Deep Learning for DeepFake Detection

Elena Govi
229285@studenti.unimore.it

Tobia Poppi
257211@studenti.unimore.it

Fabio Marinelli
259857@studenti.unimore.it

Abstract

Deep Learning has been successfully applied in order to solve many tasks. The one we would like to work on is about DeepFake images.
DeepFake algorithms can generate fake images and videos that humans cannot distinguish from authentic ones. This is a powerful tool, but it might lead to dangerous consequences if used with bad intentions. The proposal of technologies that can automatically detect and assess the integrity of digital visual media is therefore indispensable [8]. This report illustrates the development of a project aimed at the detection of deepfake images. Our code is available at https://github.com/tobiapoppi/CVCS_project_DeepFake

1. Introduction

Recently the world is witnessing a significant increase in multimedia content generation due to the availability of economical digital smart devices like cellphones, laptops, tablets and digital cameras. At the same time, the tremendous advancement in artificial intelligence allows anyone to easily manipulate multimedia content and spread disinformation online. In fact, sophisticated algorithms are available to everyone. Disinformation has potentially tremendous consequences, according to [6], such as election manipulation, creation of warmongering situations, defaming any person. For these reasons, deepfake detection has to be considered as an important and actual problem.
In particular, this report aims to detect images manipulated by several methods, such as FaceSwap, Deepfakes, Face2Face and NeutralTextures.

Figure 1. Examples of FaceSwapp from Deepfake Challenge Dataset and FF++ Dataset.

Focusing on FaceSwap images, they consists in the realistic combination of two images, the source and another candidate one. Recently, realistic results are being provided by some DL-based approaches, which have become popular for synthetic media creation. For these reasons, fake detection will play a fundamental role in the future.
The main contributions of our work are the following:

- Introducing a new dataset, as a combination of already existent two;
- Introducing a new pipeline for a given image to be detected;
- Providing two different variations of Xception architecture, a powerful Convolutional Neural Network (CNN), created for image classification.

Our pipeline is based on the following points:

1. Choose an image to check;
2. Apply the image retrieval algorithm in order to find if there are some similar, fake or real, images in the search pool(Section 4);
3. Crop the face of each subject in the image with EfficientDet Head Detector (Section 3)
4. Apply to them the trained DeepFake classifier in order to find which are the real one (Section 5).

Our pipeline could be the starting point for the implementation of a bigger project, able to find on the web similar images and identify the manipulated against the real ones.

2. Dataset

In our project pipeline we used 5 datasets, of which 3 were already used in literature, and 2 new datasets created by us from those.

- **Deepfake Challenge dataset. [2]** This dataset was made available by the Image Processing Lab of the University of Catania, in order to promote a challenge. The dataset contains 15000 close-up images of faces, of which 10000 real and 5000 fake, so it's unbalanced.

Fake images have been generated by 5 different algorithms: AttGAN [17], GDWCT [14], StarGAN [15], STYLEGAN [12] and STYLEGAN2 [13].

- **FaceForensics++ dataset**. [9] This is a famous dataset for DeepFake detection. It consists of 1000 original video sequences that have been manipulated with four automated face manipulation methods: Deepfakes, Face2Face, FaceSwap and NeutralTextures.

- **HollywoodHeads dataset. [10]** This dataset contains 369846 human heads annotated in 224740 video frames from 21 Hollywood movies 2. We used this dataset in order to achive the *head detection* task. This dataset utilizes the Pascal VOC format.

- **CVCS DFD (our dataset).** This is the first dataset we created by combining the Deepfake challenge dataset and frames extracted from *FF++* video sequences.

  It has been automatically created with our python script which performs two operations: it first extracts frames by the *FF++*'s sequences, then it puts together all the images contained in the individual *Deepfake Challenge dataset* and all the frames extracted by *FF++*, in order to obtain a unique bigger dataset, which better generalizes the problem.

  Then all the images are shuffled and split in three different folders: train, validation and test. The ratio of each split w.r.t the whole set can be chosen by the user.

- **CVCS Cropped DFD (our dataset).** This is an alternative version of *CVCS DFD* dataset, in which all the images are cropped in order to give to the classifier only faces, without body and background. It is a copy of the CVCS DFD dataset but all the images represents close-up heads. If an image does not contain a head, it's not included in CVCS Cropped DFD.

  To generate cropped images we used a custom-trained EfficientDet model to make inference on "CVCS DFD" dataset. This will be explained in details in section 3.1.

Figure 2. Two example images of HollywoodHeads dataset.

2.1. Annotations

Furthermore, we provided annotations. In the two original datasets the only way you can trace labels is by checking which folder they come from.

We will use our dataset to train some deep and non-deep models, so we need explicit annotations. We provided two different kind of annotations, because they are both useful on different algorithms used.

- **Yolo-like annotation type:** the dataset contains for each image a *txt* file named the same. That file contains 0 if the image is real and 1 if the image is fake.

- **Data-list annotation type:** outside from train, validation and test folders, there are three *txt* files which contain, for each line, the path to an image of the split the file refers to, and the label (0 if the image is real and 1 if the image is fake).

3. Head Detection

The CVCS DFD contains mixed images from Deepfake Challenge dataset and FF++ dataset. This means that a lot of images represent a scene with the full body person.

Given that our Deepfake Detector works on face details, it needs to receive in input images that represent just the head of the subject. For this reason we tackled the problem of head detection.

In computer vision the problem of face detection is so well known and there are already a lot of techniques that can reach good results. Anyway we considered important calling this task with the name of *head detection*. For our deepfake detector it is really important to see the whole the head of the person. There are some deepfake examples in

which hair are essential in order to detect wheter the image is fake or not.

For this reasons, using a simple method as *dlib* wouldn't be proper, because the result would be the detection of just a part of the face, moreover it wouldn't be so precise in finding really the right bounding box of the head. Thus, we decided to train an EfficientDet model.

3.1. EfficientDet

EfficientDet [7] is a state-of-the-art object detection architecture which gives top priority to efficiency and scalability.

The architecture propose the BiFPN module (weighted bi-directional feature pyramid network), which allows easy and fast multiscale feature fusion.

We trained EfficientDet on the HollywoodHeads dataset, which utilizes PascalVOC format. We didn't appply any architectural change to the network. The only change we made was in the dataset generator, due to dataset compatibility purpose.

3.2. Training

We approached the network training process by dividing it into first training phase and fine-tuning phase.

- In the **first training phase**, we started with a snapshot of *Imagenet* pre-trained weights, freezing the backbone parameters. We chose $\phi = 0$ as the scale hyperparameter, which corresponds to the simplest and lightest version of the network, and a batch size of 32 with 1000 iterations. This training elapsed 50 epoches.
- In the **fine-tuning** phase we started the training using the model parameters of the last epoch of first training. Batch size is reduced to 4 and iterations are increased up to 10000, and this will result on the one hand in increased learning details, on the other hand in a more noisy learning.

  In this particular case the network doesn't freeze the backbone, but it freezes the *batch normalization* layers, which is a best-practice in a network fine-tuning, especially if it's done on examples different from the first training. Keeping the batch-norm parameters of the first training, which uses an higher batch size, helps the model to better generalize. The training lasted 50 epochs.

After both phases, we selected the best weights of the fine-tuning based on the training-validation loss tradeoff graph.

3.3. Results

After the first training phase we obtained a *mAP* value of 0.85, and with the fine-tuning we got a *mAP* value of 0.88.

Figure 3. Two examples of head detection on CVCS DFD dataset.

4. Image Retrieval

Given a set of images and a query image, the purpose of this phase is to find the images most similar to the query. This tool could be useful in our pipeline in finding some similar images modified in different ways.

4.1. Methods

The usual pipeline for this task consists of:

- Feature extraction method
- Distance/similarity computation between the query and all other images
- Sorting of closest images

Image features are local, meaningful, detectable part of the image. Two different methods are tried for the feature extraction, only one of them uses weights learned from a CNN.

The first is Harris Detection algorithm [3], able to detect edges of an image. We chose it because edges are important features of an image. However, as we will see, they are not enough. As pre-processing, each image is resized to 299×299 and then converted to its grayscale version. Then the feature extraction is applied.

The second method utilizes pretrained weights from resnet [4]. It is a Convolution Neural Network made for image classification problems. In details, the forward phase of the resnet network, pre-trained on imagenet, is computed, with the exception of the final layer, which would predict the class.

For this work *Resnet16* is chosen, therefore the feature extraction output is a 512×1 array. Since *resnet* requires a 244×244 image size, also in this case a pre-processing phase is necessary. In addition to resize, pixel values are normalized with specific mean and standard deviation values.

For similarity/distance computation, both cosine similarity and euclidean distance are chosen.

Representing (x_1, x_2) as scalar product and x_i as feature vector, cosine similarity is:

$$CosSim(x_1, x_2) = \frac{(x_1, x_2)}{\|x_1\|\|x_2\|}$$

and Euclidean Distance is :

$$EucDist(x_1, x_2) = \sum_{j=0}^{n} (x_{1j} - x_{2j})^2$$

In the first case the highest value means that the image is the closest to the query. The Euclidean distance works in the opposite way. Finally, the possible combinations of methods are four.

4.2. Results

A qualitative analysis, figure 4, is done on this task. Observing images, the combinations *Harris detection - cosine similarity* and *Resnet16 - cosine similarity* appear to perform better. For each method there are 5 common images and 5 different: this means that the most similar images are the same for each pipeline, nevertheless each case extracts different features.

5. Deepfake Detection

5.1. Methods

The methods proposed in this report should be divided in two categories:

- non-deep category
- different types of Convolutional Neural Networks based on the same architecture, called Xception

5.2. Non-Deep Detectors

The first non-deep methods are inspired by [16], which follows the standard detection pipeline and apply, first, a feature extraction algorithm and, second, a binary classifier. Despite the choice of [16], SURF, we decided to use the Scale-Invariant Features Transform ([5]) as feature extractor. More in details, SIFT is a keypoints localizer and descriptor. The first phase is based on keypoints' localization through the Difference of Gaussians method: different Gaussian filters are applied at different scales and then the subtraction results between them are saved. Final outputs should be approximations of Laplacian of Gaussian Convolutions.

Consequently, local maxima and minima are chosen comparing them to their 8 neighbours. In addition, not all local maxima and minima are saved, but they are chosen by a selection algorithm.

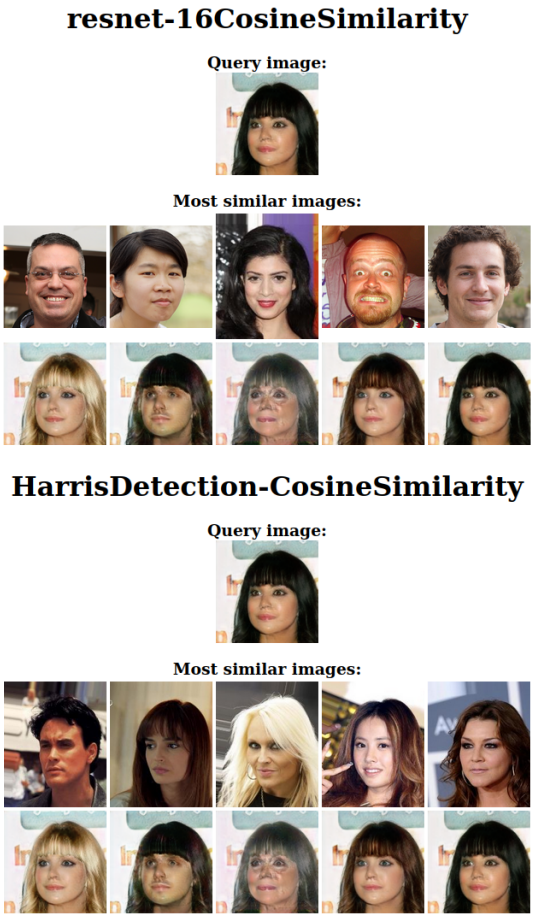


Figure 4. These are two examples of 10 most similar images to the query image, retrieved by the four different feature-similarity combinations.

In the second descriptive phase, keypoints are described by the gradient directions histogram.

Each image was then represented as a matrix in which each row is a keypoint and each column is a feature of it ($n_{samples} \times 256$ - dimensional array). On the obtained SIFT features, we adopt a bag of word (BoW) model to represent the image in a more compact and effective manner. First, the keypoints are quantized into visual words to form a codebook (codebook generation part). This phase is done by a k-means clustering with $N_{clusters} = 256$. Each visual word is then represented by the centroid of its cluster. The second phase, BoW feature construction, finds the occurrences in an image of each specific visual word in the codebook and represent them in a histogram. Each image is represented by a histogram which counts how many descriptors belong to each visual word,

$$BoW = [n_1, ..., n_C],$$

n_i is the count for cluster c_i. The procedure after SIFT is described in the image 5.

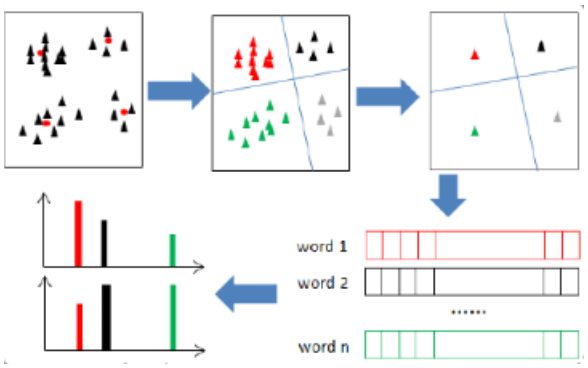


Figure 5. Bag of Word model after feature extraction.

Consequently, a Support Vector Machine Algorithm and a Random Forest Ensamble method were trained. They are machine learning, non-deep, methods. They receive, as input, the BoW histogram. Results were better than randomness, but not good: for the first the accuracy on the test results 0.6756, while the second obtained 0.6753. Before testing them, both classifier were optimized, respectively, on box constraint regularization parameter and on the number of classifiers, figure 6.

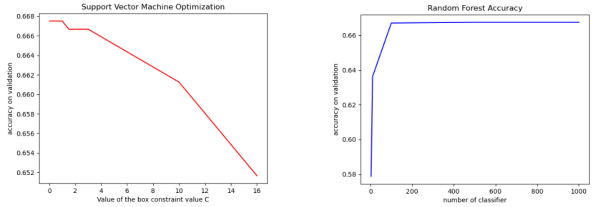


Figure 6. On the left SVM regularization parameter C is optimized on the validation set. On this case, SVM seams to prefer little values of C. Also different kernel function were tried and the best, used in this case, was the gaussian kernel function. On the right the number of classifier for Random Forest method are optimized: after 500 the classifier does not improve its result, therefore 500 is the best combination of high accuracy and low computational cost.

5.3. Deep Detectors

In deep learning architectures feature extraction and classification are done during the same process. In this case, we decided to choose Convolutional Neural Network to work on image classification. The network from which we started is called Xception [1].

Xception is a CNN which gave a new interpretation to Inception modules implemented by GoogLeNet in 2014 [11]. François Chollet invented the *depthwise separable convolution*, which is the main module used in Xception.

If we consider a simplified Inception module 7 that only uses a fixed size convolution and does not include average pooling, we could then replace all the parallel 1×1 convolutions with a single one but larger in terms of filters number 8. His output channels will be split in a number of sets equal to the number of parallel convolutions of the module.

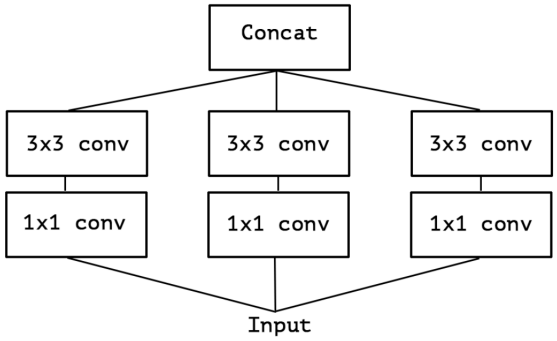


Figure 7.

This kind of operation is called *grouped convolution*. Xception implements an extreme version of grouped convolution, defining the grouping parameter equal to the number of input channels of the convolution. Thus, it is pretty reasonable to call this version of grouped convolution with the name of *depthwise convolution*.

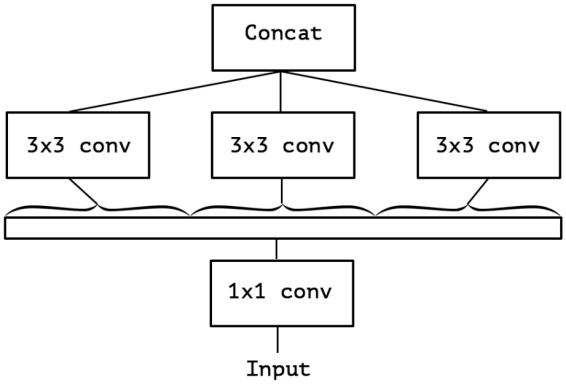


Figure 8. An equivalent reformulation of the simplified Inception module.

We proceed by outlining the three architecture variants we propose in this project.

Xception Architecture This is the original Xception architecture, taken from [1] and represented in Figure 9.

MidwayXception Architecture In this version of Xception we made some architectural changes. In first place we replaced the depthwise separable convolution module with a new one, which finds place in the midway between Xception depthwise separable module and Inception simplified module.

In [1], they make the following hypothesis: that the mapping of cross-channels correlations and spatial correlations in the feature maps of CNNs can be *enterely* decoupled. Our MidwayXception module uses as the grouping parameter half the number of input channels and as a result the new network has more parameters, and we lose the decoupling hypothesis.

That hypothesis can be interpreted as a too strong assumption. Furthermore, in [1] the author asserts that there is no reason to believe that depthwise separable convolutions are optimal, and he suggest those lines as a good future direction. For these reasons we decided to implement MidwayXception.

A second architectural change is the 1×1 parallel convolution added to the *Middle flow* 9 instead of adding the identity function.

LightXception Architecture It is our lighter version of Xception original architecture. Four of the eight blocks of the *Middle flow* have been removed. You can see these blocks in the original architecture represented in figure 9.

5.4. Results

N	Crop	Net	Opt	LR	BS	Ep.
1	No	Xcept	Adam	0.001	16	20
2	Y	MidXcept	Adam	0.001	16	20
3	No	MidXcept	Adam	0.001	16	20
4	Y	MidXcept	Moment	0.001	16	20
5	Y	MidXcept	Moment	0.01	16	20
6	Y	LightXcept	Adam	0.001	16	20
7	Y	Xcept	Adam	0.001	8	25
8	Y	Xcept	Adam	0.001	16	25
9	No	Xcept	Adam	0.001	16	15

Table 1. Architectures details. Opt is the optimizer type, which can be Adam or SGD Momentum. The Loss function, for all training, is the Weighted Cross-Entropy. Some trainings are done on the Cropped CVCS-DFD dataset, some others on the same data without the head detection. The Learning Rate is not fixed, as in the original Xception is an adaptive lr. Dropout is equal to 0.5.

N	ACC_v	ACC_t	$Prec_0$	$Prec_1$	Rec_0	Rec_1
1	0.85	0.85	0.82	0.87	0.72	0.92
2	0.76	0.843	0.77	0.87	0.73	0.89
3	0.835	**0.897**	0.96	0.87	0.70	0.98
4	0.755	0.824	0.73	0.86	0.72	0.87
5	0.764	0.776	0.62	0.89	0.82	0.75
6	0.803	0.849	0.74	0.91	0.82	0.86
7	0.763	0.778	0.89	0.696	0.60	0.93
8	**0.867**	**0.867**	**0.79**	**0.91**	**0.82**	**0.89**
9	0.893	0.896	0.97	0.88	0.71	0.99

Table 2. Final results: best accuracy on the validation set, accuracy on the test, Precision on the test for class 0 and 1, Recall on the test for class 0 and 1.

Results are resumed in table 1 and 2.

The best optimizer seams to be Adam, with learning rate equal to 0.001. In details, the learning rate is not fixed: it is adaptive, with a decay of 0.05 each 5 epochs. Unfortunately, we had not enough computational power for trying higher-dimensional batch-size. On the contrary of our expectations, architecture 3 with non-cropped data seams to be the best for accuracy on test. However, this is unexplainable, in fact the classification should be easier for detected images. At the same time, we noticed that in architecture 3 the recall for class 0 is the lowest, and the difference between recall 0 and precision 0 is the highest. This means that probably the network classify lot of images as 0, even if our training was balanced. For this reason we decided to consider, as best result, the 8-th architecture, with pre-cropped data. Our suppositions are confirmed by the final phase on our faces.

From figure 10, we observed that the architecture learns rapidly and, then, the high number of epochs does not help improving the final accuracy. Therefore, we tried on the test both parameters of the 8-th and 24-th epochs: the results were respectively of 0.8800 and 0.8668. Therefore, one future purpose should be the addition of an early stopping method. Moreover, the results on the 8-th epoch seems to be more balanced for classes: the precision on class 0 is 0.8428, while on class 1 is 0.8980, the recall on class 0 is 0.7811, while on class 1 is 0.9297.

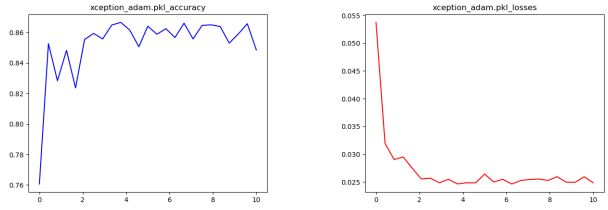


Figure 10. From these images, representing the accuracy and loss function values on the validation, is evident that improving the number of epochs does not achieve better results.

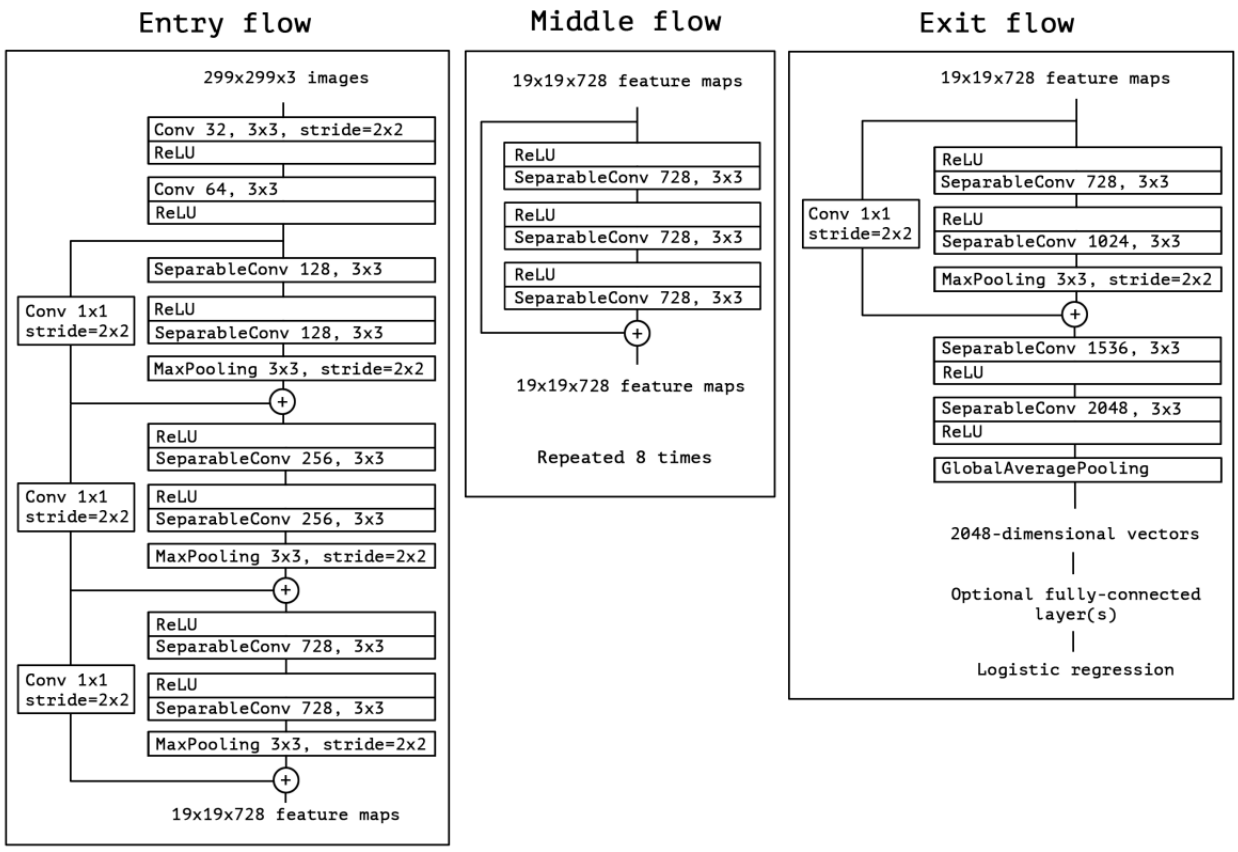
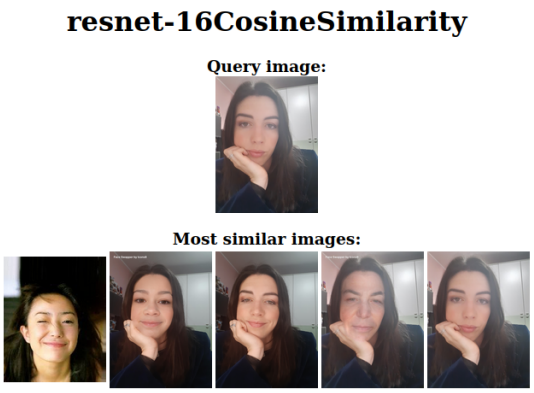


Figure 9. Original Xception Architecture details from the paper.

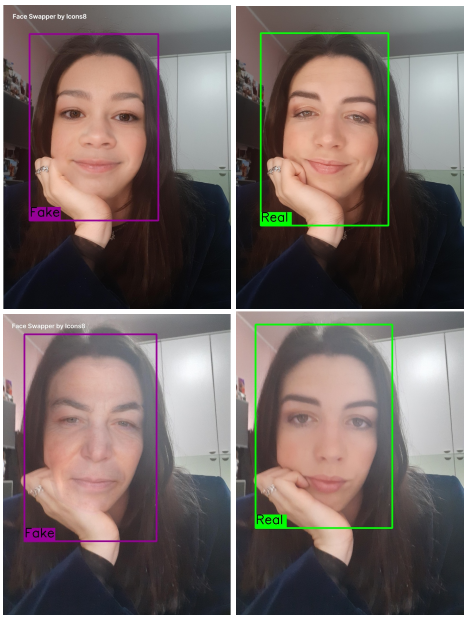
6. Final application: our faces

Finally, we decided to try out our pipeline on our faces. First, images fake and real are acquired. The modification is done on the website *https://icons8.it/swapper*, which works with AI algorithms. Here are the results:

1. Image Retrieval best result;



2. Head Detection and Classification;



7. Future purposes

Xception is a powerful but complex architecture. Since we had limited time and computational resources, we only proposed 2 variation of Xception, but other possible changes could be applied to further improve final result. For example, a combination of our LighterXception and MidwayInception could be implemented. In addition, the hyperparameters should be optimized for our customized architectures, especially for the optimizer and learning rate choices, which play a fundamental role on the final results.

8. Conclusions

In general, deepfake detection will be more challenging in the future, due to the fact that deepfake generation makes deepfake more realistic and easier to make. Moreover, deepfake has been a significant threat to national security, democracy, society, and our privacy, which calls for deepfake detection methods to combat potential threats. For these reasons we deeply believe that research must continue on this way.

References

[1] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. pages 1800–1807, 07 2017.

[2] Università di Catania. Deepfake images detection and reconstruction challenge, 2022. 21st International Conference on Image Analysis and Processing.

[3] Christopher G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[6] Momina Masood, Marriam Nawaz, Khalid Malik, Ali Javed, and Aun Irtaza. Deepfakes generation and detection: state-of-the-art, open challenges, countermeasures, and way forward. 02 2021.

[7] Tan Mingxing. Efficientdet: Scalable and efficient object detection. 07 2020.

[8] Thanh Thi Nguyen, Cuong M. Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection. *CoRR*, abs/1909.11573, 2019.

[9] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images, 2019.

[10] Muhammad Saqib, Sultan Khan, Nabin Sharma, and Michael Blumenstein. Person head detection in multiple scales using deep convolutional neural networks. pages 1–7, 07 2018.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[12] Karras Tero. A style-based generator architecture for generative adversarial networks. 11 2018.

[13] Karras Tero. Analyzing and improving the image quality of stylegan. 12 2019.

[14] Cho Wonwoong. Image-to-image translation via group-wise deep whitening-and-coloring transformation. 06 2019.

[15] Choi Yunjey. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. 11 2017.

[16] Ying Zhang, Lilei Zheng, and Vrizlynn L. L. Thing. Automated face swapping and its detection. In *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*, pages 15–19, 2017.

[17] He Zhenliang. Attgan: Facial attribute editing by only changing what you want. 11 2017.