BSP3: Detecting AI-generated Art

Friday 16th February, 2024 - 09:06

Angelica Rings
University of Luxembourg
Email: angelica.rings.001@student.uni.lu

This report has been produced under the supervision of:
Bereket A. Yilma
University of Luxembourg
Email: bereket.yilma@uni.lu

Abstract

This bachelor semester project aims to explore the topic of image classification and the creation of a software that serves as an AI art detector, which distinguishes between AI-generated and human-made artworks. The scientific deliverable of this project is centred around the scientific question "How can advancements in image recognition technologies, such as convolutional neural networks, be leveraged to detect AI-generated art?". The report focuses on the theoretical aspects, including the fundamentals of Convolutional Neural Networks (CNNs), training a CNN and binary classification. On the technical side, we build a web application with a basic GUI. A dataset with AI-generated and human-created artworks is collected, and then a CNN-based architecture will be employed to build a classification model. This model is designed to analyze input images and provide a prediction of whether or not the image has been ai-generated or not.

1. Introduction

Recently, AI advancements have enabled machines to create artworks autonomously. Thus, differentiating between human-made and AI-generated art has become increasingly difficult due to the improved AI generation models. This has created a significant problem. One of them is the ease with which realistic images can be fabricated to propagate misinformation and fake propaganda. As well as exploiting it to scam people and commit crimes. The art scene also suffers from this, since people rely more and more on AI Generators to create the art they envision or steal another persons art as a sample.

A solution for that problem is creating an application that can predict whether or not the input image has been AI-generated or is an artwork created by a human. In this report we explore the process of creating a CNN and develop a webapp additionally where you uplaod the image.

2. Project description

2.1. Domains

2.1.1. Scientific . The scientific domains of this project is Artificial Intelligence, specifically Computer Vision. This part goes over the theoretical aspects of image classification.

2.1.2. Technical. The domain of the technical part of this project is Computer Vision.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The scientific question of this project is **"How can advancements in image recognition technologies, such as Convolutional Neural Networks (CNNs), be leveraged to detect AI-generated art?"** This deliverable focuses on the machine learning technique used in this project, examining the functionality and performance in context of classifying different image datasets. It provides insights into the mechanism.

2.2.2. Technical deliverables. The aim of this project is to develop a web application centered on image classification. This involves creating a foundational website that enables users to upload images for classification, distinguishing between those generated by AI and those created by humans. This section outlines the process of creating the dataset, convolutional, neural network and the web application.

3. Pre-requisites

3.1. Scientific pre-requisites

To begin with, it is essential to have a good understanding of various machine learning techniques and the concepts of computer vision, as well as understanding how AI art is being generated, and what distinguishes it from human made art.

3.2. Technical pre-requisites

Coding knowledge in python and a good understanding of how neural networks function, to later implement it. As well as knowledge in web development to create an interface where the user can upload an image.

4. A Scientific Deliverable

4.1. Requirements

The goal of this project is to detect AI-generated art.

Let: X be the set of input images.
Y be the set of possible labels, where:
Y = 0 represents AI-generated art.
Y = 1 represents human-created art.
D is the dataset with pairs (x, y),
where $x \in X is an image and y \in Y is its label$.

Now, we aim to find a classification model (hypothesis) H that maps an input image x to a predicted label y:

$$H : X \to Y \tag{1}$$

The classification model H could be represented by a function such as a neural network, which takes the image as input and produces an output in the set Y.

Specifically we define this binary classification problem mathematically as follows:

1) **Training Data**: Given a training dataset D_{train} consisting of pairs (x, y), where x is an image and y is the corresponding label (0 for AI-generated, 1 for human-created):

$$D_{train} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{2}$$

2) **Hypothesis Space**: We define the hypothesis space H as a set of possible classification functions:

$$H = \{h(x; \theta) \mid \theta is a set of model parameters\} \tag{3}$$

Here, $h(x; \theta)$ is a function parameterized by θ that maps input images to predicted labels.

3) **Objective Function**: We define a loss function, typically a binary cross-entropy loss, that measures the dissimilarity between the predicted labels and the true labels in the training data:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; \theta), y_i) \tag{4}$$

Where L is the loss function, and θ represents the model parameters.

4) **Optimization**: Find the optimal model parameters $\theta^* = argmin J(\theta)$ by minimizing the loss function over the training data.

Once we have trained our model and found the optimal parameters θ^*, we can use the classification model $h(x; \theta^*)$ to predict whether a given image x is AI-generated or human-created by evaluating it with the model:

$$y_{pred} = h(x; \theta^*) \tag{5}$$

The predicted label y_{pred} will be either 0 (AI-generated) or 1 (human-created), effectively solving the binary classification problem. Thus, we formulate the following key research question to investigate in this project:

RQ: "How can advancements in image recognition technologies, such as Convolutional Neural Networks (CNNs), be leveraged to detect AI-generated art?"

To answer this key research question we explored state-of-the-art CNN-based approaches for image recognition and employed them to solve the problem formulated above.

4.2. Design

The production section of this project discusses the following topics:

- Introduction to Image Classification
- Related Works
- Convolutional Neural Network
- Experiments

Initially, a comprehensive introduction will be provided on the topic of image classification to aid the comprehension of the following sections. The following section discusses various studies and projects related to the topic of detecting AI generated images with machine learning techniques. Those works give context on the project using existing scientifical research and works. After that, the process of image preprocessing and classification will be covered and explained. The methodology and outcomes of experiments conducted in the project are detailed, offering insights into the practical aspects of implementing image classification techniques.

4.3. Production

Image Classification is a machine learning task and part of the computer vision field. An image classifier model takes an image as input and assign the image to one of the classes of the dataset it was trained with. For example looking at a model called ResNET50, that has been trained with the dataset ImageNET would have 1000 classes. The classifying process relies heavily on the deep learning background to extract the necessary information, patterns and features from the image to give a prediction. In this section the process of image classification will be explained. This encompasses image preprocessing, feature extraction utilizing a convolutional neural network, dataset division, model training, and subsequent model testing. [5]

4.3.1. Related Works. In the recent years, various methods for categorizing AI-generated imsages and real images have been proposed. In this section, we will delve into several approaches.

The work of Jordan J Bird et al. [3] proposed the utilization of CNNs with binary classification. They created a dataset called CIFAKE, which consists of 60,000 images generated using latent diffusion, alongside another 60,000 real images,

that were taken from the CIFAR-10 dataset. The dataset was used to train the model. Additionally, they incorporated a feature called XAI (explainable AI), which is used to interpret the results of the image classification. To achieve this, they employed Grad-CAMs (Gradient Class Activation Maps) to extract the most important features of the image. This is particularly useful because the most crucial aspects of an image typically highlight the inaccuracies present in AI-generated images. This approach yielded an accuracy of 92,98%. The highest accuracy was achieved by employing two output layers with 128 filters and a loss of 0.221.

A similar approach to explain the decision has been shown in the work of Samah S. Baraheem et al. [2], they have compared and implemented different kinds of Class Activation Maps (CAM) techniques, such as AblationCAM, LayerCAM, Faster ScoreCAM and Grad-CAM as in the previous paper. They created their own dataset called RSI, which consists of several other datasets. This dataset is used to fine-tune a classifier that has been pre-trained with the ImageNet dataset. The CAMs were used to find specific artifacts and visualize it. An explanation technique known as Local Interpretable Model-agnostic Explanations (LIME) has been integrated, and highlights regions with super-pixels extracted from the CAMs. This technique can provide insights into which regions of the image contributed to the model's prediction. Based on the test outcomes, the model EfficientNetB4 exhibited the highest efficiency, achieving an accuracy of 100%. The study's findings suggests that GAN-generated images exhibit specific imperfections, distortions and artifacts, which well-trained models can use to identify whether an image has been generated by AI.

On the other hand, Luca Guarnera et al. [6] suggested to use the image classification model ResNET-34. This model has 14 output classes, of which the class with the real images is being removed for enhanced accuracy. The multilevel deepfake detection system consists of 3 levels, the first level determines whether the image was AI-generated or if it is a real image, the second level distinguishes between DM (Diffusion Model) and GAN (Generative Adversarial Networks) and finally, the third level identifies the engine used to generate the image. The overall accuracy of this project was 97% for each task, with the first level classification achieving an accuracy of 94.85

Another different approach to this problem has been presented by Peter Lorenz et al. [9]. In their study, they utilized the Multi Local Intrinsic Dimensionality (multiLID) to detect AI-generated images. While most existing approaches and studies mainly focused on GAN-generated images, this method specifically targets DM-generated images. In their method, they used an untrained ResNET-18 model to extract the low-dimensional features of the images. This step is crucial because calculating the LID of a high-dimensional image is inefficient in this case. In the following step, they calculate the multiLID and pass the images through a trained classifier to determine, based on the LID value, whether the image has been generated by AI or not. An expected LID value is taken as a reference, thus if the calculated LID of the image does not match, it indicates irregularities in the image, potentially stemming from AI generation. After conducting tests with various models and the LSUN-Bedroom dataset, it was observed that the pre-trained models employed performed less effectively than the custom binary CNN they had developed. Calculating the multiLID will enhance the performance and accuracy of CNNs. However, the downside of this approach so far is that it does not perform very well with new data.

Lastly, in the paper of Riccardo Corvi et al. [4], they have compared different AI image detectors. Namely: Spec, PatchForensics, Wang2020, Resnet50 and Grag2021. Those models are trained using only the dataset ProGAN, which contains GAN-generated images. However based on the test results, this has limited the capability and accuracy of the detectors. They performed slightly worse in context of DM-generated images or new data, thus it is important to train the detectors with images of different generation engines. Out of all models, the model Spec has had the highest score. For the test with the uncompressed image it has achieved an accuracy of 70.5/75.2, which is 93,75% and for the test with the compressed images it has had a 100% accuracy. In conclusion, the model Spec works the best for DM-generated images.

4.3.2. Convolutional Neural Networks. This subsection will explain the process of image classification and explore various elements comprising convolutional neural networks. This encompasses image preprocessing, feature extraction utilizing a convolutional neural network, dataset division, model training, and subsequent model testing.

Steps of implementation:

- Preprocessing data
- Design and define the CNN
- Choose a loss and optimizer function
- Train CNN model
- Evaluate CNN model

Preparing the data. Preparing the data for the image classification task is a very crucial step. The input data has to be preprocessed. The first step is data collection, then the dataset has to be loaded and labeled. Then the image will transformed or enhanced. Under this falls normalizing the image, converting it to greyscale, and resizing it. Afterwards, the image is transformed into a Tensor. Normalization enhances the contrast of the pixels, meaning that the difference between the dark and light areas will be clearer. For images with lower contrast, this can improve the visibility of certain features. [5] In Figure 1 we can see a significant difference between the original image from the dataset and the same image after being normalized.

If the image classification task does not require colors, then it is converted to grayscale to reduce the complexity. Grayscale images only show the differences in brightness. There are 256 different shades of gray, ranging between black and white. This means since there is only one channel passed through, it takes less information. [5] In Figure 2 the original image and

Fig. 1: Original and Normalized Image

the grayscale image don't show much of a difference since the saturation and contrast of the original image are low.

Fig. 2: Original and Grayscale Image

The last transformation applied is resizing the image. To resize the image the original image will be scaled down in the same ratio, and if necessary added padding so that every image has the same dimensions. Generally, the size does not align with the original pixels which creates the issue of the brightness and saturation not matching, to solve that an interpolation method is being used to estimate and adapt the color or intensity of pixels at the new position. Resizing is important to ensure the model learns the features consistently. As well as accelerating the training process. In Figure 3 the image has gone through all transformations and has been resized from 608x1000 to 50x50 pixels. In this example, the bicubic interpolation mode has been used. It takes samples from the larger neighbouring area of the pixels to estimate the new values.

Fig. 3: Original and Fully Transformed Image

Figure 4 shows an image from the dataset on the left and on the right the same image after going through the transformations. The values in the grid show the tensor values per pixel.

After preprocessing the image and saving the data, the dataset has to be split into the train and test datasets. The most common ratio for training and validation is 80/20.

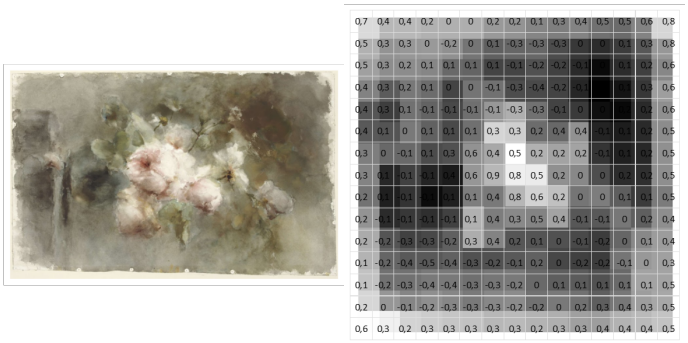

Fig. 4: Original Image and Transformed Tensor (25x25 pixels)

Model Architecture. A Convolutional Neural Network consists of several layers through which the input images are passed. The main types of layers are the input layers, convolution layers, pooling layers, and fully connected or dense layers [5]. Those layers consist of neurons and have different operations.

The base of a CNN are **convolutional layers**. It comprises a kernel (also known as a filter) that can be adapted with the padding and stride parameters. This type of layer takes an input matrix, transforms it, and passes the output to the next layer. It is specifically designed to detect patterns within the input data. Thus images, comprising various patterns, such as shapes, edges, objects, textures, and colours are systematically analyzed by the convolutional layer. The filter of the layer traverses each set of pixels of the input until every pixel has been processed. Those filters embedded within the layer, detect specific patterns. Starting with simple features like edges or shapes, the filter progressively evolves into more complex patterns as the network delves deeper. The resulting matrix is called a feature map. From a mathematical perspective, the formula used to extract the features is:

$$W = (WF + 2 * P)/S + 1$$

with W being the image tensor, F being the kernel matrix, P being the padding and S being the stride value. [1] **Stride** is the offset or step the filter makes between each traversal. **Padding** adds pixels around the area the filter is processing. The stride and padding values have an impact on the dimensions of the input or output vector. Between each convolutional layer, the Rectified Linear Unit (ReLU) activation function gets called. If the value of the input is positive, it will return the same value unchanged, however, if the value is negative it will return 0. After this, the output gets passed to another layer. The function itself is defined as [5]

$$f(x) = max(x, 0)$$

Afterwards, the feature map is passed to the next layer in the CNN, which is usually a pooling layer. There exist two types of **pooling layers**: Max Pool and Average Pool. Max Pool retains the most significant elements of the feature map, while Average Pool computes the average of each region of the feature map. The primary purpose of pooling is to reduce

the dimensions of the input data, effectively shrinking the image. This reduction minimizes the number of parameters and computations required in subsequent layers of the neural network. Max Pooling vs. Average Pooling Formulas: [5] [8]

$$MaxPooling(x_{i,j}) = \max_{(p,q)\in R_{i,j}} x_{p,q}$$

$$AveragePooling(x_{i,j}) = \frac{1}{|R_{i,j}|} \sum_{(p,q)\in R_{i,j}} x_{p,q}$$

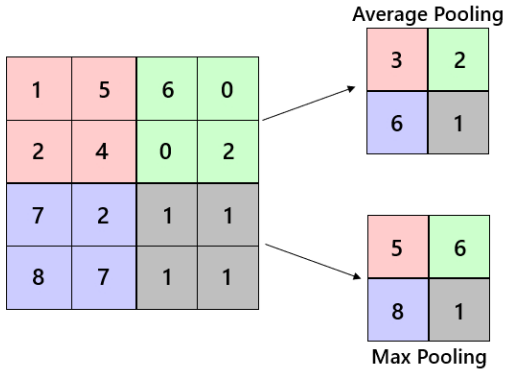

Fig. 5: Max Pooling vs Average Pooling

The Figure 6[1] shows an example of a convolutional neural network, as it is also a deep neural network. The graphic shows the input layer, hidden layers and output layer. Each of them containing multiple neurons that are connected to other neurons.

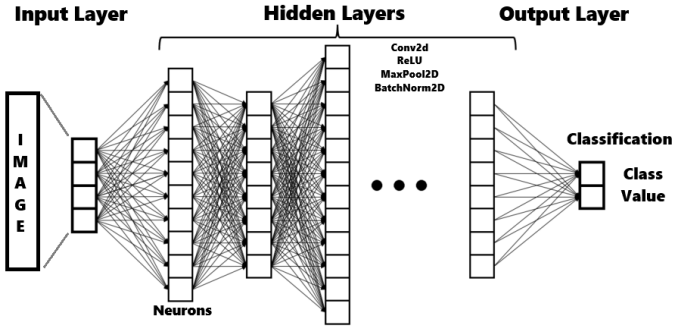

Fig. 6: Example of Deep Neural Network

The convolutional and pooling layers progressively reduce the spatial dimensions of the feature maps while simultaneously increasing the number of channels. After that the fully connnected or dense layers combine all feature maps from the previous layers and create a feature vector, which is used to classify the images. The result can be calculated through an activation function, such as Softmax or Sigmoid. Softmax returns multiple values between 0 and 1, while Sigmoid returns a single value between 0 and 1. For that reason the Sigmoid activation function is generally used for binary image classification and to give a probability score for the class it has assigned. In mathematical terms the Sigmoid activation function looks like this: [5]

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

x represents the input. $1 + e^{-x}$ is the denominator, which ensures that the output of the function is bounded between 0 and 1. Softmax on the otherhand has a different formula: [5]

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$\mathbf{x}_i$ is the input vector with the i-th value. $\sum_{j=1}^{n} e^{x_j}$ represents the sum of the exponentials of all elements in the input vector. The softmax function takes in a set of numbers and normalizes them into a probability distribution, where each number represents the likelihood of the input belonging to a particular category. This is particularly useful for multi-class problems.

Optimizer. There are several optimizer and loss functions to choose from. First we have to establish the difference between optimizers. There are adaptive optimizers, such as Adam. It possesses an automatic learning rate adjustment feature throughout the training process, achieved through the usage of gradient first and second moment estimates. It is very efficient and requires less trial-error to find an optimal learning rate. However, the issue with Adam is that it converges too quickly. Meanwhile another optimizering technique is gradient descent, such as the Stochastic Gradient Descent (SGD) optimizer. It operates by fine-tuning the model's parameters according to the errors in its predictions. The term "stochastic" indicates that it randomly selects a subset of training data for each step, enhancing speed and efficiency, especially with large datasets. [7]

4.3.3. Experiments.

Experiment 1: Hyperparameters. To find the best hyperparameters to work with, we have conducted some experiments with the CNN. This involves training the CNN using the art dataset created in the technical part and changing different parameters, such as the number of layers, number of filters, padding value, kernel size, number of epochs, the learning rate and training it with an without the optimizer, to achieve the highest accuracy.

In Table 11, you can see the test results of the AI-Art Classifier. The table consists of the number of layers, neurons the fully connected layers have, the padding used for the filter, the filter (kernel) size, the learning rate, the average loss computed after training and finally the model accuracy. To provide context on the experimental setup, the model was trained on the ArtDataset created in the technical part, comprising approximately 7000 images for training and about 1700 for validation. The dataset consists of 50x50 normalized

1. edited image; original illustration by BrunelloN: https://commons.wikimedia.org/wiki/File:Example_of_a_deep_neural_network.png#filelinks

Layers	Filters	Padding	Kernel Size	LR	Avg. Loss	Model Acc.
14	128	1	3	0,001	22,5	70,10%
14	128	1	3	0,0025	21	70,28%
14	128	1	3	0,003	21	70,51%
14	128	1	3	0,005	22	70,85%
14	8192	1	2	0,003	17	79,72%
14	16384	1	2	0,0025	16	83,35%
14	16384	1	2	0,003	17	80,70%
14	16384	1	2	0,0035	17,4	77,02%
14	16384	1	2	0,005	54	51,38%
14	32768	1	2	0,005	54	48,62%
20	512	1	3	0,001	25	60,43%
20	512	1	3	0,002	25	66,85%

TABLE 1: Experiment Results

grayscale images, that were trained in batch size of 32. The training process involved 15 epochs using the Adam optimizer. The corresponding learning rates are listed in table. As the table shows, the classifier's performance tends to degrade, as the amount of layers increases. It performs the best with a learning rate between 0,002 and 0,003 and any value bigger than 0,003 starts losing accuracy significantly.

The results for experiments 9 and 10 in the table stand out as outliers due to unexpectedly low accuracy. The test run with 32768 neurons achieved an accuracy of 48.62%, notably worse than random chance. Similarly, the other test run with 16382 neurons exhibited a comparable result with a loss of 54 and an accuracy of 51,38%. As they both have a high learning rate, suggesting the low accuracy can be attributed to the higher learning rates. Thus we can conclude that increasing the number of layer may lead to degradation in performance or diminishing returns. The number of neurons in each layer affects the model's capacity to learn complex patterns.

Experiment 2: Input Image Sizing. The next test we conduct is an image sizing experiment to determine the best dimensions for the transformations. The training was conducted on the optimal model architecture determined in the technical section, the training involved 25 epochs and a batch size of 32 images. The architecture comprised 19 layers with a total of 769 neurons, with the optimizer Adam and the learning rate was set to 0.0025.

Image Size	Avg. Loss	Model Acc.
25x25	16,1927	82,49%
32x32	13,4715	84,15%
50x50	14,9847	86,35%
64x64	14,9046	86,15%
75x75	16,7732	83,76%
100x100	18,9191	83,95%

TABLE 2: Image Sizing Experiment Results

The Table 2 presents the results of the experiment. Each row corresponds to a specific image size. The other 2 columns record the corresponding average loss computed from the training epoch, and the model accuracy. As the image size increases, the average loss tends to slightly rise, while model accuracy is generally kept within the same range. The highest accuracy is achieved with a 50x50 image size at 86.35%, suggesting that this resolution is optimal model for the given architecture and parameters. To understand the significance of image sizes Figure 7 shows an original image in comparison to the transformed images ranging from the smallest size at the top left to the image with the biggest size in the right lower corner.

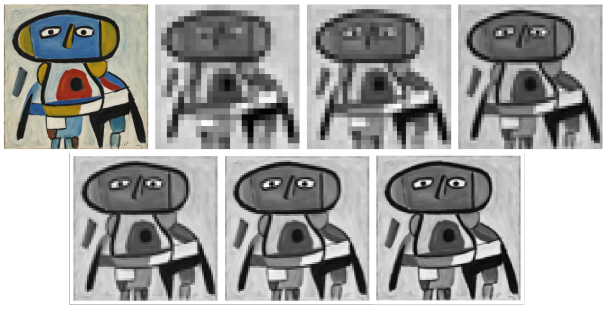

Fig. 7: Comparison between original and image sizes 25x25 to 100x100

Experiment 3: Learning Rate. This experiment aims to explore the impact of learning rates with both Adam and SGD optimizers.

Optimizer	LR	Avg. Loss	Model Acc.
Adam	0,0001	15,4064	82,26%
	0,0002	17,3165	86,14%
	0,00025	15.4411	86,35%
	0,0003	19,6508	84,27%
	0,0005	20,9407	84,17%
	0,001	23,1240	83,01%
	0,01	109.8354	50,63%
SGD	0,01	15,9250	83,59%
	0,001	5,4959	86,33%
	0,002	30,6068	84,33%
	0,005	11,6501	68,26%
	0,0001	31,4968	62,73%
	0,0005	32,4393	67,51%
	0,0009	51,3723	68,89%

TABLE 3: Learning Rate Experiment Results

First comparing all of the Adam values with each other in Table 3, there is a general trend of higher learning rates leading to higher average loss and a slightly lower model accuracy. For instance, moving from LR=0.0001 to LR=0.0002

results increase in both average loss and model accuracy. However the past that point, the accuracy and average loss starts increasing again. Thus, LR=0.0002 achieves the highest model accuracy among the Adam settings, suggesting an optimal learning rate. Next, comparing the SGD results with each other, we can observe that the highest model accuracy for SGD is 86.33%, achieved with a learning rate of 0.001. Out of all results Smaller learning rates (0.0001 and 0.0005) result in significantly higher average losses and lower model accuracies. Overall we can conclude that Adam is a better choice, considering the good performance across a range of learning rates, whereas SGD appears to be more sensitive, requiring more fine-tuning of the learning rate to achieve optimal results.

4.4. Assessment

The scientific deliverable has fulfilled the requirements that were set. An insight on recent research in the domain of image classification in context of AI-generated images has been given. As well as elaborating on the process of a CNN was presented, and experiments that determine the performance of the classifies have been conducted and analysed. A short overview of what has been discussed: Convolutional Neural Networks can be used to detect certain features in images which answers the scientific question of "How can Convolutional Neural Networks (CNNs), be leveraged to detect AI-generated art?" The conducted experiments not only show the optimal hyperparameter configurations but also emphasized the significance of careful tuning in achieving superior model performance.

5. A Technical Deliverable

5.1. Requirements

First, we have to establish the requirements for this project. We divide this section into 2 parts, as the project's objective is to develop a web application focused on image classification.

The initial phase involves the implementation of a binary image classifier, which will be exported for integration into the web application. Proceeding to the next phase, the upcoming task involves the creation of a basic GUI, that enables users to upload images and then generates a response, presenting the input image alongside the classifier's output. Simultaneously working on the backend of the web application on the integration of the classifier.

Image Classifier. The first requirement for this project is to successfully create an image classifying model and export it, but before that we have to establish the requirements for the process of creating the model. The criteria to create a good dataset are the number of images, diversity, quality of images and equal number of images for each class. Each image in the dataset has to be assigned a label. The quality and diversity of the dataset is very crucial, because the accuracy and performance rely on the data on which the classification model was trained on. So it must meet the following criteria to ensure the success. If we provide the model with a diverse set of content, it will be exposed to different kinds of styles and features. The second criterion involves determining an appropriate size for processing images, as the accuracy may be influenced by both the size of the images and the batches. The next criterion involves establishing the optimal configuration of layers and hyperparameters, such as the optimizer and learning rate, for the Convolutional Neural Network, aiming to fine-tune and enhance its overall performance. Finally the evaluation of the model should yield an accuracy of at least 80% to 90%.

Web Application. As mentioned previously, the core of the project is the image classifier. Following the completion of the classifier, the model has to be exported and implemented in the backend. Moving forward, the next step consists of developing a basic graphical interface allowing users to upload images. The model generates a response presenting the uploaded image alongside the classifier's predicted label. Thus, the web application features encompass a fundamental HTML website incorporating an intuitive upload button. Upon the user's submission of an image, the application is designed to generate a processed image with a labeled output, discerning between categories of "human-made" and "AI-generated."

5.2. Design

Initially, for the development of a custom image classification model from scratch, we utilized the CatsvsDogs dataset from Kaggle. Given the dataset was created specifically for a machine learning task, it is a common way to understand and build a basic binary image classifier. Essentailly, the task is the same only with different datasets. The procedure was documented in a Jupyter notebook, serving as a reference point for the following tasks. Specifically, we use it as a basis for the classifier and incorporate the dataset mentioned in the next section that contains artworks and the AI-generated counterparts. And the second part involves adapting the hyperparameters of the basic architecture to match the specifications of this task, since the underlying structure remains essentially identical.

The following items listed are the design choices for the CNN architecture. Some of them are based on the experiments listed in the scientific production.

Activation Functions. : The activation function chosen for this project is the sigmoid function. This choice is motivated by its suitability for binary classification tasks, where the model needs to predict whether an image belongs to one of two classes.

Image and Batch Sizes. : In the scientific experiment section, it was observed that the choice of image size and batch size significantly impacted the model's performance. The learning rate of 0.00025 was selected with the Adam optimizer due to

its adaptability. Larger image sizes led to increased processing time and lower accuracy. For instance, a comparison between 50x50 and 100x100 pixel sizes revealed a 2.5% accuracy difference. Additionally, batch sizes, specifically a choice of 32, were found to enhance results and avoid overfitting or underfitting issues.

Hyperparameter Adaptation. The optimizer chosen for the project is Adam, as opposed to the more commonly used optimizer SGD. The choice of a learning rate of 0.00025, coupled with the Adam optimizer, was based on experiments revealing that higher learning rates resulted in less optimal outcomes at specific points. Moreover, the impact of image sizes on the model accuracy was observed, the images with dimension 50x50 yielded optimal results compared to larger dimensions.

Loss Function. The choice of Mean Squared Error (MSE) loss is stems from its ability to provide a more insightful visualization of the loss compared to other algorithms, such as CE or Exponential. MSE loss highlights errors by penalizing larger deviations between predicted and true labels. In the context of binary classification tasks, its application allows for a dynamic representation of loss throughout the training process. In contrast to alternative loss functions that may persist at a diminished level following misclassification, MSE loss reflects the magnitude of errors distinctly. It highlights the areas the model is struggling in.

Evaluation Metric. : The accuracy of the image classifier is assessed using the sigmoid activation function. The evaluation metric involves a systematic calculation of accuracy, measuring the model's correctness in predicting binary outcomes. Specifically, predictions are compared against ground truth labels, and the percentage of correct predictions is determined. This metric shows the overall performance and efficiency of the image classifier on the test dataset.

5.3. Production

5.3.1. Creation of Dataset. Since the objective of this project is to detect AI-generated artworks, the initial step involves creating a dataset consisting of AI-generated work. For this approach, several different datasets have been combined, with primary based of the publicly available dataset from the Rijksmuseum. For a diverse dataset paintings and drawings featuring various art styles and motives have been chosen. Ranging from portrait paintings, scenic paintings to landscapes or animals. The images of the dataset are used as a reference to generate counterparts for each image using AI. For that we have used a Diffision Model from RunawayML[1]. RunawayAI offers a tool called "Image Variation" based on a custom adaption of Diffusion Models. Given an input image, it generates different variations of the image altered by AI, aiming to produce variations closely resembling the original artworks.

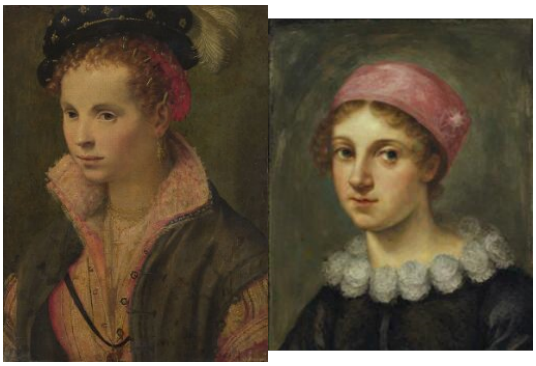

Fig. 8: Example of artwork and AI generated counterpart (human portrait)

Fig. 9: Example of artwork and AI generated counterpart (landscape painting)

The Figure 8 shows an example of a real artwork on the right, and on the left is the variation of the image. The images look very similar in terms of color scheme and painting technique, however the AI generated variation is less detailed and has a different texture that paintings don't have.

Another example of a landscape painting can be seen in Figure 9. The artworks are very similar in terms of composition and landscape. The AI generated counterpart looks less detailed and has a different style.

5.3.2. Building the Classifier. The image classifying model is developed using Python and the deep learning framework Pytorch[2]. PyTorch simplifies the process of training neural networks, making it particularly useful for this project, given our utilization and creation of a custom Convolutional Neural Networks (CNNs). The focal point of this task is a binary classifier for a standard dataset, to offer an insight and guide for the initial development of a CNN. In this example we use the well-known "CatsVSDogs"[1] dataset, which is commonly used to learn and understand binary classification. The goal of this example is to classify the images of cats and dogs, and essentially to create a prototype for the model with a high accuracy. To see the steps in more detail, please refer to the Jupyter Notebook for CatsVSDogs and AIArtDetector.

1. https://runwayml.com

2. https://pytorch.org/

1. https://www.kaggle.com/c/dogs-vs-cats/data?select=train.zip

Building an Image Classifier can be summarized in these steps of implementation:

- Preprocessing data
- Design and define the CNN
- Choose a loss and optimizer function
- Train Model
- Evaluate Model
- Save Model

Following the steps, the first thing to do before preprocessing the dataset is loading it into a dataframe from a CSV file to process the images. After this we apply the transformations to the dataset. Later, the dataset underwent partitioning, wherein approximately 20,000 images were allocated for training purposes, while the remaining 5,000 images were reserved for testing.

Preprocessing Data. The chosen transformation operations begin with resizing the images to a dimension of 150x150 pixels using the bicubic interpolation mode. Next, the images are converted to grayscale, limiting the channel to one. Then a normalization step is applied to scale pixel values, centering them around a mean of 0.5 and a standard deviation of 0.5. The transformation then convert the images into tensors. Finally, the transformation then convert the images into tensors. Figure 10 shows how the original images look in comparison to the image after the transformation.

Fig. 10: Example of Cat Image Transformation

Design and define CNN architecture. The first layer is a 2D convolutional layer (Conv2d) with 1 input channel, 32 output channels (or filters), a 3x3 kernel size, and 2x2 padding. Followed by a MaxPooling layer (MaxPool2d) with a kernel size of 2x2 and a stride of 2. A Rectified Linear Unit (ReLU) activation function is applied. Batch Normalization (BatchNorm2d) is used to normalize the input to the next layer. Batch normalization helps stabilize and accelerate the training process. This sequence repeats three more times with the following pattern: a convolutional layer, followed by max pooling, ReLU activation, and batch normalization. The number of filters progressively grows from 64 to 128, then to 256, and finally to 512. As for the fully conncted layers, the output from the last convolutional layer is flattened using Flatten. A fully connected layer (Linear) with 512 input features and 256 output features is added, followed by ReLU activation. The final layer is a linear layer with 256 input features and 1 output feature, followed by a sigmoid activation function.

Train Model. The neural network is set to training mode. Within each epoch, a nested loop iterates through the training data batches, employing the Adam optimization algorithm. For each batch, the optimizer's gradients are set to 0, and the network's output is obtained. The loss is calculated using the Mean Squared Error (MSE) loss function, with the output and ground truth tensors appropriately reshaped and converted. Backpropagation is then performed, and the optimizer updates the model parameters. During training, the running loss is accumulated, and predictions are thresholded at 0.5 to classify them as binary outcomes. The number of correct predictions is saved, contributing to the overall accuracy computation. At the end of each epoch, the accuracy and total loss are printed. The training process is completed after the specified number of epochs.

Evaluate. Accuracy metrics are computed by comparing the model's predictions to the true labels of the test set. After processing all test batches, the accuracy is determined as the ratio of correctly predicted samples in relation to the total number of samples. This specific architecture has yielded an accuracy of 90,20%. The loss is measured with MSE loss and records the error between the predicted values of the model and the actual values.

Adaptation to Art Image Classifier. The Art Image Classifier was trained with the same model created for CatsVSDOgs with slight adaptations. The images were scaled to 50x50 instead of 150x150. This choice is justified by the experiments in the scientific section, in which the 50x50 images resulted in the highest accuracy for this classifier. The amount of epochs used to train the model was 25. Each training batch consisted of 32 images. Those minimal adaptation ensured that the model learned to distinguish AI-generated and human made art effectively. We will shortly discuss the training and validation of the model.

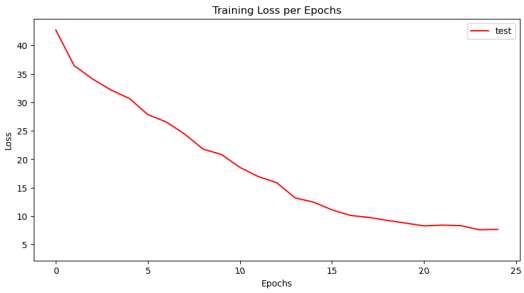

Fig. 11: Train Loss per Epoch

As seen in Figure 11, the training loss decreases consistently from epoch 1 to 15. After epoch 15 it slowly becomes more constant. This suggests that the model learns the patterns of the images throughout the initial stages of the training and

slowly becomes accustomed to the features in the later stages. It also suggests that the model has reached a point where all features have been learned, as well as the data being overfitted at the end. Generally, based on the performance of the model in the evaluation phase we can say that this has not been the case.

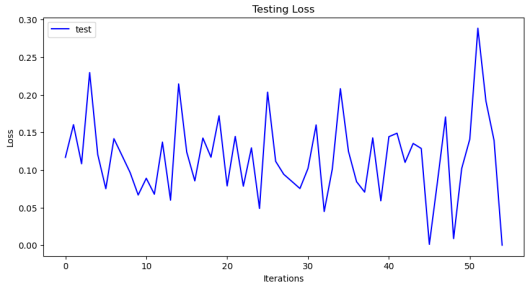

Fig. 12: Test Loss per Iteration

Finally, Figure 12 shows how the model performed during the validation phase with an accuracy of 86,35%. After validating the model the general loss is 6.384. This provides an overall measure of the model's performance on each batch of images from the test dataset. The highest loss an iteration has experienced was 30 and the lowest was close to 0, with an general loss of 6.384. The highest value of 30 indicates that there was a batch of images where the model's predictions were quite far from the true labels. Contrarily the lowest value of 0 indicates that there were batches where every image has been predicted correctly.

5.3.3. Web Application. In the initial phase, we will create a simple website. Users will be able to upload images through the interface, initiating the image classification process. The application uses Javascript for the interface elements and interactions. For the backend we will use Python, specifically the Flask library, to handle the image processing and the communication with the image classifier. The backend handles user requests and runs the image through the classification process. Finally, the application provides the result of the classification stating whether it is a real or an AI-generated art. Thus, the focus of this phase is the functionality of the image classifier, as well as having a basic interface.

The upcoming paragraphs explain the various files and components of the web application.

app.py: This module is the Flask application. The application receives an image from the upload form and saves it to the upload folder. Then it uses a pre-trained model to make predictions on the uploaded image. The results are then presented on a webpage, including the original image and the model's predictions.

predict.py: This module includes two functions for preprocessing an image and making predictions using the pre-trained neural network model. It takes an input image and file path, then preprocesses the image with the specified transformations. The transformations are the same as mentioned in the previous section for AIArtClassifier. The predict function loads pre-trained model and then passes the image through the model. The output is a probability value, which is converted into a binary prediction based on a threshold, 0.7 in this case. The script returns a dictionary containing the predicted label ("ai generated" or "human made") and the associated probability score.

net.py: This module defines the corresponding neural network class of the model.pth, as Pytorch requires this to function.

The Figures 13 and 14 show what the website looks like with and without input. The image that has been uploaded was in fact AI-generated.

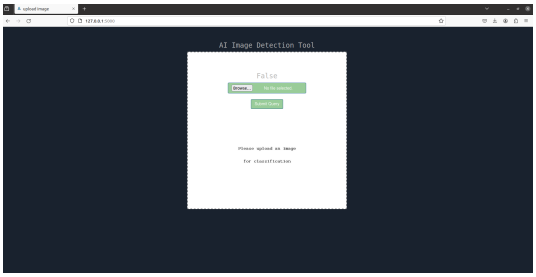

Fig. 13: Webapp without any Input

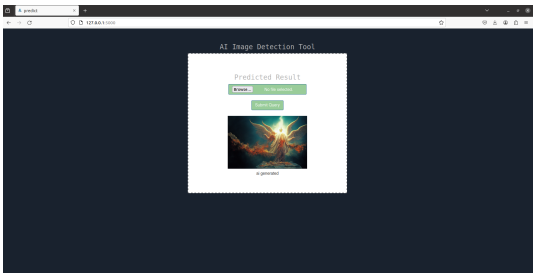

Fig. 14: Webapp with Image and Classification Result

5.4. Assessment

Most of the requirements that were set in the have been met. A custom image classifier has been created and implemented. We have been able to create a very basic web application that returns for every input either human made or AI-generated. However, the interface does not look very aesthetically pleasing and lacks an explanation for the prediction. The model yields an accuracy of 86%, which is sufficient considering it has been trained with only approximately 9000 images.

Acknowledgment

The author would like to thank their project tutor, Bereket A. Yilma, for their guidance and support as a Project Academic Tutor. He has been a great help throughout the whole project duration and taught us everything we needed to know about deep learning in context of image classification.

6. Plagiarism statement

This 350 words section without this first paragraph must be included in the submitted report and placed after the conclusion. This section is not counting in the total words quantity.

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] Cs231n: Convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/. Accessed: 09/01/2024.

[2] Samah S Baraheem and Tam V Nguyen. Ai vs. ai: Can ai detect ai-generated images? *Journal of Imaging*, 9(10):199, 2023.

[3] Jordan J Bird and Ahmad Lotfi. Cifake: Image classification and explainable identification of ai-generated synthetic images. *arXiv preprint arXiv:2303.14126*, 2023.

[4] Riccardo Corvi, Davide Cozzolino, Giada Zingarini, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. On the detection of synthetic images generated by diffusion models. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[5] Ahmed A Elngar, Mohamed Arafa, Amar Fathy, Basma Moustafa, Omar Mahmoud, Mohamed Shaban, and Nehal Fawzy. Image classification based on cnn: a survey. *Journal of Cybersecurity and Information Management*, 6(1):18–50, 2021.

[6] Luca Guarnera, Oliver Giudice, and Sebastiano Battiato. Level up the deepfake detection: a method to effectively discriminate images generated by gan architectures and diffusion models. *arXiv preprint arXiv:2303.00608*, 2023.

[7] Aman Gupta, Rohan Ramanath, Jun Shi, and S Sathiya Keerthi. Adam vs. sgd: Closing the generalization gap on image classification. In *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*, 2021.

[8] Alexandros Iosifidis and Anastasios Tefas. *Deep learning for robot perception and cognition*. Academic Press, 2022.

[9] Peter Lorenz, Ricard L Durall, and Janis Keuper. Detecting images generated by deep diffusion models using their local intrinsic dimensionality. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 448–459, 2023.

7. Appendix

7.1. Source Code

Github Link: https://github.com/angelri03/ai-art-detector