Deep and Reinforcement Learning

Assignment 1 Report

MSc in the Artificial Intelligence

Group 6
Michal Kowalski - up202401554@edu.up.pt
Pedro Azevedo - up201905966@up.pt
Pedro Pereira - up201708807@edu.up.pt

Faculdade de Ciências da Universidade do Porto

April 7, 2025

Contents

List of Figures

1 Introduction

For the first assignment of the class "Deep and Reinforcement Learning" we were challenged to develop deep learning discriminative and generative models, applied in the context of human *deepfakes*. Meaning the detection and generation of human images. We decided to approach the task by from two angles. The first, was the exploration and implementation of a GAN framework [Goodfellow et al., 2014] aimed at generating highly realistic deepfakes. The second, developing an independent classifier model to distinguish between real and fake face images. In this report, we will describe our iterative process of refining each model, the experiments conducted, and the evaluation results, followed by our conclusions and potential directions for future work.

2 Datasets & Processing

The dataset provided for the task was DeepFakeFace from [Song et al., 2023], which includes 30,000 real images of celebrities taken from the IMDB-WIKI dataset, as well as 90,000 fake images generated using three different diffusion models: Stable Diffusion v1.5, Stable Diffusion Inpainting, and InsightFace, with each model creating 30,000 fake images. The dataset consists of both face and full-body images (Figure 1).

Figure 1: Datasets

2.1 Overview & Initial Processing

The generated data with fake faces was already, as expected, standardized. All files being in the JPG format, RGB encoded, 512x512 pixels. The real data had variations across all the categories mentioned previously and required standardization. To be able to train sufficiently complex generator however, we had to scale down the images to 64x64 pixels.

In the case of the classifier model, the preprocessing required more flexibility in handling image sizes. The images were also standardized in terms of JPG format and RGB encoding, but the number of images varied depending on the specific model and the changes made during preprocessing. Moreover, due to the class imbalance in the dataset, we implemented an algorithm to augment images generated from the real face dataset to ensure the proper classes distribution.

2.2 Dataset Extensions

After some experimentation with different techniques to try to improve the generator with just the base dataset, we explored extending the dataset with two other sources. The first we considered was the CelebA dataset [Liu et al., 2015]. This dataset, however, solves a slightly different problem than the original, as this one contains only face images. With that, the images produced are visually more appealing, which gave us the idea to experiment with processing the original data by extracting the faces. Having achieved at least visually similar results with the face extraction on the original dataset, we decided against further testing the CelebA dataset.

The second extension we considered was, in fact, the source of the real images in the provided dataset, IMDB-WIKI, [Rothe et al., 2015]. Preliminary experiments showed no significant performance improvements, which we attribute to hardware limitations, meaning we could not deepen the models and had to use small 64x64 pixel images which may distort the features the model is supposed to learn. Because of this, we believed data augmentation would similarly have little to no impact, and so elected to focus on other improvements for the generative model.

2.3 Images Augmentation

Class imbalance can significantly affect the training of reliable models, making the task more challenging. A potential solution to this problem is data augmentation, which not only increases the size of the data set but also introduces diverse variations that can enhance the model's robustness and prevent overfitting. There are several methods to generate additional image data: geometric transformations, color image processing, and intensity modifications. Geometric transformations modify the spatial relationship between pixels, including affine transformations or elastic deformations, while color image processing focuses on varying the color properties of the input images [Xu et al., 2023].

In our classifier, we applied several data augmentation techniques to increase the data set size and improve model performance. These included horizontal flipping with hue adjustment, brightness and hue changes, and zoom augmentation. The flipping and hue adjustment added diversity by modifying the image orientation and color. Brightness and hue adjustments simulated different lighting conditions, while zoom augmentation varied the image size and perspective. These techniques significantly expanded our dataset, helping to boost the performance of our models.

Figure 2: Original Image vs Image after using YOLOv8

2.4 Faces Extraction

While working on improving the performance of our models, we came across the possibility of using a lightweight face extractor to assist in deepfake detection. This approach enables data quality improvement by focusing solely on human faces, ignoring the background, which may introduce noise and complications when extracting key features that help distinguish real from fake human faces.

In that case, we used the YOLOv8 model [Ultralytics, 2024], developed by Ultralytics, which is an advanced object detection algorithm that enhances previous YOLO versions, providing improved speed, accuracy, and user-friendliness [Zhang et al., 2024]. This function efficiently detects the first face found in each image, crops it from the original image, and resizes it to a standardized size of 200x200 pixels (Figure 2). It also ignores faces that are too small (below 50x50 pixels) to maintain the quality of the processed data. The resulting cropped and resized images are saved in the specified output folder. Some images were dropped because of difficulties in detecting faces. However, the balance between real/fake images was not significantly disrupted during the process.

3 GAN Modeling

In this section, we will introduce the baseline Generative Models used for this project.

3.1 Baseline Generative Model

The GAN framework was introduced in 2014 by [Goodfellow et al., 2014]. The main idea is to train a generative model, by simultaneously training discriminator model that attempts to detect if a data entry is real or generated and having the generator train to deceive the discriminator. As the discriminator learns what to look for in the real examples, the generator learns how to generate more convincing fakes. It is a framework and not a model, because it can be applied with any underlying architecture. That being said, the original paper uses Multilayer Perceptrons for both models.

In 2016, a new architecture for the GAN framework [Radford et al., 2015] was proposed that has since become the standard [Goodfellow, 2016], the DCGAN or Deep Convolutional GAN. Based on Deep Convolutional Neural Networks rather than Multilayer Perceptrons. We use this as our baseline generative model.

From [Radford et al., 2015], the rules for building a stable DCGAN are as follows:

- Remove fully connected layers;
- Replace pooling layers with strided (discriminator) and fractional-strided (generator) convolutions;
- Use batch normalization;
- Use ReLU activation for all layers of the generator except for the output which uses *tanh*
- Use LeakyReLU for all layers of the discriminator;

For our first implementation we followed an official TensorFlow Tutorial on DCGANs [TensorFlow, 2024]. Curiously this tutorial replaces the ReLU's on

the generator for their leaky variants, but doesn't provide a reason as to why. Not wanting to deviate from the original architecture for no apparent reason, we monitored the gradients on the network in training, and found no reason to use the LeakyReLU. What we did not realize soon enough is that while it claims to follow a DCGAN architecture, but the fact they use a Dense layer as the first layer of the generator seems to be inspired by the FCC-GAN[Barua et al., 2019]. A fact we took some time to acknowledge, and unfortunately, at the point of writing this report, are unable to fix in our experiments. That being said having a Dense layer in the generator, as was proposed in the FCC-GAN, could enrich the images generated by creating more variations from the input noise vector.

The discriminator employs the following architecture:

- 4, 2-d Convolutional Layers, kernel size = 4, strides = 2
- 1, 2-d Convolutional Layers, kernel size = 4, strides = 1
- Between 2 and 4th convolutional layers apply batch normalization
- Layers 1-4 use LeakyReLU activation
- Layer 5 uses sigmoid, to bind result between 0 and 1

The generator employs the following architecture:

- 1 Fully Connected Layer with $8 * 8 * 128$ nodes
- 2 2-d Convolutional Transpose Layers, $(5, 5)$ kernel, with $(2, 2)$ strides
- Batch Normalization and ReLU activation between hidden layers
- 1 final 2-d Convolutional Transpose Layers, $(5, 5)$ kernel, with $(2, 2)$ strides, with *tanh* activation

Both models make use of the ADAM optimizer with base learning rates of $1e - 4$.

3.2 Model Improvements

This section will discuss the various improvements considered for the GAN models. Because our baseline was just barely manageable by our hardware, we focused on performance-friendly improvements.

3.2.1 One-Sided Label Smoothing

Label Smoothing is a technique proposed to help deep learning classifiers better generalize to new examples [Szegedy et al., 2016]. In classifiers without label smoothing we typically denote the true class with a "1" and the remaining with "0". In this case, minimizing cross-entropy loss can be seen as maximizing the log-likelihood of the correct label, which in practice is approximated if the ground-truth logit is much bigger than the remaining classes'. This can lead to overfitting which reduces the model's ability to adapt to new data. With label smoothing, the incorrect labels are smoothed to an ϵ, and the correct label to $1 - \epsilon$. This makes the model less confident in it's predictions but theoretically should improve the discriminator, and a better discriminator should force the generator to be better by being harder to deceive. Now because we still want the generator to aim for the best/most deceiving images, we only apply this label smoothing on the discriminator loss. [Goodfellow, 2016]

3.2.2 Wasserstein GAN with Gradient Penalty: WGAN-GP

Wasserstein GANs replace the binary cross entropy loss functions of GANs with the Wasserstein (or Earth Mover's) distance between the probabilities of the fake and real distributions [Arjovsky et al., 2017]. This was proposed to stabilize training, but in [Gulrajani et al., 2017], the authors state that the original weight clipping method was still prone to not converging, and proposed an alternative method to stabilizing training: Gradient Penalty. This technique, instead of simply thresholding gradients, which is a crude way of controlling update impact, smoothens the gradients with a penalty that still allows some of the original impact through.

3.2.3 Two Timeline Update Rule: TTUR

TTUR was originally developed for actor-critic learning, a Reinforcement Learning framework [Prasad et al., 2015] with some parallelisms to the GAN framework. The original idea was to have the critic have a higher learning rate so that the actor does not get stuck training with a weak critic. In [Heusel et al., 2017], the authors show how GANs can also benefit from this paradigm. They experiment with both DCGANs and WGANs. Curiously, and somewhat contrary to the original idea, their results indicate that for DCGANs it is often better that the generator has a higher learning rate than the discriminator ($1e-4/1e-5$). For WGANs optimal results were achieved with parameters more in line with the original implementation ($1e-4/3e-4$).

4 Generative Models Experiments

4.1 Setup

Our experiments cover the following models, and datasets:

- Baseline GAN, DeepFakeFace
- Baseline GAN + Label Smoothing, DeepFakeFace
- Baseline GAN + TTUR ($1e-4$ generator, $1e-5$ discriminator), DeepFakeFace
- WGAN-GP, DeepFakeFace
- WGAN-GP + TTUR ($1e-4$ generator, $1e-5$ discriminator, similar to test on Baseline), DeepFakeFace
- WGAN-GP + TTUR ($1e-4$ generator, $3e-4$ discriminator, more appropriate for WGAN according to [Heusel et al., 2017]), DeepFakeFace
- WGAN-GP, DeepFakeFace with Face Extraction

Before the experiments, we split the DeepFakeFace Real Images Dataset 75/25 with 25000 images for training and 5000 for testing. We used the same splits when doing Face Extraction and lost about 3000 images on the training data and 1000 images in the testing data due to no faces being successfully extracted. We used a batch size of 128.

Since the task on the original data proving more complex than on face data only, we trained these models for 200 epochs. The one trained on the face dataset was only trained for 40 epochs as it was quicker to show better results, and we ran out of time.

4.1.1 Training

During training we plot the evolution of both model's loss, which we will go into in section 4.2 when discussing the improvements of each new technique. We also monitored gradients in generator to detect vanishing or exploding gradients.

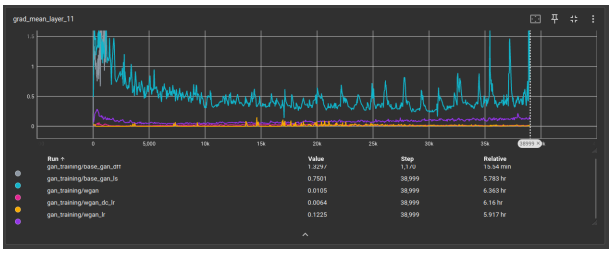

Figure 3: Mean gradients at last layer for the models trained on DeepFakeFace

As can be seen in the picture, no experiment, except maybe for the WGAN with and $1e-4, 1e-5$ learning rates, shows signs of vanishing or exploding gradients, seeing as their means are a bit bigger than 0. Notice we are missing the gradients for the test on the DCGAN with the different learning rates as we accidentally deleted them before the delivery while renaming the folders before taking the graph image.

4.1.2 Evaluation Metrics

We evaluate the generative models with the FID metric. Introduced in [Heusel et al., 2017], the idea is, given 2 sets of images, one real, one generated, run the sets through the InceptionV3 model, extract their feature maps and compare the sets through their means and covariance matrices combined into one number the Fréchet Inception Distance.

4.2 Results

We compare our results to the other models present in the DeepFakeFace dataset. And apply Face Extraction to these as well to compare to the models with that preprocessing step.

4.2.1 DeepFakeFace Dataset

Table 2 compiles the results of experiments on the original DFF dataset.

Interestingly, only the application of the WGAN-GP loss had a positive impact on the overall FID, which contradicts most of the literature referenced in this work. That being said, even the best model was not visually appealing, as can be seen in the image bellow of images generated at epoch 200 for the WGAN model.

What's more comparing the loss curves for the Baseline models, the loss remains simillar for the Label Smoothing example and actually decreases for the TTUR experiment. The latter however was showing a trend to grow for the generator which may indicate it either hadn't converged yet or was starting to overfit.

Model	AVG FID 100 batches
Insight	131.99
Inpainting	135.66
Text2Img	164.62
Baseline	207.95
Baseline + LS	266.97
Baseline + TTUR	319.58
WGAN-GP	192.60
WGAN TTUR $1e-4, 1e-5$	253.69
WGAN TTUR $3e-4, 1e-4$	247.06

Table 1: Performance metrics for different models on DeepFakeFace

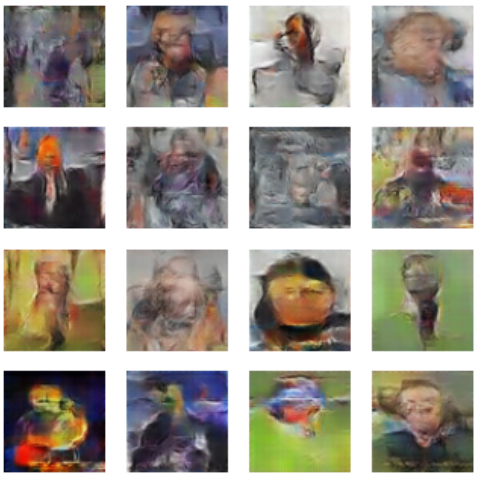

Figure 4: Images Generated by WGAN after 200 epochs of training of DFF

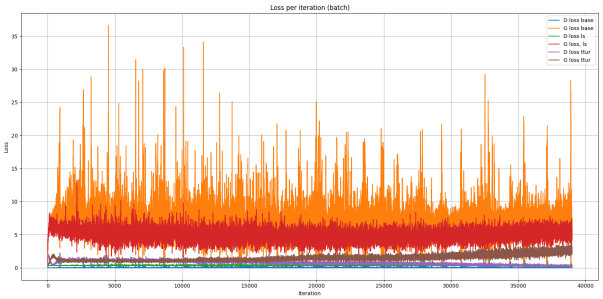

Figure 5: Plot of Training Losses for Baseline model and non-WGAN variants

4.2.2 Face Extraction Dataset

In this section we report only the results of the best model from then last experiment being the WGAN-GP model and compare them to the best of the pretrained models.

Model	AVG FID 100 batches
Insight	67.44
WGAN-GP	136.899

Table 2: Performance metrics for models on DeepFakeFace with Face Extraction

Interestingly, the performance gain is more significant in the pretrained model, though that may be explained by the fact that this model was trained on fewer epochs due to time constraints. Still the results visually are much more interesting even on less time to train.

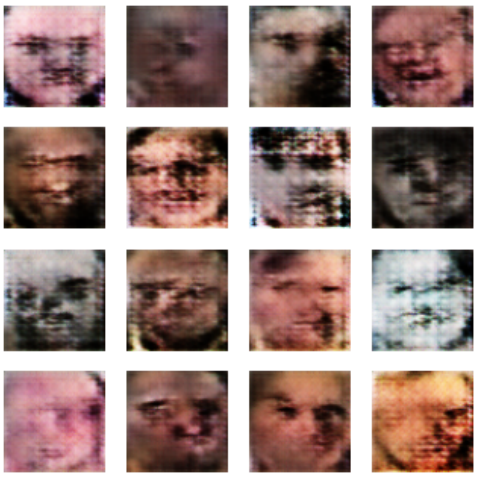

Figure 6: Generated image after 40 epochs of WGAN on DFF with Face Extraction

5 Discriminator Modeling

In this section, we will introduce the baseline Discriminator Models used for this project.

5.1 Baseline Discriminator Model

Before CNNs, image recognition techniques depended mostly on artificial design features, which were limited to capture the medium and low-level image patterns and it is difficult to extract the deep-level information of the image [Xiao, 2024]. CNNs use deep neural networks and convolutions to analyze, learn, and interpret data as a human being. It has strong generalization abilities, allowing to capture more complex image features. Today, CNNs are widely used in various applications, including image classification.

For the implementation of our first classifier model, we used the TensorFlow Core Guideline [TensorFlow, 2025] as our main reference. This guide outlines

a standard machine learning workflow, which includes examining and understanding the data, building an input pipeline, constructing the model, training it, testing its performance, and iteratively improving the model based on the results. These information, combined with our own knowledge and experience, formed the basis for developing the first CNN model for classifying real or fake face images, with the ability to fine-tune the model to effectively handle the task.

Key Features of the First CNN Model:

- The model consists of 2 *Conv2D* layers, followed by a *Flatten* layer and 1 *Dense* layer with 128 units.
- *MaxPooling* is used to reduce spatial dimensions, and *BatchNormalization* is applied for stable training.
- Use *ReLU* activation for all layers of the generator except for the output which uses *Sigmoid* activation for binary classification (real or fake).
- The dataset is balanced, image normalization was applied, and the model was trained with a batch size of 64.

5.2 Models Improvements

This section will cover the different improvements we explored for the Discriminator Models. Since our baseline was barely manageable with our hardware, we prioritized performance-friendly enhancements.

5.2.1 Data Quality Adjustment

During the evaluation of the classifiers, our main task was to properly adapt the data for efficient and effective training. To achieve this, in addition to standardization, TensorFlow enables data augmentation during the training, a technique used to increase the diversity of the train set by applying random (but realistic) transformations, such as random image rotation, shear, zoom, and horizontal flipping. Additionally, the image size needs to be adjusted carefully to strike a balance between reducing dimensionality and retaining the most important information. To achieve this, we also tried to use the Yolov8 model mentioned in section 2.4 to extract faces from the images and remove the background that might introduce a noise. Moreover, we adjusted the batch size according to the available resources to ensure efficient training without overloading the system.

5.2.2 Deeper CNN Model

Face recognition is widely considered one of the most challenging technical because of the computational complexity. It is essential to appropriately adjust the depth of multilayer models to efficiently capture the problem of extracting relevant patterns from images. Initially, our training was unstable and resulted in unsatisfactory predictions. Therefore, it was necessary to modify the number of layers, either by adding new ones or removing existing ones, and comparing the model's performance.

5.2.3 Learning Rate Decay

Learning rate decay is a commonly used technique for training modern neural networks. It begins with a high learning rate and gradually reduces it over time. It is empirically observed to help both optimization and generalization [You et al., 2019]. Therefore, it was also used during the process of improving our modeling.

5.2.4 L2 Regularizer

L2 regularization is one of several regularization techniques used in linear regression models. It is a statistical method aimed at reducing errors caused by overfitting the training data [IBM, 2025]. By adding a penalty to the magnitude of the model's coefficients, it discourages overly complex models that fit the noise in the training data. This helps improve generalization to unseen data and leads to a more robust model.

5.2.5 Pretrained Models Tuning

ResNet50 is a deep Convolutional Neural Network architecture consisting of 50 layers, which revolutionized deep learning by introducing residual blocks with skip connections. These connections allow for the training of significantly deeper networks by addressing the vanishing gradient problem through the learning of residual functions rather than direct mappings. Initially designed for image classification, its ability to learn powerful feature representations has led to its widespread adoption in various computer vision tasks. ResNet50 is often used as a pre-trained backbone on large datasets like ImageNet, before being fine-tuned for specific applications [He et al., 2016].

6 Discriminator Models Experiments

In this section, we discuss the results of the experiments with the discriminator model, including the parameters used and an evaluation of the outcomes we achieved.

6.1 Setup

For image recognition, we used TensorFlow 2.11.0 together with the Keras API for building and training models, but also for efficient image processing and augmentation. The environment was set up with Python 3.10.1, using Pandas for data handling and PIL Image for image preprocessing. Matplotlib was used to visualize results. The project was run in a Jupyter notebook.

6.2 Evaluation Metrics and Loss Function

In evaluating the model's performance, we focused on four main metrics: accuracy, precision, recall, and loss function. Accuracy reflects the overall effectiveness of the model in classifying images as real or fake. Precision measures how well the model identified fake images, avoiding the misclassification of real ones as fake. Recall shows how well the model detects fake images, minimizing false negatives. The loss function, binary cross-entropy, calculates the difference between the model's predictions and the actual labels, helping the model optimize. We chose these metrics because they provide a more comprehensive understanding

of the model's performance, according to the model ability to detect deepfakes. The graphs of these metrics also provide tracking of the training progress over time.

6.3 Training, Validation and Test Sets

For Models 1-3, a dataset consisting of 63,000 images for training, 13,500 for validation, and 13,500 for testing was used, with pixel values normalized to the range [0,1]. The dataset was split in a 70/15/15 ratio due to the large amount of data. In Model 1, the images had a size of 200x200 and were divided into batches of size 64. In Model 2, the images were resized to 224x224 with a batch size of 64, and in Model 3, the images were also resized to 224x224, but with a batch size of 128.

For Models 4-5, 30,000 real images and 30,000 fake images were subjected to face detection using YOLOv8, which resulted in the loss of approximately 5,800 real images and 10,000 fake images. Despite this, the class imbalance was accepted for the purpose of model creation. Finally, the dataset was normalized to [0,1], resized to 200x200 and split again in a 70/15/15 ratio for training, validation, and testing using batch size of 128.

For Model 6, the training set contains 42,000 images and 9,000 images each for validation and testing, also resized to 224x224. The batch size was 64, and the data split was as previously.

6.4 Models

In this section, we present the architectures of the models used in the experiment, along with the modifications made to improve classification results.

6.4.1 Models 1 - 3

The first model starts with two Conv2D layers, each using ReLU activation, followed by BatchNormalization and MaxPooling2D layers to help extract relevant features from the images. All these layers use ReLU activation. After that, a Flatten layer is used to convert the 2D output into a 1D vector. Then, there's a Dense layer with 128 units and ReLU activation, followed by a Dropout layer set to 50% to avoid overfitting. Finally, the output layer uses a sigmoid activation for binary classification. The model was trained for 10 epochs, and the performance was measured using established metrics.

The second model introduces random data augmentation during training to improve generalization and prevent overfitting. Additionally, a 25% Dropout layer is added after the CNN layers to further regularize the model and reduce the risk of overfitting. These changes aim to improve the model's ability to generalize on unseen data.

With the third model we tried using a deeper neural network by adding an additional CNN and Dense layer. We also modified the Dropout rates and introduced learning rate decay to stabilize and accelerate the training process. These changes aim to enhance the model's ability to learn more complex features and improve overall performance.

6.4.2 Models 4 - 5

Models 4 and 5 used only face-detected images. Based on the positive results from Model 3, the architecture of Models 4 and 5 was built upon the

same structure, with intuitive adjusting the dropout rate. For Model 5, an L2 regularizer with a rate of 0.01 was added to each layer (instead of the output layer) to prevent overfitting and improve model generalization. Moreover, we used 5 more epochs than usual.

6.4.3 Model 6 (ResNet50)

In Model 6 we used ResNet50 with pre-trained weights from ImageNet as the base model, with the top layers excluded. The layers of the base model were frozen to avoid retraining them. After that, a GlobalAveragePooling2D layer was added, followed by a Dense layer with 256 units and ReLU activation. A Dropout layer with a rate of 50% was included to prevent overfitting, and the final output layer used a sigmoid activation as well. The model was trained for 8 epochs, with learning rate decay to stabilize and speed up the training process.

Model	Accuracy	Loss	Precision	Recall
Model 1	0.99	0.06	0.99	0.99
Model 2	0.59	0.98	0.55	0.99
Model 3	0.86	0.35	0.92	0.80
Model 4	0.77	0.50	0.71	0.96
Model 5	0.67	0.89	0.63	0.97
Model 6	0.60	0.66	0.59	0.72

Table 3: Performance metrics for different models

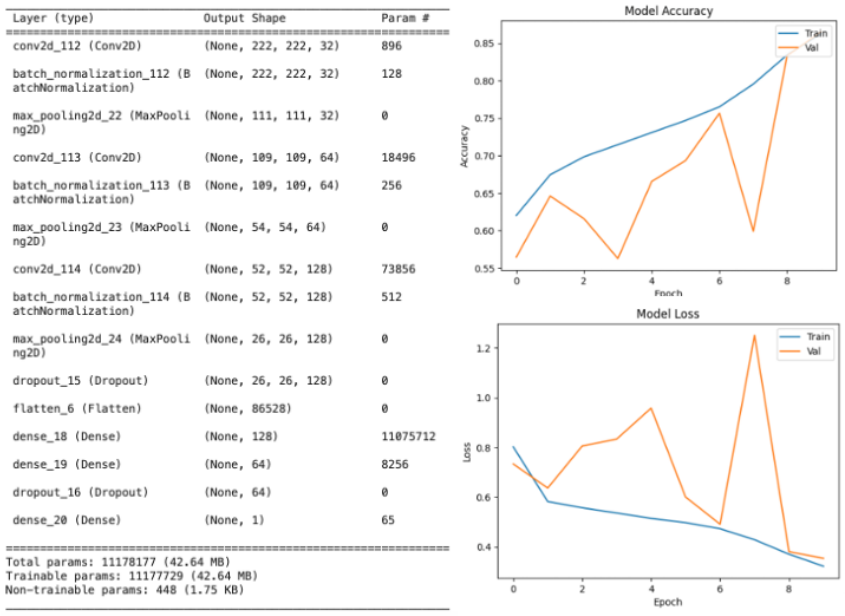

Figure 7: Model 3 structure and performance

6.5 Results

In this section, we present the results of the models during the experiments. The table 3 summarizes the performance metrics, including accuracy, loss, precision, and recall, for each model.

During the training of Model 1, we noticed that the training accuracy (around 74%) stayed almost the same throughout the epochs, which suggests underfitting. This likely means the model is too simple or hasn't been handled the classification problem properly. However, the validation and test accuracy were both unexpectedly higher than with the training set (98.97% in epoch 10), indicating that models perfectly generalize to unseen data but struggles during training. This inconsistency seems irrational, so we made changes in the setup for our next models.

In the second model, after applying data augmentation during training and increasing the image size to retain more information, the training process became much more stable, and the training accuracy increased with each epoch, indicating better classification performance. In later epochs, high recall was achieved for the training, validation, and test sets, while precision hovered around 0.6 for unseen sets. A precision value of 0.6 suggests that the model struggles to accurately predict the positive class and often classifies negative instances as positive. The high recall indicates that the model identifies most positive cases, but its ability to avoid false positives remains limited. The test accuracy was still relatively low at 0.59.

Among the custom models, Model 3 (Figure 7) produced the best results, achieving an accuracy of 0.86, a loss of 0.35, precision of 0.92, and recall of 0.80. The training metrics remained stable throughout, while the validation metrics were slightly more variable, likely due to the limited number of epochs. The deeper CNN architecture, combined with learning rate decay, contributed to the model's superior performance in deep fakes recognition. We assume that increasing the number of epochs during training would lead to an improvement in the model's accuracy.

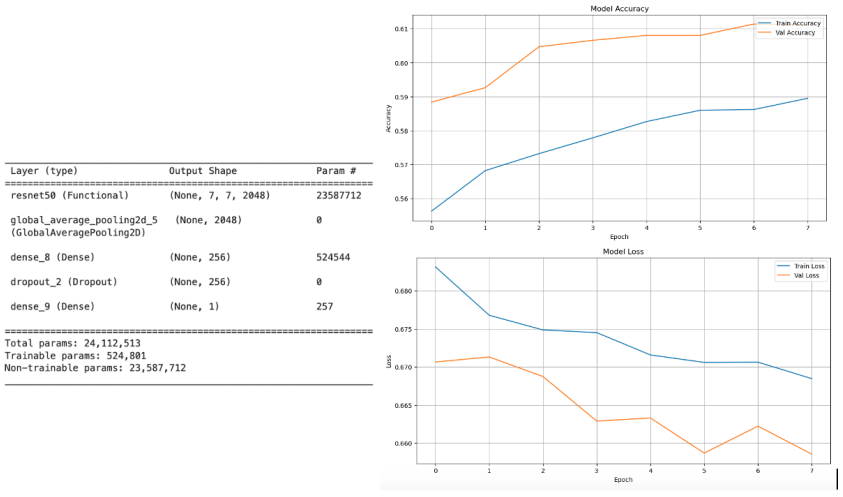

Figure 8: Model 3 structure and performance

After a more detailed preprocessing of the images, including faces detection and cropping, both models 4 and 5 showed a stable learning process (Figure

8), with noticeable improvements in accuracy and a decrease in loss for both training and validation data. Model 4 achieved significantly better results on unseen data, with accuracy of 0.77, loss of 0.50, precision of 0.71, and recall of 0.96, compared to model 5. Model 5, which used L2 regularization, likely overgeneralized the classification problem, resulting in worse performance. It means that by properly handling the data, the models can learn faster and more effectively, leading to better results.

In the Model 6 we used pre-trained layers from the ResNet50 network and applied fine-tuning to adjust our model for the deepfakes detection. This allowed us to take an advantage of the knowledge gained during previous training on a large amount of images for faster learning and better generalization. The phenomenon in which the validation and test metrics were higher than the training metrics in this case was likely caused by the use of too high a dropout rate or too few epochs. Furthermore, adjusting the learning rate could have affected the learning speed, and increasing the number of epochs would have improved the overall performance of the classifier.

7 Conclusion

The results of the generator on the original dataset highlight some of the difficulties of training GANs as only the WGAN showed any positive impact on the base dataset. Theoretically, most of the changes implemented should have improved the results, however, maybe due to the fact that the generator has a fully connected layer interfered with the expected gains. As those improvements were developed in the literature for DCGANs and DCGAN based WGANS.

We would have liked to experiment further with the Face Extraction dataset and the TTUR improvement, however, time constraints do not allow it. During this project our time management was put under test as training each model was a long process often spanning over 6 hours, we take this as a lesson for future work.

Had we caught the issue with the generator architecture not being exactly DCGAN, we would have liked to experiment with the canonical form of this model. That being said, the hybrid model between a DCGAN and an FCC-GAN still showed interesting results, even if not comparable to state of the art diffusion models.

In this project, we tested CNNs for deepfake detection and got a solid 86% accuracy with our custom model. However, we quickly realized that image processing in deep learning needs a lot of computational power and time. Training models from scratch was super demanding, and as we tried more complex models, the hardware limitations we had became a real bottleneck. If we had more resources, it would make sense to explore deeper pre-trained models like ResNet50 or VGG16. These models could speed up the training process and potentially improve our results. Fine-tuning them could really make a difference, helping us take our deepfake detection to the next level.

Additionally, it's important to focus on the image data itself. We need to find methods that keep the best balance between image quality, dimensionality reduction, and standardization. Finding this trade-off could help improve the model's performance by simplifying the data without losing important details.

References

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

[Barua et al., 2019] Barua, S., Erfani, S. M., and Bailey, J. (2019). Fcc-gan: A fully connected and convolutional net architecture for gans. *arXiv preprint arXiv:1905.02417*.

[Goodfellow, 2016] Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, 30.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

[IBM, 2025] IBM (2025). Ridge regression. Accessed: 2024-04-07.

[Liu et al., 2015] Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738.

[Prasad et al., 2015] Prasad, H., LA, P., and Bhatnagar, S. (2015). Two-timescale algorithms for learning nash equilibria in general-sum stochastic games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1371–1379.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[Rothe et al., 2015] Rothe, R., Timofte, R., and Van Gool, L. (2015). Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 10–15.

[Song et al., 2023] Song, H., Huang, S., Dong, Y., and Tu, W.-W. (2023). Robustness and generalizability of deepfake detection: A study with diffusion models. *arXiv preprint arXiv:2309.02218*.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[TensorFlow, 2024] TensorFlow (2024). Deep convolutional generative adversarial network. Accessed: 06-04-2025.

[TensorFlow, 2025] TensorFlow (2025). Image classification with tensorflow. Accessed: 2025-04-07.

[Ultralytics, 2024] Ultralytics (2024). New-yolov8 in pytorch ¿ onnx ¿ openvino ¿ coreml ¿ tflite. Accessed on 26 June 2024.

[Xiao, 2024] Xiao, M. (2024). Cnn advancements and its applications in image recognition: A comprehensive analysis and future prospects. *Applied and Computational Engineering*, 46(1):116–124.

[Xu et al., 2023] Xu, M., Yoon, S., Fuentes, A., and Park, D. S. (2023). A comprehensive survey of image augmentation techniques for deep learning. *Pattern Recognition*, 137:109347.

[You et al., 2019] You, K., Long, M., Wang, J., and Jordan, M. I. (2019). How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*.

[Zhang et al., 2024] Zhang, R., Deng, B., Cheng, X., and Zhao, H. (2024). Gcs-yolov8: A lightweight face extractor to assist deepfake detection. *Sensors*, 24:6781.