    valid_loss = valid_loss
                    model.save_weights(model_name)
                else:
                    patience += 1

                print('\nTRAIN LOSS:', train_loss, 'VALID LOSS:', valid_loss,
                      'VALID LOSS CLIPPED:', clipped_loss)
                history.append({'epoch': epoch_n, 'chunk': i, 'train_loss':
    train_loss,
                                'valid_loss': valid_loss, "val_loss_clip":
    clipped_loss,
                                'valid_accuracy': val_acc, 'valid_auc': val_auc,
                                'valid_cm': val_cm})

                # early stopping
                if patience >= early_stopping:
                    model.load_weights(model_name)
                    return model, history
    model.load_weights(model_name)
    return model, history
```

Listing 3: Training

```python
def evaluate(models, valid_set, batch_size, transforms, root_dir, clips, verbose=0):
    loss = keras.losses.BinaryCrossentropy()

    y_pred = np.array([])
    target = np.array([])
    for X_valid, Y_valid in tqdm(generate(valid_set, batch_size,
                                          transforms, root_dir, train=False)):
        target = np.append(target, Y_valid, axis=0)
        outputs = np.zeros(batch_size)
        for model in models:
            outputs += (model.predict(np.array(X_valid),
                                      batch_size=batch_size).reshape(-1) / len(
    models))

        y_pred = np.append(y_pred, outputs, axis=0)

    valid_loss = loss(target, y_pred).numpy()

    clipped_losses = np.array([])
    for low, high in clips:
        clip_loss = loss(target, np.clip(y_pred, low, high)).numpy()
        clipped_losses = np.append(clipped_losses, [clip_loss], axis=0)

    acc = accuracy_score(target, np.round(y_pred))
    auc = roc_auc_score(target, y_pred)
    cm = confusion_matrix(target, np.round(y_pred))
    curve = roc_curve(target, y_pred)
    if verbose:
        print(len(y_pred) - np.round(y_pred).sum())
        print(classification_report(target, np.round(y_pred)))
    return valid_loss, acc, auc, cm, clipped_losses, curve
```

Listing 4: Evaluation

```
def predict(models, df, root_dir, d_type=’video’, extract=’No’):
    device = torch.device(’cuda:0’ if torch.cuda.is_available() else ’cpu’)
    mtcnn = MTCNN(keep_all=False, select_largest=False, device=device, min_face_size
     = 60)
    test_transforms = get_test_transfoms()
    preds = []
    b = time.time()
    for index, row in tqdm(df.iterrows()):
        if d_type == ’video’:
            imgs = extract_faces_from_video(root_dir + row.filename, mtcnn, t=1,
                                            transforms=test_transforms)
        else:
            img = cv2.imread(root_dir + row.face)
            if extract == ’Yes’:
                img = extract_face_from_frame(img, mtcnn)
            else:
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            if len(img) == 0:
                preds.append(0.51)
                continue
            imgs = test_transforms(image=img)[’image’].reshape((-1, 224, 224, 3))
        if len(imgs) == 0:
            preds.append(0.51)
            continue
        output = 0
        for model in models:
            output += model.predict(imgs).mean() / len(models)
        preds.append(np.clip(output, 0.01, 0.99))
        print(index, len(imgs), output)
    print(time.time() - b)
    return preds
```

Listing 5: Inference