DeepFakes Detection Lab - Report

Luis Antonio Ortega Andrés
Antonio Coín Castro

March 23, 2021

Task 1: intra-database analysis

The goal of this task is to develop and evaluate DeepFake detection systems over the same database. In this task, you should use only the UADFV database, which is divided into `development` *and* `evaluation` *datasets.*

The notebook associated with this task is `Task1-2.ipynb`.

a) *Provide all details (including links or references if needed) of your proposed DeepFake detection system.*

Our DeepFake detection system consists on the following modules:

Preprocessing

First of all, the given images from UADFV have been preprocessed following these steps:

1. The data is loaded using `Keras`' `flow_from_directory` function (Chollet *et al.*, 2015), so that we don't have to keep all images in memory while training our model. We resize each image to a resolution of 768×768.
2. We use the MTCNN face detector (Zhang *et al.*, 2016) in order to crop the section of the image that contains the face. The aim of this phase is to erase those parts of the image that are not relevant for our classification task.
3. We add random noise to the images as a form of regularization, to force our classifier to learn more relevant features. More precisely, we randomly decide wether the image is slightly blurred using a Gaussian filter from OpenCV (Bradski, 2000), or simply perturbed by Gaussian noise. Each of these outcomes is equally likely and every image goes through one or the other, but never both. Note that these perturbations are only applied to training samples and not to the evaluation images.

Feature extraction

We decided that the best way of extracting features from the faces was `dlib`'s 68-point face landmark detector (King, 2009). For each (cropped) face image, we obtain the spatial position of the 68 landmarks, and our final feature vector is composed of the gray-scale intensities of these landmarks. We then center each feature vector to have zero mean.

We also considered the possibility of applying PCA for dimensionality reduction, but since we wanted to try both approaches, we delegated this decision to the actual training step.

Classifier

The final step of our model is a machine learning model that acts as a classifier to which the feature vectors are fed for training. At first we considered four distinct families of models for this job: **Linear SVM**, **SVM with RBF kernel**, **2-layer perceptron** and **Logistic Regression** models. The process by which a final classifier was selected can be seen on the next section.

b) *Provide all details of the development/training procedure followed and the results achieved using the development dataset of the UADFV database. Show the results achieved in terms of Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC).*

For the training procedure we performed a hyperparameter grid search using `Sklearn`'s API (Buitinck *et al.*, 2013), in which each family of classifiers mentioned above is tested with a wide range of possible hyperparameters to maximize the AUC metric. For example, the number of hidden layers considered for the MLP are $\{(50), (100), (50, 50), (100, 100)\}$, and the linear SVM regularization term C varies from 10^{-3} to 10^{3} with 40 equidistant steps. To make the process fair, we employ **5-fold cross validation** to select the best hyperparameters for each model, and in the end we select the model that achieved the highest cross-validated AUC. Note that each family of classifiers is trained using the same folds, so the results are comparable.

Before completing the training pipeline, we added the possibility to train each classifier **with and without PCA reduction**, to see whether it would have a positive impact on the final result. The fraction of explained variance to be retained was also an hyperparameter, whose search space was $\{0.90, 0.95, 0.99\}$. An important detail is that the data is first standardized to have zero mean and unit variance, since we observed that the performance of the classifiers was noticeably better after this normalization.

The best models of each family after running the grid search are shown in Table 1, along with their cross-validated AUC scores.

Model	**PCA**	**Parameters**	**AUC (CV)**
SVM RBF	No	$C = 1.6681$, $\gamma = 0.0167$	0.9937
SVM Linear	No	$C = 0.0174$	0.9186
MLP	No	$\alpha = 0.2154$, $size = (100, 100)$	0.9883
LR	No	$C = 0.1624$	0.9180

Table 1: Results of the hyperparameter tuning phase.

As we can see, the best performing model was the **non-linear SVM with RBF kernel using the raw features**, and this is the model that we choose as our final classifier (with the optimized parameters found). An interesting thing to note is that PCA was discarded on every model, probably because the number of raw features considered is not very high and the model wouldn't benefit from having less features. The results of evaluating our chosen model on the whole development set can be seen on Figure 1, in which we see that we achieve a perfect score.

c) *Describe the final evaluation of your proposed DeepFake detection system and the results achieved using the evaluation dataset (not used for training). Show the results achieved in terms of ROC curve and AUC.*

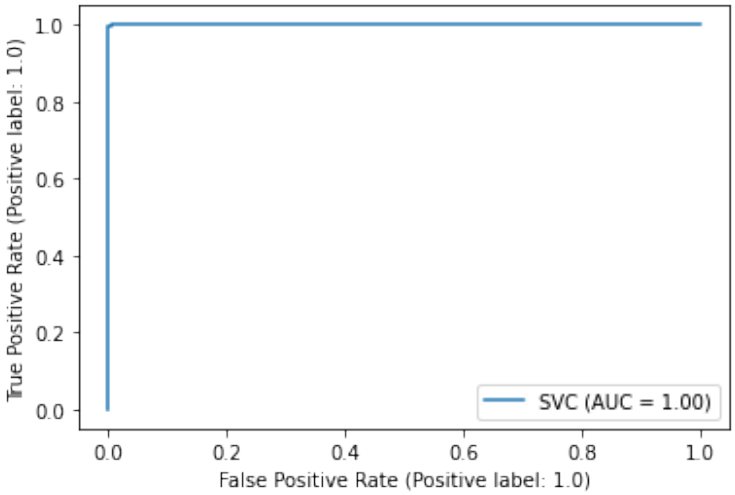

Figure 1: ROC curve and AUC value on UADFV's development set.

Provide an explanation of your results.

Having chosen our model, we simply evaluate it on the previously unseen evaluation set of the UADFV database, and the results can be seen in Figure 2. We obtain an AUC of 0.95, quite close to the perfect score of 1.0. Since there was no sudden drop in the performance of the model, it is safe to assume that the distribution of the evaluation set is more or less the same as that of the development set (as it should be). We can say that we have succeeded in this task, since our model has learned to detect DeepFakes on the UADFV database while avoiding overfitting.

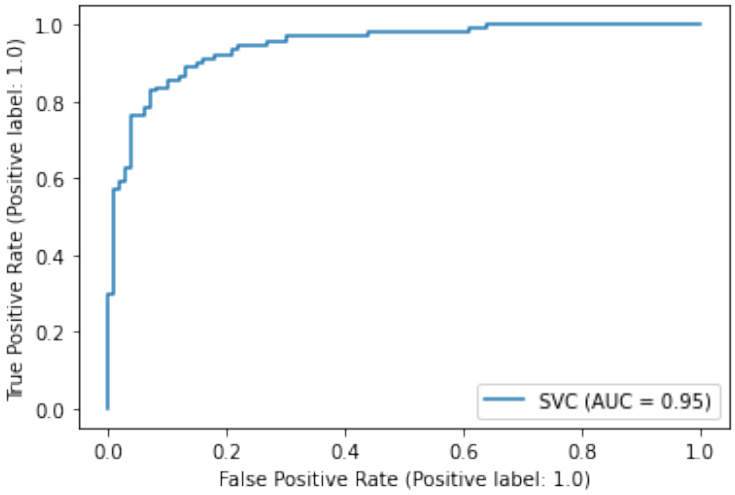

Figure 2: ROC curve and AUC value on UADFV's evaluation set.

Other approaches and failed attempts

1. We considered extracting features using an image destriptor such as SURF or SIFT, and then following the Bag of Visual Words technique. However, the results obtained were worse, so we discarded this approach.

2. Along with the landmarks intensity, we made some attempts at using the the landmarks locations as features, but this resulted in overfitting. As a regularization technique, we centered those locations (substracting the mean) in order to make them location invariant. The performance improved a bit but in the end we decided to stick to the face landmarks only.
3. At one point, MTCNN did not correctly detect the faces for every image, and during training we decided to simply skip these images. However, since we cannot skip test cases, we decided that an image with no face detected would be assumed to have the same features as the last image with the same label. In the end we solved the underlying problem (by resizing the input images), but we left all the code checks just in case they were needed in the future.
4. More models were tested in the grid search, such as Random Forests and other boosting techniques. However, they performed so poorly compared to the rest that we decided not to include them in the final report.

Task 2: inter-database analysis

The goal of this task is to evaluate the DeepFake detection system developed in Task 1 with a new database (not seen during the development/training of the detector). In this task, you should use only the Celeb-DF. You only need to evaluate your fake detector developed in Task 1 over the `evaluation` *dataset of Celeb-DF, not train again with them.*

The notebook associated with this task is `Task1-2.ipynb`.

a) *Describe the results achieved by your DeepFake detection system developed in Task 1 using the evaluation dataset of the Celeb-DF database. Show the results achieved in terms of ROC curve and AUC. Provide an explanation of your results in comparison with the results of Task 1.*

We evaluate the model established in the previous task (SVM with RBF kernel and optimized hyperparameters) on the evaluation set of Celeb-DF. The results achieved can be seen in Figure 3.

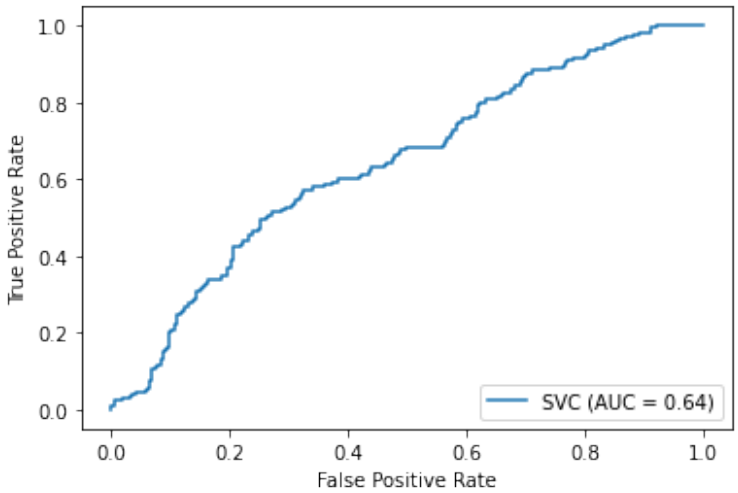

Figure 3: ROC curve and AUC value on Celeb-DF's evaluation set.

We obtain an AUC of 0.64, which is considerably lower than the one achieved in the previous task. This behaviour makes sense, considering that the training dataset was less challenging than this

one, in the sense that the deepfakes were more easily spotted by a human. In the UADFV dataset there were usually vertical and horizontal strips surrounding the place where the face exchange had taken place, and in this new dataset deepfakes have a higher quality and do not show such markings. In short, our detection model scales poorly to this dataset because the data in which it was trained **is not representative enough** of this new evaluation set.

Task 3: inter-database proposal

The goal of this task is to improve the DeepFake detection system originally developed in Task 1 in order to achieve better inter-database results. You must consider the same evaluation dataset as in Task 2 (i.e. the `evaluation` *dataset of the Celeb-DF database).*

a) *Describe the improvements carried out in your proposed DeepFake detection system in comparison with Task 1.*

In the first place, we tried to increase the quantity of data we had to train our model. To build a new training set, we used the full given UADFV dataset (merging development and evaluation) and several samples of the DeepFakeTIMIT and VidTIMIT databases (Sanderson & Lovell, 2009). As DeepFakeTIMIT is composed of videos, we developed a python script in order to take frames from the videos (`TIMIT_data_preprocess.py`). Moreover, since these datasets have a relatively low diversity (only 32 different people appear on the videos), we decided not to take all the samples we could, but instead get a random sample of both real and fakes images while we cropped a 160×160 face image using MTCNN (`Face_detector_splitter.py`).

Having aumented our data, we followed different approaches in order to find and optimize the most suitable technique for this problem. In all cases the **training dataset** was composed of 1829 160×160 face samples, and the **evaluation dataset** was composed of the 160×160 face crops of each sample (out of 600) in the given `Celeb-DF` dataset. Although the MTCNN face detector remains an essential part of our model architecture, we saved the cropped faces to disk and worked only with them to increase the computational efficiency.

Train the same model with more data

First of all, we wanted to check how the best model found of the previous task behaved with this new training dataset, and set a *baseline* to be compared to the subsequent attempts.

The procedure was simple and can be seen in the notebook `Task3_ML.ipynb`. We reused the majority of the code developed for the previous task and created a pipeline with the optimal classifier. The AUC obtained using this approach was 0.67, which is only a bit better that what we had before.

Consider alternative features

In this approach we intended to use `FaceNet` (Schroff *et al.*, 2015) to obtain a more representative set of features for each face. To this end, we took a pre-trained model of FaceNet[1] that computed a face embedding to 128 features, and used these features **along with the landmarks intensities** to train and cross-validate the same families of models considered in Task 1. A small change that

[1] https://github.com/nyoki-mtl/keras-facenet

we make is that we stop randomly blurring the training images, since we observed that it could heavily distort the embedding and worsen the performance of the final model.

The resulting best model was again the SVM with RBF kernel, using a regularization term $C = 4.6415$ and a kernel parameter $\gamma = 0.0167$. The AUC score on the evaluation dataset was 0.7, which is better than our baseline model.

The code for this strategy can be consulted in `Task3_Facenet.ipynb`.

Train a deep learning model from scratch

In this attempt we wanted to take advantage of the increased number of samples in the training dataset to use a *deep-but-not-so-deep* convolutional neural network for the classification task. We considered a simple model consisting on two blocks of convolution-activation-dropout-pooling and two fully connected layers at the end, the last one being a sigmoid with only one unit. We also used data augmentation techniques such as random rotations or horizontal flipping as an extra regularization layer.

However, after tuning the parameters for a while using a validation set and the early stopping technique, the model either overfitted to the training dataset or did not learn at all, so we discarded this approach altogether. The general structure with our experiments can be found in `Task3_DL.ipynb`.

Fine-tune a pre-trained deep learning model

As out last attempt, we aimed to fine-tune a pre-trained deep neural network. We thought that we could benefit from a large network that was already trained on a big database, and perform transfer learning to try to solve our detection problem. We used the `Keras` API, and the steps followed were[2]:

1. We loaded a CNN pre-trained on the ImageNet database (Deng *et al.*, 2009). We tested several well-known models such as VGG19, ResNet50, InceptionV3, XceptionNet...
2. We dropped the top layers that act as a classifier, and put a small fully connected model of our choosing instead.
3. We train **only the top layers** on our dataset for a few epochs, so that the weights are correctly initialized and are not random. This is done to avoid disturbing the previously tuned weights of the base model.
4. **We unfreeze the last convolutional block of the base model** and train with a small learning rate to perform the actual fine-tuning of the network. We don't unfreeze the whole network because the initial layers act as a coarse feature extractor, and our hypothesis was that the images on ImageNet share the same lower level features as our images. It is only on the last layers of the network that features close to our deepfake detection problem are learned, and that is where we put our efforts. Moreover, if we trained the whole network we would almost certainly incur in overfitting, since we have a very limited quantity of training data.

We retain the random data augmentation technique from previous attempts, but we do not consider a validation set in this case, to try and make the most out of our few training examples. After some tries, the best configuration found was as follows:

[2]We followed the recommendations in https://keras.io/guides/transfer_learning/

- The chosen base model was the VGG19 network (Simonyan & Zisserman, 2015). We needed to add a preliminary preprocessing step to accomodate the images to VGG19's expectations (using the function `vgg19.preprocess_input`).
- The top model chosen was a small model consisting on two fully connected layers with ReLu activation and 512 and 256 units, respectively, with a Dropout layer of 0.25 after the first and one of 0.5 after the second. The Dropout layers are regularizing layers that "turn off" a fraction of the neurons during training, to force the model to learn more relevant features. At the end we put a Dense layer with one unit and sigmoid activation to obtain the probabilities of fake and real.
- We trained the top model for 3 epochs, and then unfroze the last convolutional block of VGG19 and trained for 15 more epochs, in this case using the SGD optimizer with a learning rate of $5 \cdot 10^{-4}$.

The code developed can be consulted in the notebook `Task3_Finetune.ipynb`.

b) *Describe the results achieved by your enhanced DeepFake detection system over the final evaluation dataset. Show the results achieved in terms of ROC curve and AUC. Provide an explanation of your results in comparison with the results of Task 2.*

After evaluating the trained model on the evaluation dataset, we obtained a shocking AUC score of 0.8107 , whose corresponding ROC curve can be seen in Figure 4. It turns out that this fine-tuning approach was the best way of enhancing our model, and we improved the results of Task 2 by almost 20 percentage points.

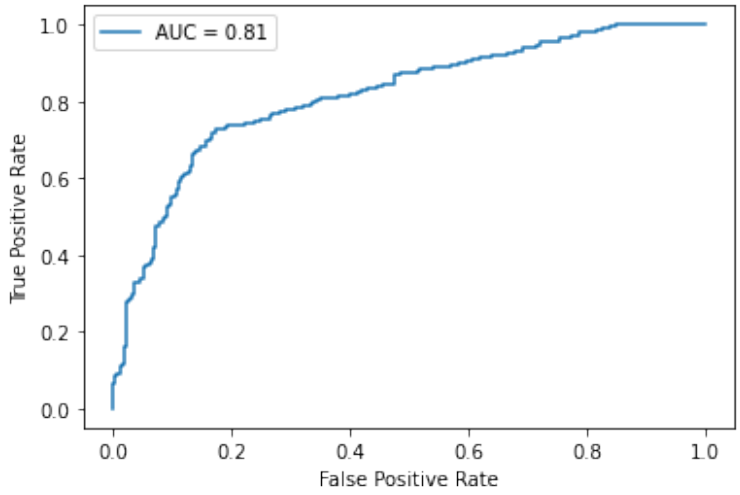

Figure 4: ROC curve and AUC value on Celeb-DF's evaluation set for our enhanced model.

c) *Indicate the conclusions and possible future improvements.*

The conclusions we draw from this assignment are summarized below:

- DeepFake detection is a challenging task, especially when there are more and more high-quality fakes that even humans have a hard time detecting. The fact that we train on a specific database and obtain good results is not a guarantee that the model will generalize to different databases.

- Feature extraction is a key part of the process, and the more relevant and accurate features we obtain, the better the performance of our final model.
- While we can obtain a decent performance using machine learning techniques, deep learning outperforms classical approaches when used appropriately.
- Deep learning is an excellent way to perform a sort of automatic feature extraction, letting the network learn which features are relevant for the specific task of deepfake detection. However, for this approach to succeed we generally need a large amount of training data.
- We can overcome the need for many training images using transfer learning and fine-tuning a pre-trained network, but we have to pay close attention to the process and employ a suite of regularization techniques to avoid overfitting.

As a final comment, we considered some alternative approaches to feature extraction, but we didn't have time to implement them. For example, we thought it would be a good idea to obtain specific masks for different parts of the face, such as eyes, teeth, nose, etc, and in general to adopt a more fine-grained approach when it comes to feature extraction. Moreover, it would be good to have an in-depth look at how deepfakes are usually generated (using GANs), and then try to exploit any weaknesses in the process to improve our detection performance.

References

Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Buitinck, Lars, Louppe, Gilles, Blondel, Mathieu, Pedregosa, Fabian, Mueller, Andreas, Grisel, Olivier, Niculae, Vlad, Prettenhofer, Peter, Gramfort, Alexandre, Grobler, Jaques, Layton, Robert, VanderPlas, Jake, Joly, Arnaud, Holt, Brian, & Varoquaux, Gaël. 2013. API design for machine learning software: experiences from the scikit-learn project. *Pages 108–122 of: ECML PKDD Workshop: Languages for Data Mining and Machine Learning*.

Chollet, François, *et al.* 2015. *Keras*.

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

King, Davis E. 2009. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, **10**, 1755–1758.

Sanderson, Conrad, & Lovell, Brian C. 2009. Multi-region probabilistic histograms for robust and scalable identity inference. *Pages 199–208 of: International conference on biometrics*. Springer.

Schroff, Florian, Kalenichenko, Dmitry, & Philbin, James. 2015. FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun.

Simonyan, Karen, & Zisserman, Andrew. 2015. *Very Deep Convolutional Networks for Large-Scale Image Recognition*.

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