Real or Fake? An In-Depth Exploration of Faces in the Era of Deepfakes

BARRACHIN Carlyne

Abstract – With the rise of artificial intelligence, the distinction between real and faked images has become a pressing challenge, particularly with the rise of deepfakes. This report explores a dataset of 70k StyleGAN-generated images and 70k real images, evaluating three deep learning architectures: EfficientNetB0, ResNet50 and Vision Transformer (ViT). These models, representing convolutional neural networks (CNNs) and transformer-based approaches, were trained and evaluated using Keras and PyTorch. The results demonstrate EfficientNetB0's superior balance between performance and computational efficiency, achieving an accuracy of 97.17%. ResNet50 also performed well, while ViT showed its potential, but was limited by its dependence on large computational resources. The results highlight the challenges of training models on diverse datasets to ensure generalizability to a variety of synthetic image sources, as well as the potential of hybrid architectures for better fake image detection.

Keywords – Deepfakes, CNNs, AI-generated images, EfficientNetB0, Vision Transformer (ViT), ResNet50

1. INTRODUCTION

Today, it's difficult to believe everything we see in the media and on social networks, especially with the increasing use of AI-generated images. This problem is particularly relevant when it affects privacy as AI makes it possible to **generate a realistic face** of a known or unknown person. It is then easy to use it in fictional situations, often with deceptive or manipulative intentions. The main use of **deepfakes** lies in simulating people in specific contexts, saying or doing things they have never actually done. In a context where social networks play an important role in everyone's life, media manipulation is becoming a powerful tool for influencing public opinion, spreading false information or damaging the reputation of individuals and organizations.

One of today's main challenges is differentiating real from fake faces. It can be difficult to do with the naked eye, especially with advanced models like **StyleGAN** [1] a neural network that generate ultra-realistic synthetic images.

The aim of this study is to detect synthetic faces by comparing three models: **EfficientNetB0** [2], **ResNet50** [3], and **Vision Transformer** (ViT) [4]. This involves examining two types of architectures: the first two models are **convolutional neural networks** (CNN), while the third is based on the **Transformer** architecture. A **CNN** is designed to process grid-like data, such as images, by applying convolutions to extract important features. A **Transformer**, originally developed for natural language processing, relies on a mechanism called attention, which captures global relationships between different parts of the input data. This approach will enable an evaluation of the effectiveness of these different architectures on the same dataset comprising **140k face images**.

This report explores several aspects, including the implementation and training, as well as a comparison of the frameworks used: **Keras** and **PyTorch**. Finally, it presents the performance of these models on a large scale, followed by a critical discussion of the results obtained.

2. PREVIOUS WORK

EfficientNet, presented by Tan and Le in 2019 [2], represents a major advancement in convolutional neural networks (CNNs) for image classification. EfficientNet models, especially EfficientNet-B7, have achieved top-1 accuracy of 84.3% on ImageNet, while being 8.4 times smaller and 6.1 times faster than models like GPipe. Additionally, EfficientNet has excelled in transfer learning, delivering state-of-the-art results on datasets such as CIFAR-100 and Flowers, using significantly fewer parameters than architectures like ResNet or NASNet. Its ability to generalize

tasks makes it highly effective in object detection, medical imaging, and synthetic image detection applications, including deepfake detection, making it a prime candidate for distinguishing real and synthetic faces.

The Vision Transformer (ViT), presented by Dosovitskiy et al. (2021) [4], demonstrates the potential of applying the Transformer architecture to image recognition tasks. Unlike traditional CNNs, ViT treats images as sequences of non-overlapping patches, much like words in natural language processing (NLP). By leveraging self-attention mechanisms, ViT captures global relationships between image patches, providing a deeper understanding of the image content. In benchmark tests, ViT outperforms CNNs like ResNet on large datasets such as ImageNet when pre-trained on JFT-300M, though it has limitations with smaller datasets. To address this, hybrid models that combine CNN feature extractors and transformers, like DeiT, have emerged, optimizing ViT for smaller datasets through techniques such as knowledge distillation. ViT has shown success in various applications, including object detection, segmentation, and multimodal tasks.

3. DATASET AND PREPROCESSING

3.1 Description

For my experiments, I retrieved a dataset from the open-access website **Kaggle** named '**140k Real and Fake Faces'**. It consists of **140k images** of **faces** collected by **Nvidia**. Half of the images, which are **real faces**, come from an image and video hosting service named **Flickr**. This set contains considerable variations in terms of age, ethnicity and image background. We also find a variety of accessories such as glasses, sunglasses, hats, etc. And the other half of the dataset, the **fake images,** part correspond to a sample of 1 million fake faces generated using **Nvidia's StyleGAN** [1].

3.2 Preprocessing

The images had already all been resized to **256 pixels** and split into **three** folders: **train**, **test** and **valid**. I have checked the **balance of classes** in each of them, so that no class is more represented than any other. This way, there will be no bias in favor of a majority class or under-learning for minority classes. **Figure 1** shows that the **Training Dataset** contains exactly 50k images of each class and that **Test** and **Validation Dataset** each contain 10k images for each class.

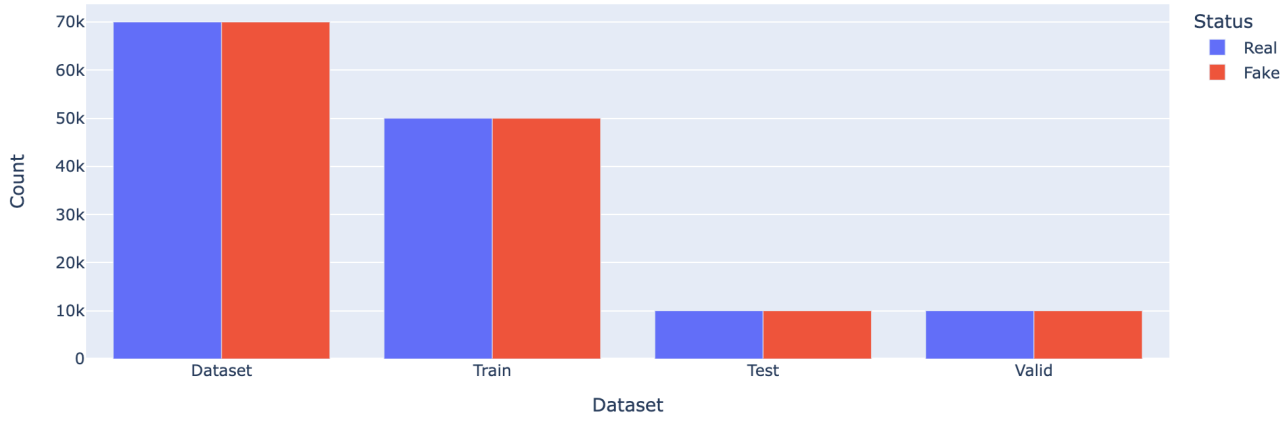

***Figure 1** - Distribution of Real and Fake Images in each Dataset*

These three folders are important for training and evaluating the model, helping prevent overfitting. The **training dataset** allows the model to learn complex relationships between images and their corresponding labels (Fake or Real). During training, the model is evaluated on the **validation dataset** to select the best parameters and avoid overfitting. This ensures the model generalizes well. After training and tuning, the model is tested on the **test dataset** to estimate its performance on unseen data.

3.3 Exploration

To classify the face images, I used a binary classification with **0** corresponding to **Fake** and **1** to **Real**. The figures below show examples of real and fake faces in each dataset.

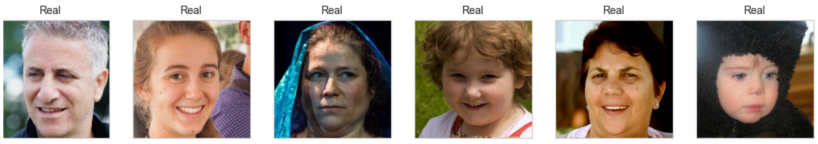

Figure 2 - Real** images from the **training dataset

Figure 3 - Fake** images from the **training dataset

Figure 4 - Real** images from the **test dataset

Figure 5 - Fake** images from the **test dataset

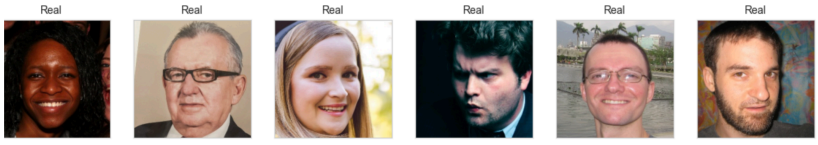

Figure 6 - Real** images from the **validation dataset

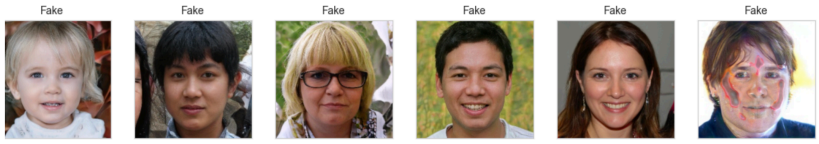

Figure 7 - Fake** images from the **validation dataset

4. METHODOLOGY

4.1 Models

I have retrieved from three references some information, detailed in ***Table 1***, on the three models used: **EfficientNetB0** [2], **ResNet50** [3] and **ViT** [4]. Each of these models has variants. For example, EfficientNet ranges from **B0 to B7**, with larger models offering better performance but consuming more resources. ResNet also has versions like **ResNet18 or ResNet152**. These different variants allow each architecture to be tailored to specific needs, whether for efficiency, robustness, or power.

FEATURE	RESNET50	EFFICIENTNETB0	VIT (BASE, PATCH16, 224)
YEAR INTRODUCED	2015	2019	**2020**
ARCHITECTURE TYPE	CNN (Deep Residual Network)	CNN (Convolutional Neural Network)	Transformer (Vision Transformer)
NUMBER OF PARAMETERS	25.6M	**5.3M**	86M
INPUT IMAGE SIZE	224 x 224	224 x 224	224 x 224
FLOPS	~4.1B	**~0.39B**	~17.6B
STRENGTHS	Robust deep learning	Energy-efficient	Excellent performance on **large-scale** data
WEAKNESSES	Heavy, resource-intensive	Less effective on complex large datasets	Requires large datasets for good generalization
TRAINING COMPLEXITY	Medium	**Low**	High
KEY STRUCTURAL FEATURES	**Residual connections**	NAS-based architecture	**Patch** embedding and global self-attention
NUMBER OF LAYERS	50 layers	234 layers	12 Transformer encoder layers
LAYER TYPES	Convolutional layers, **residual blocks**	**MBConv** blocks with depthwise separable convolutions	Self-attention, **MLP** layers, **LayerNorm**
ATTENTION MECHANISM			**Multi-head** self-attention (12 heads)

Table 1 *- Models Architecture comparison*

I'll now clarify some of the technical terms mentioned in the table.

First, the **FLOP** is a key metric that quantifies the computational demand of a model. **Layers** are the building blocks of a network where each layer transforms the data it receives by applying convolutions, normalizations, or activations. The more layers a network has, the better it can model complex relationships, but this can also make training more difficult.

ResNet50 is a CNN architecture with **layers** primarily composed of **convolutions** and **residual blocks**. These blocks help improving model's performances because the network learn more efficiently by allowing gradients to pass through the network more easily during training.

EfficientNetB0, another CNN, adopts an optimized design using **Neural Architecture Search** (NAS) which automatically identifies efficient layer configurations. It uses **MBConv** blocks, which are based on depth-separable convolutions to reduce the number of parameters and improve processing efficiency.

ViT, based on the Transformer architecture, uses **Transformer encoder layers** to perform global attention calculations and capture relationships between image patches. As shown in **Figure 8**, ViT splits an image into smaller patches, transforms them into vectors with positional information, and processes the data globally using attention mechanisms.

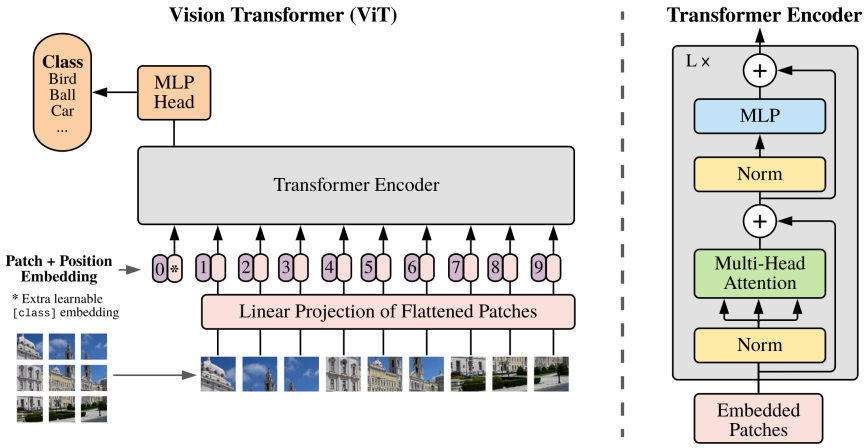

***Figure 8** - ViT model overview [4]*

4.2 Frameworks

4.2.1 Keras

Keras [5] is a high-level API built on top of TensorFlow. It is designed for humans thanks to its ease of use and understanding. It helps reduce cognitive load thanks to its intuitive API. Keras makes it easy to build, train and evaluate models, especially for beginners or rapid development. It simplifies the implementation of CNNs with prebuilt models available, allowing seamless customization and transfer learning.

4.2.2 PyTorch

PyTorch [6] is a framework with a highly customizable API, making it suitable for implementing advanced architectures such as vision transformers (ViT). The availability of ViT models in libraries such as **timm** simplifies implementation and experimentation.

I compared the two frameworks to see which was the easiest to use for this type of model and found that PyTorch was. Both frameworks offer excellent GPU acceleration, but PyTorch is often preferred in research for its adaptability for complex architectures like Transformers. Also, it allowed me to experiment with another framework.

4.3 Experimental Setup

4.3.1 Loss Function

$$L(y,p) = -(y\log(p) + (1-y)\log(1-p))$$

y: class label, **p**: predicted probability of the class being 1

I used the **Binary Cross-Entropy Loss** (BCE), a function loss used for binary classification problems. It is used during training to measure the dissimilarity between the predicted probability and the true class label.

4.3.2 Hyperparameters Optimization

To optimize the hyperparameters of each model, the **Adaptive Moment Estimation** (Adam) algorithm [8] was chosen. The aim is to help the model improve its predictions by automatically adjusting its hyperparameters to minimize the **loss function**. This algorithm, widely for neural networks, combines two classical optimization methods: **stochastic gradient descent with momentum** and **learning rate adaptation**.

Adam is built on **three key steps**: **Gradient tracking**, **Correction of Bias** and **Adaptation of fitting steps**. In simple terms, gradients represent the direction to follow to adjust parameters and reduce the loss function. Adam uses **two moments**: The **first moment**, or 'momentum', is the average of the previous gradients and helps guide the optimization in a consistent direction. And the **second**

moment, the average of the squared gradients, measures the variation in the gradients to prevent large and unstable updates.

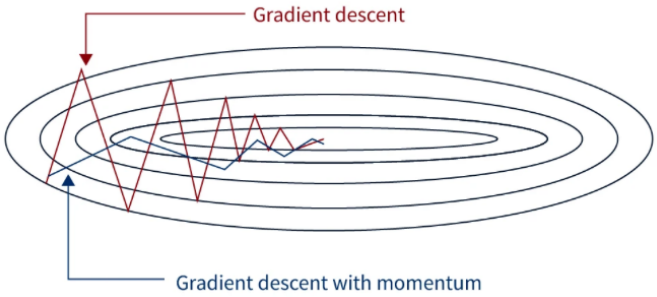

Figure 9 *- Gradient Descent Vs Gradient Descent with Momentum*

The difference with conventional gradient descent is that it updates parameters based on the current gradient, while gradient descent with momentum accumulates past gradients to smooth updates and speed convergence, as shown in **Figure 9**. This helps avoid oscillations or excessive adjustments.

After showing the chosen loss function and hyperparameter optimizer, we can see how each model has been configured.

4.3.3 Using Keras: EfficientNetB0 and ResNet50

The first step was to **resize the images** to ensure that all match the required input size. For this, I used **ImageDataGenerator**, a class provided by this framework, for image generation and pre-processing. Its use has also made it possible to convert images to **grayscale** for greater simplicity, and to classify them in **batches** with their labels. Train, valid and test folders are processed in a similar way in terms of image loading and pre-processing. The only thing that changes is the batch size. The batch size of **32**, used for training and validation, offers a good compromise between learning speed and memory efficiency. A larger one may speed up learning but may also increase memory usage, while a smaller batch size may result in slower convergence. I tried with a batch size of 100, but this took up a lot of memory and consequently a lot more time, so in view of my computer's performance I went down to 32. And the batch size of **1**, used for testing, means that each image is processed individually. This is useful for obtaining detailed predictions for each image, as well as for calculating metrics. Another difference is that for training and validation data is automatically **shuffled** by default to prevent the model from adapting to a specific data order.

After the processing step, I opted to load the model **without pre-trained weights**. Although pre-trained weights, such as those in large-scale models, can facilitate transfer learning by learning from previously acquired knowledge, I decided to train the model from scratch. I had a large enough dataset to train the model effectively without resorting to transfer learning and I wanted to evaluate the performance of each model when trained entirely on the new data, without any influence from previous optimizations.

Next, I added two layers to the model. Firstly, a **GlobalAveragePooling2D** layer reduces the dimensionality of the outputs, i.e. the size of the information extracted from the image to a single value. Then, a **Dense** layer with a sigmoid activation function produces a single output between 0 and 1, representing the probability for the binary classification task. We can see below the structured overview of the model architecture for each model.

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,048,991
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)	0
dense_1 (Dense)	(None, 1)	1,281

Total params: 4,050,272 (15.45 MB)

Trainable params: 4,008,253 (15.29 MB)

Non-trainable params: 42,019 (164.14 KB)

Figure 10 *- Structured overview for* ***EfficientNetB0***

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,581,440
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1)	2,049

Total params: 23,583,489 (89.96 MB)

Trainable params: 23,530,369 (89.76 MB)

Non-trainable params: 53,120 (207.50 KB)

Figure 11 *- Structured overview for* ***ResNet50***

Then, the models are trained for **five** epochs. During each epoch, the model updates its weights based on the loss calculated from the current batch of images. For the **EfficientNetB0 model** we can compare, for each epoch, the **training and validation accuracy** on **Figure 13**, as well as the training and validation loss on **Figure 12**. Calculating these measurements for each epoch is crucial to determine whether the model is overfitting, underfitting, or generalizing well. So, if accuracy or loss is much better than val_accuracy or val_loss, it suggests overfitting to the training data. If both training and validation metrics are low, it indicates underfitting, where the model fails to capture patterns. Here, the training and validation metrics are close to each other, showing good generalization to unseen data. Moreover, the loss values are tending to 0 and the accuracy ones are tending to 1 which shows that the model is improving its predictions over epochs with a closely matching between predictions and actual values.

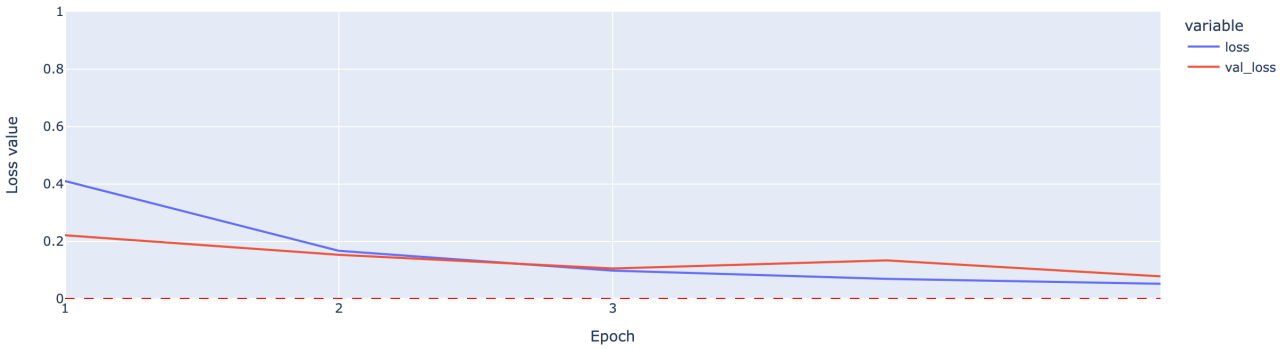

Figure 12 *- Training and Validation* ***Loss*** *for* ***EfficientNetB0***

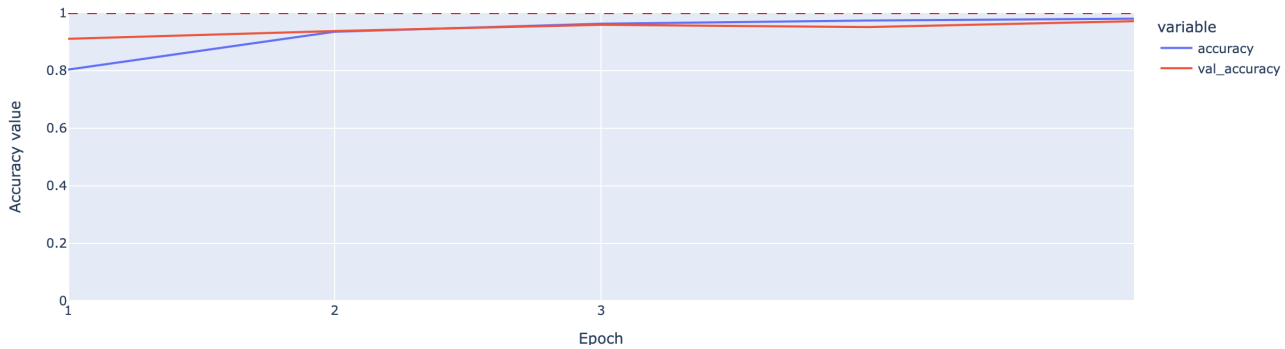

Figure 13 *- Training and Validation* ***Accuracy*** *for* ***EfficientNetB0***

For the **ResNet model** we can compare the training and validation accuracy on **Figure 14**, as well as the training and validation loss on **Figure 15**. Like the previous model, the training and validation measurements are close to each other, the loss values are tending to 0 and the accuracy ones are tending to 1 too. A remark that we can make is that at the second epoch the model was overfitting to the training data, as indicated by the gap between the training accuracy (0.91) and validation accuracy (0.63), as well as the lower training loss (0.23) compared to the validation loss (1.11). This suggests that the model was performing well on the training dataset but struggled to generalize to the validation data. However, after this epoch, the gap began to decrease a lot, indicating that the model adjusted and improved its generalization to unseen data.

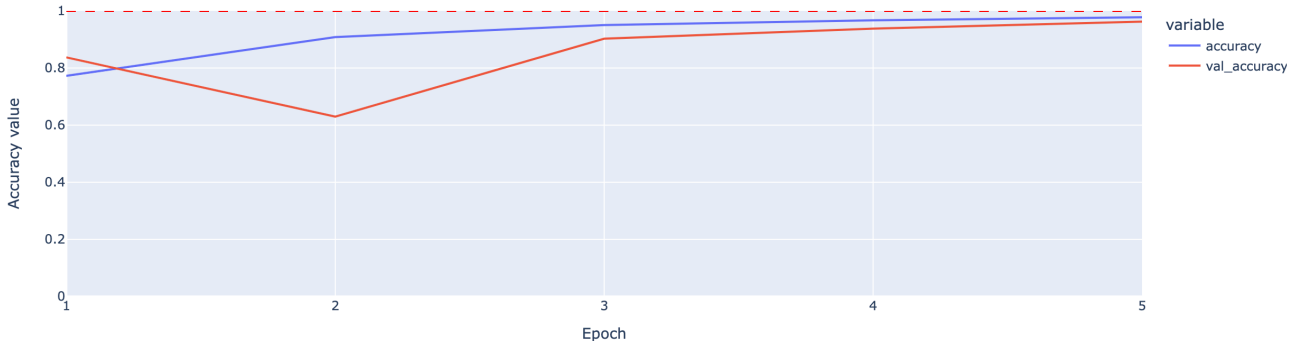

Figure 14 *- Training and Validation* ***Accuracy*** *for* ***ResNet50***

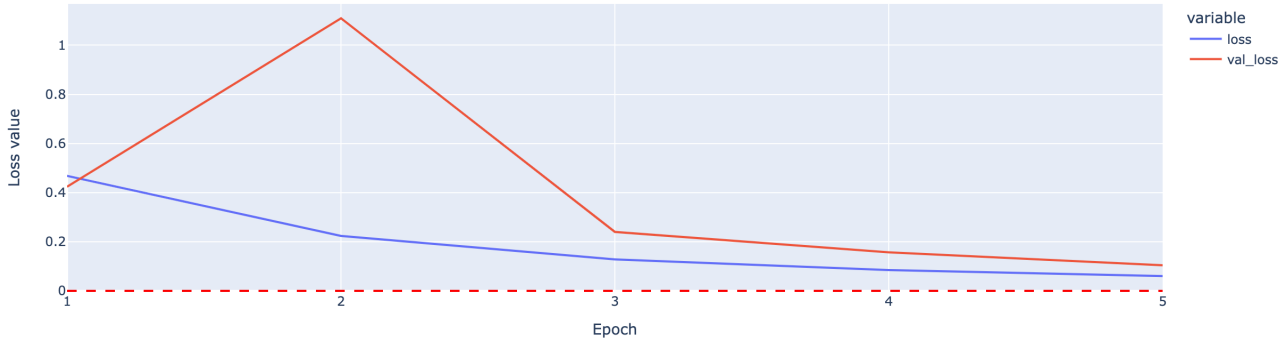

Figure 15 *- Training and Validation* ***Loss*** *for* ***ResNet50***

Finally, to avoid overfitting and save training time, the **EarlyStopping** option was used. In this way, training ends if there is no improvement in validation loss for two consecutive epochs, restoring the best model weights of the epoch with the lowest validation loss.

4.3.4 Using PyTorch: ViT

Images are preprocessed using the **PyTorch transforms** library. The images are resized to 224x224 pixels, converted to **grayscale** and normalized using a mean and standard deviation of 0.5, to prepare them for training with ViT. Then, **DataLoaders**, which efficiently load and organize data into batches, are created for each set, with a batch size of 32 for training and 1 for testing like other models.

Next, the Vision Transformer model, specifically the **ViT Base (patch16_224)** from the pyTorch library **timm**, is initialized. The model is modified by replacing the model's head (the final layer) by a **linear layer** that maps the features to a single output and a **sigmoid** activation function that outputs a value between 0 and 1. The model is also initialized without pre-trained weights. It is trained on a **GPU** using **Adam** optimizer with a small learning rate of 1e-4 that allows the model to learn more slowly and make more precise updates.

Then, training is carried out using **early stopping** over only 3 epochs because of the need to test quickly while ensuring that the model has enough time to learn the characteristics of the data set. I didn't choose 5 epochs like other models because it was far too long. If we compare the training and validation measurements on **Figure 16** and **Figure 17** we can see that, like the previous model, they are close to each other. But the loss values are tending to 0,4 and the accuracy ones are tending to 0,80. These values are not excellent probably because it should take more epochs to improve the model. But compared with previous models, these values were better in epoch 3 than on this model.

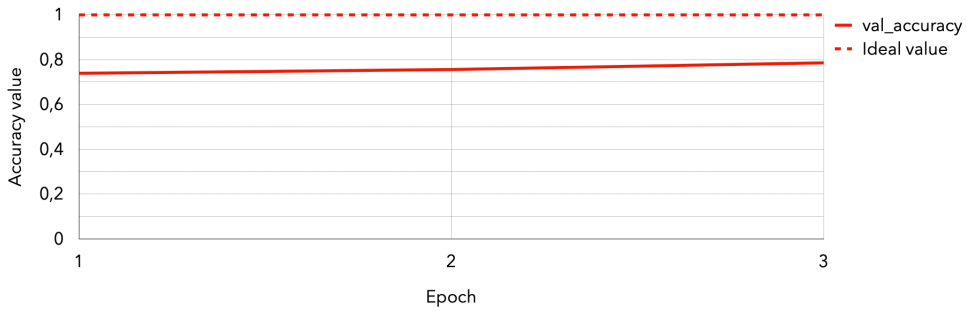

Figure 16 *- Training and Validation* ***Accuracy*** *for* ***ViT***

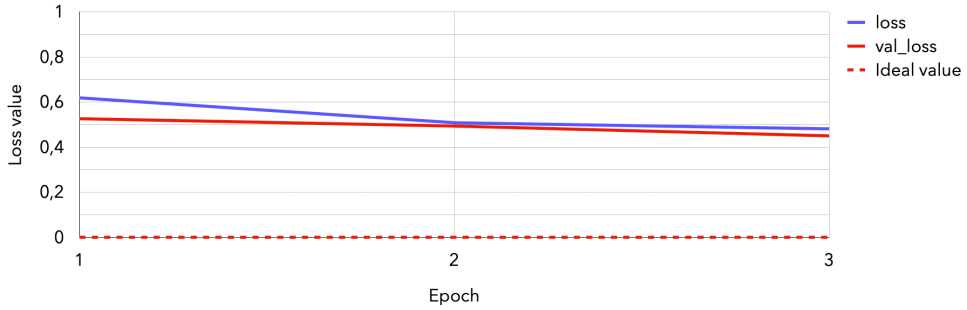

Figure 17 *- Training and Validation* ***Loss*** *for* ***ViT***

4.4 Evaluation Metrics

To help me define the metrics to use for a classification task I used the resources in reference [7]. I therefore chose to evaluate my model by analyzing the results of the following metrics:

4.4.1 Accuracy

This metric corresponds to the ratio of correctly classified samples to the total number of samples.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **True Positives (TP)**: Correctly identified positive cases.
- **True Negatives (TN)**: Correctly predicted negative cases.
- **False Positives (FP)**: Negative cases incorrectly predicted as positive.
- **False Negatives (FN)**: Positive cases incorrectly predicted as negative.

The aim is to obtain an accuracy of 100%, which means that all the model's predictions are correct, while a low accuracy indicates that the model has made many errors.

4.4.2 Confusion Matrix

In binary classification, the confusion matrix is a 2×2 grid summarizing the model's performance. Diagonal elements represent correct predictions, while off-diagonal elements show misclassifications. Maximizing diagonal values and minimizing off-diagonal ones reflects a well-performing model. This tool helps identify systematic errors and evaluate overall classification accuracy.

4.4.3 Recall

This metric corresponds to the **proportion of true positive** instances correctly identified by the model:

$$\text{Recall} = \frac{TP}{TP + FN}$$

The maximum recall is 1, which means that the model has correctly identified all positive cases.

4.4.4 Precision

Precision measures the **accuracy of positive predictions** made by a model. This is calculated by dividing the number of true positive predictions by the total number of positive predictions, which includes both true and false positives:

$$\text{Precision} = \frac{TP}{TP + FP}$$

The maximum Precision is also 1, which means that the model has correctly identified all the **positive cases among those it predicted as positive**. However, achieving perfect accuracy may reduce Recall, as the model may fail to identify all positive cases, thus increasing False Negatives.

4.4.5 F1-score

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-score combines the two metrics Precision and Recall into one, giving a balance between the model's ability to identify positive cases (Recall) and avoid false alarms (Precision). A F1-score close to 1 indicates a good balance between the two, showing good model performance.

4.4.6 ROC-AUC Curve

The Receiver Operating Characteristic (**ROC**) curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). The Area Under the Curve (**AUC**) quantifies the model's ability to distinguish classes. An AUC of 1 indicates perfect classification, while 0.5 suggests random predictions. The closer the ROC curve is to the top-left corner (high TPR and low FPR), the better the model.

4.4.7 Precision-Recall Curve and AP score

The Precision-Recall Curve plots **Precision** against **Recall**. The curve of a well-performing model is closer to the top-right corner, where both **Precision and Recall are maximized**. The average precision (**AP**) summarizes the Precision-Recall curve into a single value representing the average of all predictions. The objective here is to have an AP score of 1.

5. RESULTS

5.1 Model Performance

Now that we have a better understanding of each metric, we can use them to see the performance of each model on the test dataset.

5.1.1 EfficientNetB0

The model performs well, achieving 97,17% **accuracy**. The values on the diagonal of the **confusion matrix**, shown in **Figure 18**, are maximized. It correctly classified **9666 fake images** and **9768 real images** out of 10000 each. We can assume that the model performs slightly better in identifying real images. This is explained by the **Table 2** where the slightly higher recall for real images (0.98) compared to fake ones (0.97), indicate that the model is marginally more effective at identifying real images without missing them. But the precision for real images (0.97) is slightly lower than for fake images (0.98), showing that while the model is accurate for both classes, it is slightly better at avoiding false positives for the fake class. This difference suggests that the model has a minor bias towards correctly classifying real images.

Then, the ROC curve is very close to the top-left corner and the AUC value is nearly 1, indicating an almost perfect classification performance. This demonstrates that the model achieves a high true positive rate while maintaining a low false positive rate, highlighting its ability to separate the classes effectively.

Additionally, the Precision-Recall curve is close to the top-right corner, where both precision and recall are maximized. The AP value, being very close to 1, further confirms that the model maintains consistently high precision and recall across various thresholds.

	precision	***recall***	***f1-score***
Fake	0.98	0.97	0.97
Real	0.97	0.98	0.97

Table 2 *- Precision, Recall and F1-Score EfficientNetB0*

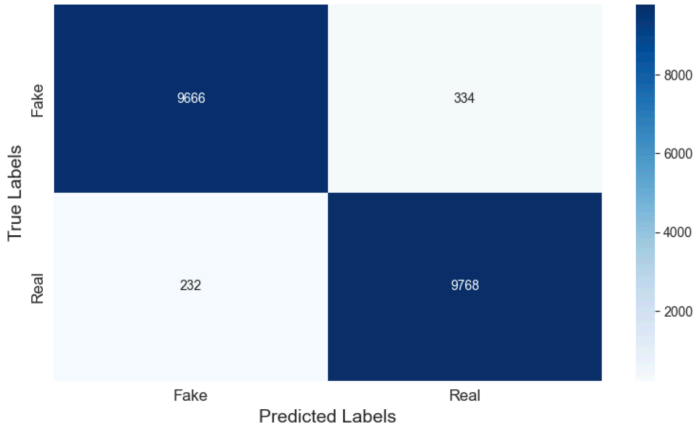

Figure 18 *- Confusion Matrix EfficientNetB0*

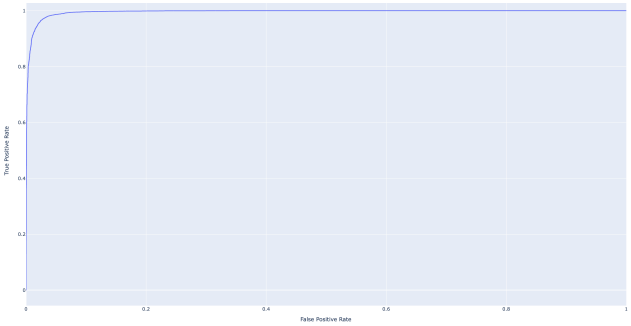

Figure 19 *- ROC-AUC Curve with* ***AUC = 0.9956***

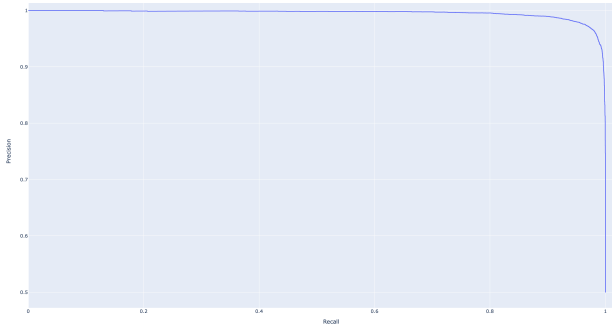

Figure 20 *– Precision-Recall Curve with* ***AP = 0.9951***

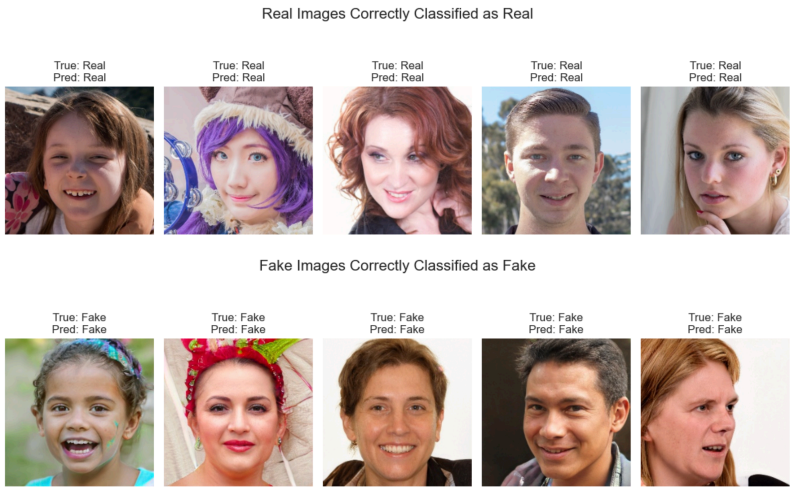

Figure 21 *- Correctly Classified Images*

1.1.1 ResNet50

Accuracy is 96,41% and the values on the diagonal of the **confusion matrix** shown in **Figure 22** are maximized. This model correctly identified **9480 fake images** and **9802 real images**. Here, we can also hypothesize that the model performs slightly better at identifying real images. If an image is not well classified, it's more likely to corresponds to fake images classified as real. This is explained by the **Table 3** where the higher recall for real images (0.98) compared to fake images (0.95). Similarly, the precision for real images (0.95) is lower than for fake images (0.98). The conclusions are the same than the previous model but it's a bit more vicible here.

Then, like EfficientNetB0, the ROC curve (**Figure 23)** is very close to the top-left corner, the AUC value is nearly 1, the Precision-Recall curve (**Figure 24)** is close to the top-right corner and the AP value is very close to 1.

	precision	***recall***	***f1-score***
Fake	0.98	0.95	0.96
Real	0.95	0.98	0.96

Table 3 - *Precision, Recall and F1-Score ResNet50*

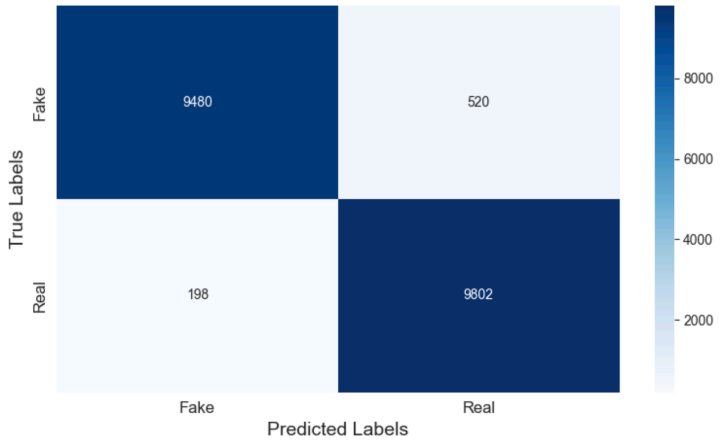

Figure 22 *- Confusion Matrix ResNet50*

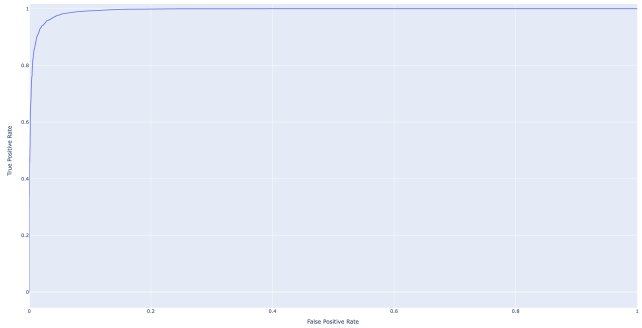

Figure 23 *- ROC Curve with* ***AUC = 0.9944***

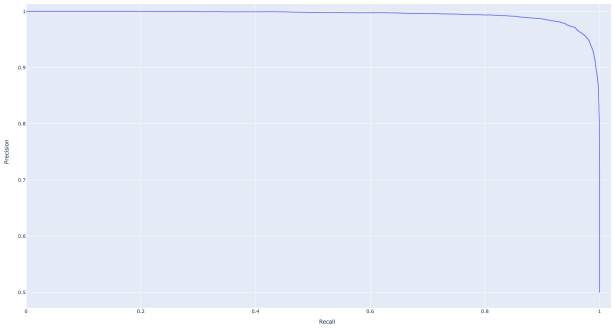

Figure 24 *- Precision-Recall Curve with* ***AP = 0.9939***

Figure 25 *- Correctly Classified Images*

1.1.2 ViT

The model performs well in this case. **Accuracy** is 78,65% and the model correctly identified **6838 fake images** and **8891 real images,** but 3162 fake images are classified as real and 1109 real images are classified as fake (**Figure 26**). These last two measurements are high and show that the model identifies real images better than fake ones and he has more difficulty classifying synthetic faces. This is explained by

	precision	***recall***	***f1-score***
Fake	0.86	0.68	0.76
Real	0.74	0.89	0.81

Table 4 where a lower recall (0.68) for fake images indicates that many were not correctly identified, reflecting difficulty in distinguishing fake faces.

The ROC curve (**Figure 27)** is reasonably close to the upper left-hand corner, with an AUC value of 0.8717, indicating a reasonable ability to distinguish real from synthetic images.

However, the precision-recall curve (**Figure 28)** does not come close to the ideal upper right-hand corner, reflecting the model's difficulties in achieving a high balance between precision and recall. The AP value of 0.7611 further confirms the model's difficulty in maintaining consistently high performance, particularly for the synthetic class, for which it has endeavored to simultaneously minimize false positives and false negatives. Although not random, the classification results suggest significant margin for improvement in managing the complexities of this task.

	precision	***recall***	***f1-score***
Fake	0.86	0.68	0.76
Real	0.74	0.89	0.81

Table 4 *- Precision, Recall and F1-Score ViT*

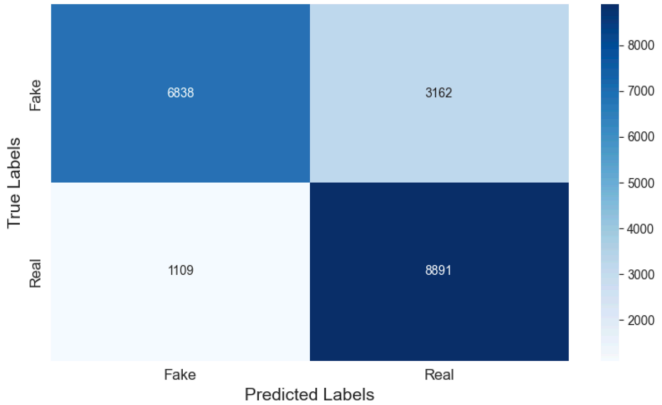

Figure 26 *- Confusion Matrix ViT*

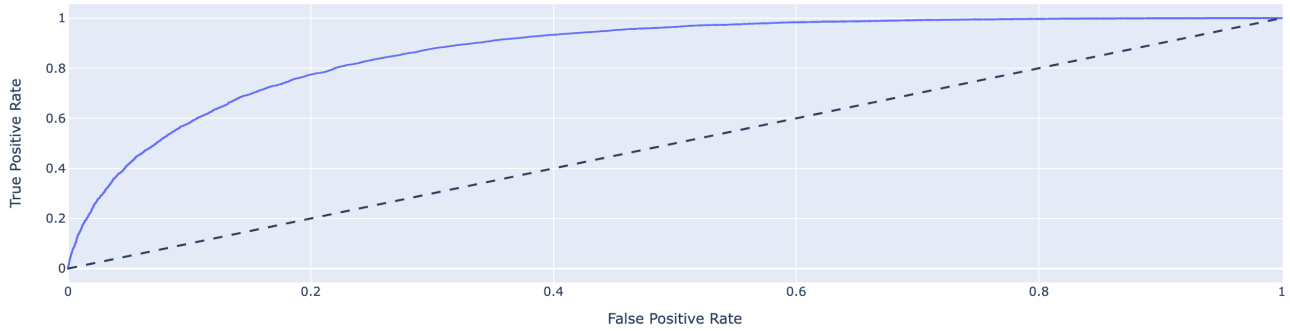

Figure 27 *- ROC Curve with* ***AUC = 0.8717***

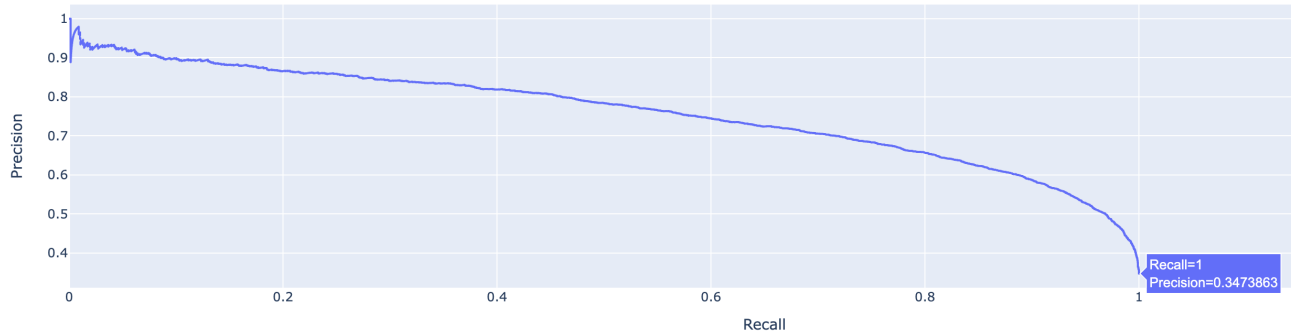

Figure 28 *- Precision-Recall Curve with* ***AP = 0.7611***

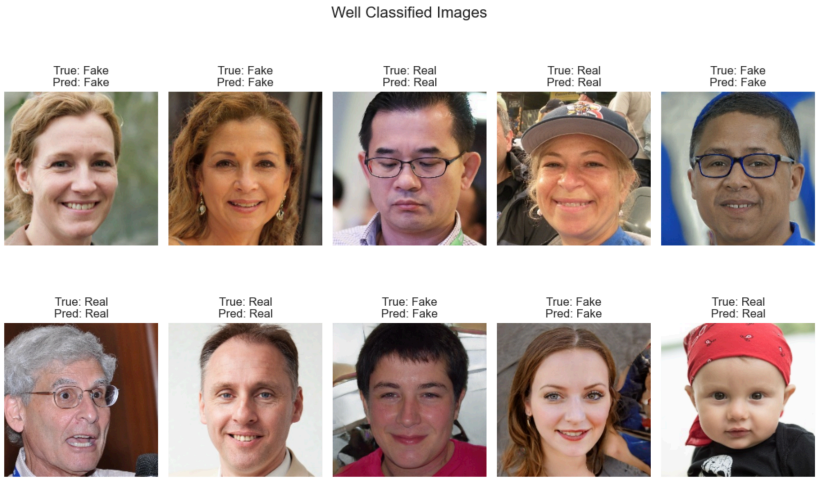

Figure 29 *- Correctly Classified Images*

1.2 Error Cases

In this section, we compare misclassified images with the correctly classified images shown **Figure 21**, **Figure 25** and **Figure 29**. Real images misclassified as Fake often feature lighting, poses or artifacts that the model interprets as synthetic. These can include unusual hair colors, as shown in **Figure 33** for EfficientNet. For ResNet (**Figure 34**), misclassified real images often feature dark lighting, uneven lighting conditions, shadows on faces or uniform backgrounds. Similarly, ViT (**Figure 35**) has difficulty with real images whose blurred backgrounds resemble synthetic features, such as those taken with a shallow depth of field.

Face orientation also has an influence on classification errors. Slightly turned or tilted faces often lead to errors, with fake images being classified as real. This problem is probably due to a bias in the training dataset, which contains more images of frontal faces.

To remedy these errors, the models should be trained on a dataset featuring a greater variety of angles, light and accessories.

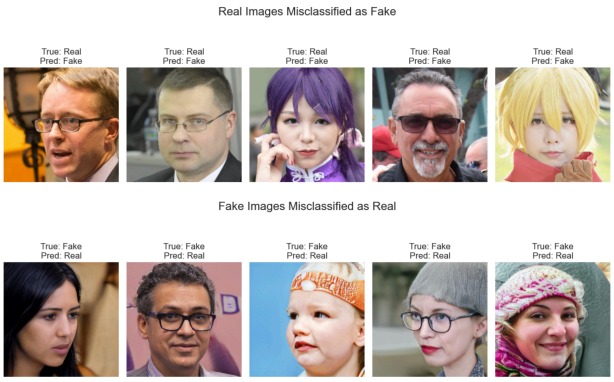

Figure 30 *- Misclassified Images* ***EfficientNet***

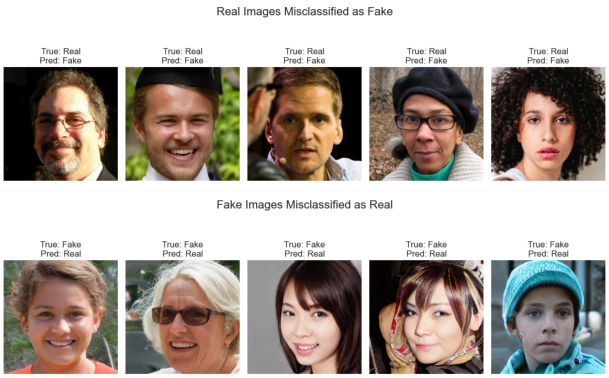

Figure 31 *- Misclassified Images* ***ResNet***

Figure 32 *- Misclassified Images* ***ViT***

1.3 Real-life test

I tried to test each model on personal images as well as on new images generated by the AI. I obtained the same results for each model. We can see the confusion matrix and the predicted and actual values for each image in **Figure 33** and **Figure 34**. All my real images were classified as such, but there are some false image classification errors. I chose to generate three images from the 'This person does not exist' site, which generates very realistic images using StyleGAN2. Almost the same template was used to train the models. And I'm using three other fake images, generated by another model, that look fake. This explains why it doesn't work for these last three images.

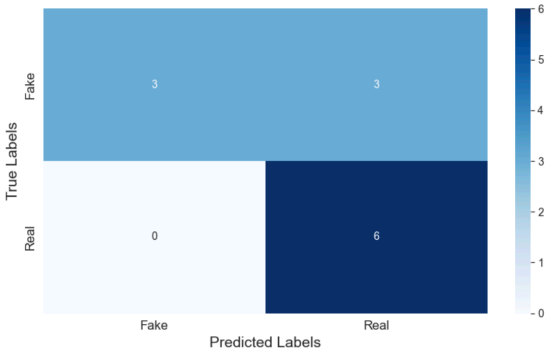

Figure 33 - *Confusion Matrix*

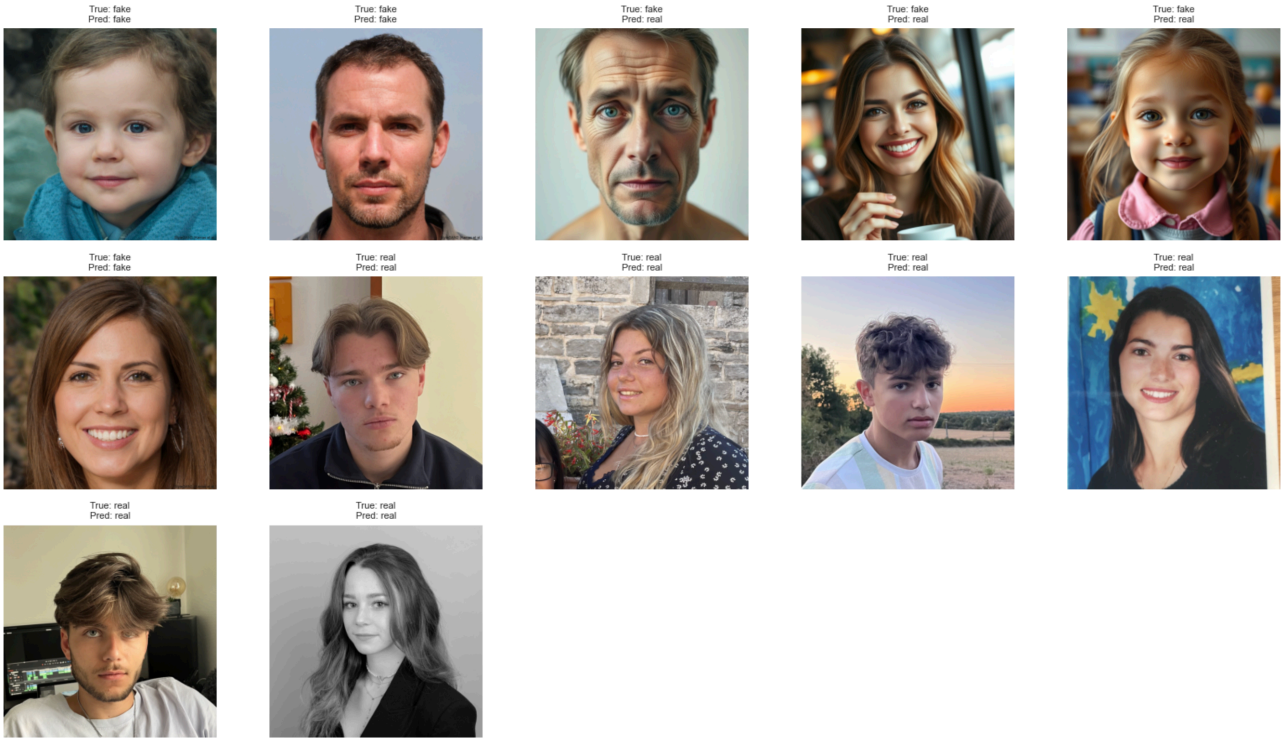

Figure 34 *- Prediction vs Real value*

1.4 Comparative Analysis

EfficientNetB0 demonstrated the best overall performance with 97.17% accuracy and other very excellent measurements shown in **Figure 35** and **Figure 36**. Its architecture requires fewer parameters and operations, so training took only 1h45 per epoch. This makes EfficientNetB0 particularly well-suited for my case where computational resources were limited.

ResNet50's robustness and generalizability were then demonstrated in its performance, with 96.41% accuracy and good results for all other measurements. The training was more computationally expensive, requiring 4h20 per epoch, mainly due to the higher number of parameters and FLOPS.

ViT is significantly less accurate than EfficientNetB0 and ResNet50 (78.65%) and requires far more computing resources. Its reliance on large-scale pre-training and its difficulty in generalizing from smaller datasets contributed to its poorer results. In addition, training was slow, taking 5.5 hours per epoch, even with only three epochs performed in this experiment. With further training, ViT could potentially outperform CNN-based models.

So, EfficientNetB0 emerged as the best-performing model for this dataset, offering a strong balance between speed, efficiency, and accuracy. ResNet50 with a higher computational cost. ViT, while promising, was constrained by limited training epochs and its high computational demands, underperforming on this dataset compared to the CNN-based models. Moreover, each one did a better job of distinguishing between real and faked images. And in term of implementation, it was as easy to implement them thanks to the two Framework.

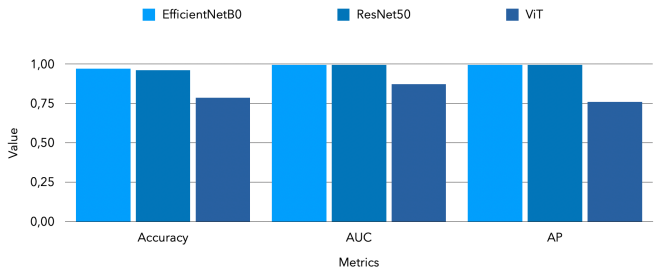

Figure 35 *– Accuracy, AUC and AP Comparison*

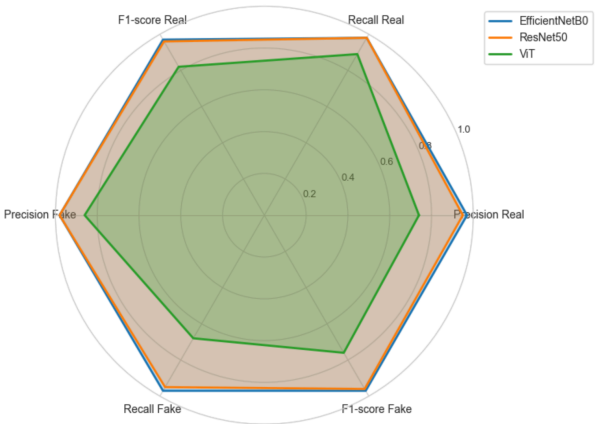

Figure 36 *– Radar plot of model performance*

2. DISCUSSION

A few limitations and weaknesses emerged. First, there's a dependence on training data specific to the StyleGan model. The model should be trained with fake images generated by many different AI models in order to recognize more details specific to AI-generated images. Even though images 3, 4 and 5 in **Figure 34** are not real, the model should still be able to tell that they are not real. In addition, we encounter a generalization problem, as images that the model doesn't know about are generally classified as real.

To go further, we could explore new architectures, such as hybridization between CNN and ViT, which combine the advantages of both approaches: the extraction of local features by CNNs and the modeling of global relationships by Transformers. Furthermore, it would be relevant to use recent images, as those used here were generated 5 years ago, and more recent models that have been optimized.

3. CONCLUSION

This study highlights the use of CNN models and transformers for synthetic face detection in AI-generated media. EfficientNetB0 proved the most effective, balancing efficiency and accuracy, while ResNet50 required more resources. ViT, while promising, has struggled to become widespread due to limited training and high resource demands. The study highlights the need for diverse training datasets from multiple AI systems and suggests exploring hybrid models and recent AI-generated images to improve detection. These ideas help to strengthen safeguards against media manipulation in the areas of privacy, misinformation and digital security.

4. REFERENCES

[1] Tero Karras, Samuli Laine and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv:1812.04948*

[2] Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946*

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929*

[5] *https://keras.io*

[6] *https://pytorch.org*

[7] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A. Chavez-Urbiola and Julio A. Romero-Gonzalez. 2024. Loss Functions and Metrics in Deep Learning. *arXiv:2307.02694*

[8] Diederik P. Kingma, Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*