Universiteit Leiden

# Master Computer Science

Deep Learning for Embryo Selection in
In Vitro Fertilization


| | |
|---|---|
| Name: | Vincent van der Sluis |
| Student ID: | s2675358 |
| Date: | [06/06/2023] |
| Specialisation: | Bioinformatics |
| 1st supervisor: | Dr. Daan Pelt |
| 2nd supervisor: | Dr. Oleh Dzyubachyk |


Master's Thesis in Computer Science


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# Contents


1 Formula sheet 6

2 Introduction 8
- 2.1 Success rate & Fertility . . . 8
- 2.2 Implantation potential . . . 10
- 2.3 Arti cial intelligence solution . . . 11
- 2.4 Goals and problem de nitions . . . 12

3 Background 13
- 3.1 IVF & ICSI fertilization . . . 13
- 3.2 Embryo phases . . . 13
  - 3.2.1 Phase: Blastulation - Cleavage . . . 13
  - 3.2.2 Phase: Blastulation - Blastocyst . . . 14
  - 3.2.3 Phase: Implantation natural vs in vitro . . . 14

4 Technical Background: Deep Learning 15
- 4.1 Neural network . . . 15
- 4.2 Backpropagation . . . 16
- 4.3 Activation functions . . . 17
- 4.4 Model loss . . . 18
  - 4.4.1 BCE Loss Metrics . . . 18
- 4.5 Batch normalization . . . 19
- 4.6 CNN Architecture . . . 19
  - 4.6.1 Existing models . . . 22
  - 4.6.2 VGG architecture . . . 22
  - 4.6.3 E cientNet architecture . . . 23
  - 4.6.4 U-Net architecture . . . 24
- 4.7 Transfer Learning . . . 24
- 4.8 Performance Metrics . . . 25

5 Embryo Data & Incubator 25

6 Methods 27
- 6.1 Work ow - Pre-processing . . . 28
- 6.2 Determining the center of the embryo . . . 28
  - 6.2.1 Motivation for rejecting traditional image segmentation techniques . . . 28
  - 6.2.2 Neural network: center detection . . . 28
  - 6.2.3 U-Net: training parameter optimization . . . 31
  - 6.2.4 Obtaining embryo location from U-Net output . . . 32
  - 6.2.5 Embryo extraction & Coordinates smoothing . . . 33
  - 6.2.6 Coordinates smoothing . . . 34
- 6.3 Detection of the sharpest focal plane . . . 35
  - 6.3.1 The average gradient magnitude as sharpness rating . . . 36
  - 6.3.2 Sharpness selection . . . 37
- 6.4 Sharpness normalization . . . 38
- 6.5 Data sets for classi cation . . . 39

6.5.1 Image transformations and augmentations . . . . . 41

7 Training the proposed models 42
7.1 Proposed models . . . . . 43
7.1.1 TimeBlocks network design . . . . . 43
7.1.2 VGG-16 and E cientNet . . . . . 45
7.2 Hyperparameter tuning overview . . . . . 47

8 Results 48
8.1 Model performance: embryo center detection . . . . . 48
8.2 Trajectory estimation and smoothing . . . . . 50
8.3 Detection of the most informative focal planes . . . . . 53
8.3.1 Temporal sampling . . . . . 55
8.4 Neural network results . . . . . 56
8.5 TimeBlocks network . . . . . 56
8.5.1 TimeBlocks - optimizer & batchnorm in uence . . . . . 56
8.5.2 TimeBlocks: parameter in uence . . . . . 57
8.6 VGG16 . . . . . 62
8.6.1 VGG16 - Training loss & Over tting . . . . . 62
8.6.2 VGG16: batch normalization and test loss . . . . . 64
8.6.3 VGG16: parameter in uence on the Default data set . . . . . 66
8.7 E cientNet . . . . . 68
8.7.1 E cientNet: over tting . . . . . 68
8.7.2 E cientNet: optimizer in uence . . . . . 68
8.7.3 E cientNet: parameter in uence . . . . . 71

9 Discussion 74
9.1 Pre-processing: work ow . . . . . 74
9.2 Proposed models: over tting . . . . . 74
9.3 Proposed models: parameters . . . . . 75
9.3.1 TimeBlocks models: speci c parameters . . . . . 76
9.4 Research Question & Sub-questions . . . . . 76
9.4.1 Research Question: Can an AI model predict the implantation potential of an IVF embryo based on time-lapse developmental videos? . . . . . 77
9.5 Limitations and Future work . . . . . 77

10 Conclusion 78

A Appendix 79

Abstract

In vitro fertilization is a crucial treatment for individuals experiencing infertility due to various reproductive disorders. Despite advancements, IVF success rates have shown limited improvement over the years. The implantation potential of an embryo is an important factor for the success of an pregnancy. Currently, this potential is determined by the embryologist through video assessment, which is not optimal. The goal of this study is to develop an AI model that can predict the embryo implantation potential using longitudinal time-lapse imaging data. The implementation of such an AI model can lead to faster and more accurate embryo assessment, bene ting the healthcare system and improving IVF treatments. A total of 774 videos were obtained from the 92 labelled embryos. Embryos whose implantation resulted in abortion or menstruation were labelled as negative, and embryos resulting in pregnancy were labelled as positive.

The pre-processing involved localizing the embryo using an U-Net, cropping the embryo's ROI, determining z-slice quality, and generating di erent training data sets. Linear subsampling was applied to create subsets with a length of 10, 50, and 75. To prevent the lose of morphological context, surrounding frames were added.

Three classi cation networks: the adapted versions of the VGG and E cientNet, and the TimeBlocks model, were trained on these data. A total of 960 experiments were performed. Our results show that none of the models was able to learn features related to the implantation potential. Instead, we found that the models over t on the training set. This was mainly caused by having too few embryos in the data set, combined with label imbalance, forcing the models to learn trivial features.

Acknowledgements

First and foremost, I would like to thank my supervisors Dr. Oleh Dzyubachyk and Dr. Daan Pelt for their valuable feedback and mentoring throughout the project.

I would also like to thank Dr. Gonneke Pilgram and Diego Diaz de Pool from the Gynecology Department for their enthusiasm, active role in this project and for providing answers to all my questions regarding the IVF treatment.

Furthermore, I would also like to thank all the staff at LKEB for providing me with all the necessary resources, (technical) support and the pleasant working environment. I am very grateful that the LKEB gave me the opportunity to attend the Medical Imaging with Deep Learning conference in Zurich.

As last, a special thanks to my girlfriend Del na Gonzalez Lopez for motivating me during challenging times.

# 1 Formula sheet

Table 1: Formula sheet: introduction

| Variable Name | Context | Section |
|---|---|---|
| | Introduction | |
| $w$ | Weights | 4:1 Neural network |
| $z$ | Weighted sum | |
| | Activation function | |
| $b$ | Bias | |
| $x_i$ | Each neuron | |
| $w_i$ | Weights | |
| $a$ | Activation function output | |
| $D = f(i_1; o_1); : : : ; (i_n; o_n)g$ | Full dataset | |
| $(i_1; o_1)$ | Data input, data output | |
| | Weights and biases | 4:2 Backpropagation |
| $_j$ | The partial derivative of each parameter | |
| Loss | General loss function | |
| ReLU(z) | Relu | 4:3 Activation functions |
| $e$ | Euler's number | |
| $(z)$ | Sigmoid | |
| $\hat{o}_{loss}$ | Prediction loss | 4:4 Model loss |
| $\hat{o}$ | Prediction output | |
| BTE | Bernoulli trial entropy | |
| $n$ | All samples in dataset | |
| $B_{loss} = \quad (o\log(\hat{o}) + (1 \quad o)\log(1 \quad \hat{o}))$ | BCE output | |
| $B_{loss}$ | Average BCE | |
| $B = f x_1; x_2; x_i; x_n g$ | A batch | 4:5 Batch normalization |
| $\hat{x}_i = \frac{\check{x}_i \quad B}{\frac{2}{B} +}$ | Batch normalization Function | |
| $\hat{x}_i$ | Normalized output 1 | |
| | Smoothing term | |
| | Batch normalization scale value | |
| $Z_i$ | Normalized output 2 | |
| | Shift | |
| $s_h; s_w$ | Horizontal, vertical strides | 4:6 CNN architecture |
| $f_w \quad f_h$ | Kernel's width and height | |
| $z_{ijk}$ | Feature map | |
| $x_{ijk}$ | CNN input | |
| $w_{ijk}$ | Kernel weights | |
| $\hat{o}_{acc}$ | Accuracy | 4:8 Performance Metrics |
| $\hat{o}_{recall}$ | Recall | |
| $\hat{o}_{precision}$ | Precision | |

# Table 2: Formula sheet: methods

| Variable Name | Context | Section |
|---|---|---|
| | Methods | |
| $C_x$ | X coordinate | 6:2:2<br>Center detection |
| $C_y$ | Y coordinate | |
| $I$ | Intensity | |
| $I^0$ | Max normalisation | |
| $t$ | Image threshold value | |
| $Q^t(x)$ | Function for binary thresholding | |
| $m$ | Number of U-net feature maps | 6:2:3<br>U-Net: training parameter optimization |
| $H(x)$ | Heaviside step function | 6:2:4<br>Obtaining embryo location from U-Net output |
| $t = 0:0001$<br>$P(\hat{o}) = \frac{1}{n}\sum_{i=1}^{n} \hat{o}_i;$<br>$E^t(P) = H(\hat{o} - t):$ | Embryo presence detection | 6:2:5<br>Embryo extraction & Coordinates smoothing |
| $w(d) = (1 - \lvert d\rvert^3)^3$ | LOWESS Function | 6:2:6<br>Coordinates smoothing |
| $\lvert\lvert\nabla F\rvert\rvert = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$<br>$AGM = \frac{1}{n}\sum_{i=1}^{n} \lvert\lvert\nabla F\rvert\rvert$ | Average gradient magnitude | 6:3:1<br>The average gradient magnitude |
| $T_{n=0}$ | Start of sequence | 6:3:2<br>Sharpness normalisation: |
| $T_p$ | Number of timepoints | 6:5<br>Data sets for classi cation |
| $T_b$ | Numbers of mini-networks | |
| $T_f$ | Feature maps of each mini-network | |
| $T_c$ | Timepoint context | |
| $T_d$ | Timepoint focal planes (dimensions) | |
| $T_{bn}$ | Batch normalisation status | |

# 2 Introduction

In Vitro Fertilisation (IVF) is an important treatment for people suffering from some form of infertility, such as low sperm concentration, ovulation disorders, endometriosis, uterine fibroids [1]. Infertility is described as a disease of the reproductive system defined by the failure to achieve a clinical pregnancy after 12 months or more of regular unprotected sexual intercourse. It affects between 48 million couples and 186 million individuals worldwide, representing 15% of couples of reproductive age worldwide, and is classified as a global health issue by the World Health Organization [2]. For a long time, the desire of these individuals to reproduce stayed unfulfilled. This changed with the emergence of fertilization techniques like IVF and ICSI (Intracytoplasmic sperm injection). This provided an option for individuals suffering from infertility diseases.

The IVF technique has been proven to be effective, with the first successful humans application occurring in 1978. The birth of Louise Joy Brown, the world's first test-tube baby, was the result of the procedure being used to bypass Lesley Brown's blocked fallopian tubes [3]. After that, millions of successful cases followed. The IVF treatment is a complex process that involves several steps, including: ovary stimulation, oocyte (egg) retrieval, sperm preparation, fertilization, embryo selection and transfer of the fertilized oocyte into the uterus, in order to achieve a pregnancy.

## 2.1 Success rate & Fertility

It is important to note that IVF treatment success is not guaranteed and depends on patient-related factors, including: women's age, physical stress and sperm quality. Additionally, external factors, such as the culture medium and assessment of the embryologist, who is responsible for managing the development of the embryo and selecting the embryos, also play a significant role in the success of the IVF treatment [4].

Among these factors, the age of the oocyte is described as one of the most important and determinant factor for the success rate of the treatment. This is because the likelihood of fertility problems, such as uterine fibroids and endometriosis, increases with age [5]. There is also a strong relationship between ageing and having an abnormal number of chromosomes [6]. Despite this, the average age at which women become mothers has increased over the years. In 2021, the age for the first child was 30.3 years, and for the second child 32.3 years, according to the Centraal Bureau voor de Statistiek (Central Bureau of Statistics) in the Netherlands [7].

Assisted reproductive technology statistics show that women under the age of 35 have a 1.3 times greater chance of delivering a live birth compared to women aged 35-37. This number jumps to 2.0 for women aged 38-40 and rises further to 5.6 for women over the age of 40, see Figure 1A. After the age of 40, the chances of becoming pregnant decrease with each passing year until they are almost zero after the age of 43 [8]. The number of egg retrieval attempts before giving birth is also an unfavourable factor for older women, see Figure 1B.

(a) (b)

Figure 1: Women age statistics for ART (assisted reproductive technology) success, for 4 age groups [9]. a) The percentage of success rate of live-birth deliveries per embryo transfer, based on 107,795 ART cycles. For the 3 groups including: all births (including twins) in blue, only singleton births (orange), and singleton births with normal weight (green). The chance of giving birth is negatively correlated with the age of the woman. b) The average number of egg retrievals necessary for giving birth, based on 47,847 ART cycles. The number of egg retrievals is negatively correlated with the woman's age.

As mentioned above, giving birth through IVF treatment is not a guarantee. To increase the chances, multiple embryos are transferred rather than just one, which increases the likelihood of multiple pregnancies and potential complications for both mother and child. In addition, transferring embryos is also very expensive and can be emotionally heavy. According to the Dutch organisation Landelijke infertiliteit registratie (Infertility registration foundation), the overall IVF success rate of embryo-transfers in 2020 is 74.6%, and for ongoing pregnancy 27.1% per embryo-transfer [10]. Looking at the results of previous years, it is surprising to see that there has been little to no improvement in this area in recent decades, see Figure 2. This shows that despite decades of IVF study, there is still a lot of room for improvement regarding IVF/ICSI treatment.

Figure 2: Embryo transfer and pregnancy successes in IVF/ICSI treatment [11]. Neither the success rate of embryo transfers nor pregnancies have improved since 1998. The year 2018 is missing from this graph.

Additionally, recent articles have shown a decline in fertility rates in men and women. In the last 50 years, the sperm concentration of European men has decreased by 32.5 percent [12]. Other studies have shown a decline of 50{60% in North America-Europe-Australia and China [13, 14]. Rising exposure to environmental toxins found in food, water, household items, personal care products, clothing, carpeting, packaging, cooking materials, electronic and electrical equipment, etc. has been linked to potential harm to male reproductive health [15]. The role of chemicals in women's fertility is less studied. However, it is shown that certain chemicals and hormonal-disrupting substances harm the fertility of women (2023) [16]. If this trend continues, it is likely that humanity will become more dependent on assisted reproductive technology like IVF for its existence.

## 2.2 Implantation potential

An important factor for successful pregnancy is the implantation potential of an embryo. The implantation potential is the likelihood of an embryo implanting in the uterus and remaining attached until birth. On average 8 oocytes(eggs) are retrieved during oocyte pick up, varying from 1 to 35 oocytes. Only one or two of the highest-quality embryos are typically chosen for embryo transfer. This increases the chance of a successful pregnancy outcome, because these embryos have a greater implantation potential. Implantation potential depends on multiple features, such as cell number(s), embryo fragmentation, multinucleation, blastocyst formation and inner cell mass distribution [17]. Using the aforementioned markers, assessment based on morphological and morphokinetic features can be done for obtaining the most viable embryo(s).

Figure 3: Overview of an embryo time-lapse video as seen by an embryologist. The embryologist determines the development state, shown in the timeline bar at the bottom. On the right, all the embryos from a patient are shown, ready to be graded.

The embryologist's task is to review embryos for embryo selection. They use various techniques to tackle this embryo grading task, the current standard being the adoption/screening of high dimensional time-lapse movies, see Figure 3. After fertilization, embryos are placed in incubators equipped with cameras that can be programmed to take pictures at certain time intervals [18]. These embryo time-lapses videos of the embryo are used for grading at di erent stages, such as pronuclear, cleavage and blastocyst stage [19].

## 2.3 Arti cial intelligence solution

Since the last century, humanity has made big steps in increasing ART successes. However, in recent years the improvements in this  eld have saturated, due to it being more complicated to come up with major breakthroughs. This is where AI (arti cial intelligence) could step in. AI has shown that it can solve problems that seemed unsolvable by humans. It is able to learn features and identify patterns in large data sets by mimicking the work of the human brain. Image classi cation is one of the many AI deep learning applications. Various studies have shown the usefulness of AI in healthcare applications [20]. An earlier pilot study showed the potential to extract features from single IVF images [21]. Another small study was able to predict uterine activity using multiple time-sequenced ultrasound images [22]. AI could potentially also be useful for time-lapse embryo development video prediction. By predicting the implantation potential, it can improve the success rate of IVF. Additionally, AI could also point out what image features are important through model diagnosis. This could have the potential to revolutionize the embryo screening procedure.

## 2.4 Goals and problem definitions

Our goal is to build a model that can predict the implantation potential of an embryo based on high dimensional time-lapse imaging data. Such a model could potentially evaluate an embryo in a matter of seconds, while an embryologist can take up to minutes. Unknown embryo implantation features can be discovered through AI, which can improve embryo selection, resulting in fewer embryo transfers. Implantation of such a model can benefit the healthcare system by integrating AI into daily practice, resulting in cheaper, faster and better IVF treatments for the patient and the clinic.

1. Research Question: Can an AI model predict the implantation potential of an IVF embryo based on time-lapse developmental videos?
   (a) Sub-Question 1: Is it possible to localize the position of an embryo in the frame?
   (b) Sub-Question 2: Is it possible to identify the most informative focal planes for a given time frame?
   (c) Sub-Question 3: What is the impact of the different data set configurations on the performance?
   (d) Sub-Question 4: Is a multi-branch network structure able to learn the features of a specific time interval?
   (e) Sub-Question 5: Are the traditional models VGG16 and EfficientNet able to adapt to our classification task?

# 3 Background

In this section, additional background information about the IVF treatment, embryo development and deep learning understanding will be given. The IVF and ICSI treatment are discussed in Section 3.1. Section 3.2 will zoom in on the di erent development phases, and their associated factors/features. Section 4 gives a brief technical assessment of the machine and deep learning methods.

## 3.1 IVF & ICSI fertilization

The two most standard methods for in(vitro) fertilization are IVF and ICSI. During IVF treatment, sperm cells must swim on their own, before penetrating the oocyte. In contrast, in ICSI treatment (intracytoplasmic sperm injection), a single sperm is injected into the cytoplasm of an oocyte. The main di erence between the 2 methods is the number of sperm cells needed: where IVF needs 50 to 100 thousand, ICSI requires only a single cell. These processes are shown in Figure 4. Only one sperm cell can bind to the *Zona Pellucida*, which surrounds the oocyte. This *Zona Pellucida* mainly consists of glycoproteins. After fertilization, a zygote is formed [23].

Figure 4: A simpli ed visualization of the ICSI and IVF techniques. ICSI involves injecting a sperm into an oocyte via a sharp injection pipette. In IVF thousands of sperm cells are used for insemination of the oocyte. Figure adapted from [24].

## 3.2 Embryo phases

The zygote needs to go through several transitions before it will be suitable for implantation. During the  rst 5 days, the zygote undergoes rapid cell divisions and develops into a blastocyst. It is very important for embryo selection that a zygote reaches the blastocyst stage. This is the stage most likely to lead to pregnancy and is therefore one of the few selection features [25]. In the selection below, we give a brief overview of the aforementioned phases.

### 3.2.1 Phase: Blastulation - Cleavage

Cleavage is the  rst step of the *blastulation*. The cleavage period starts after the  rst mitotic cell division, resulting in the zygote having two cells. This process of doubling the number of cells is repeated every 12 to 24 hours. Despite the rapid mitotic divisions, the overall size of the zygote remains unchanged for each cycle, but the cytoplasmic and nuclear material increases. The reason for the size remaining the same is of the *Zona Pellucida*, which acts as a protective

membrane of glycoproteins that surrounds the oocyte cell, also known as the 'eggshell'. At a stage of 32 cells, the *morula* is formed [26, 27].

### 3.2.2 Phase: Blastulation - Blastocyst

The second step of the *blastulation* is the formation of a blastocyst. This happens on the fourth and fth day after fertilization. At this stage, certain outer layer cells of the *morula's* undergo permanent di erentiation and therefore develop more speci c forms and function. Blastocyst formation is characterized by the formation of the blastocoel (a uid- lled cavity) and thropectoderm cells, as shown in Figure 5. The trophectoderm is formed by trophoblasts cells (the outer layer of cells) and will later develop into the placenta. The trophoblasts plays a crucial role in embryo implantation, forming a trophic interface between the embryo and the mother [28].

The cells forming the inner cell mass within the trophoblast are rearranged to form a hollow space that is lled with a uid- lled or yolk- lled cavity called the blastocoel. As shown in Figure 5, the blastocoel is completely surrounded by the trophoblast. The inner cell mass cells are important or the body/structure of the embryo [27].

### 3.2.3 Phase: Implantation natural vs *in vitro*

In natural conception, the blastocyst travels through the fallopian tube towards the uterus. Implantation of the blastocyst into the wall of the uterus results in a pregnancy. However, *in vitro* implantation the process is a bit di erent. It starts with selecting the best embryo(s) according to the embryologist. Next, the embryo is transferred immediately into the uterus. Surplus embryos from the IVF/ICSI treatment are frozen and stored in liquid nitrogen for subsequent treatments.

Figure 5: A simpli ed visualisation of the transition of morula to blastocyst phase. Blastocyst contains trophoblast cells at the outer layer, and a hollow space(cavity) is formed named the *blastocoel* Figure adapted from [29].

# 4 Technical Background: Deep Learning

Deep learning is a specialized area within the broader field of machine learning, which focuses specifically on neural networks. The (multiple) layers of neutrons and their connections define the neural network architecture. In this section, we will focus on the technical details regarding neural networks, including the processing of visual data through convolutional neural networks (CNNs). We will also explore other CNN-based models, relevant to our embryo selection task.

## 4.1 Neural network

Neural networks are capable of performing a wide variety of tasks, such as recognition of handwriting, voice recognition and image classification [30]. Before a network is able to perform such tasks, it is trained, a process also known as *learning*. This type of learning is based on the idea of information processing and learning in the biological brain. The human brain consists of approximately 86 billion neurons [31]. Due to this high number of neurons, the brain is capable of learning and performing complex tasks such as reasoning, problem-solving, recognizing patterns, understanding language and making decisions [32].

The single-layer *perceptron* is one of the earliest architectures for translating the biological neural network architectures to an artificial neural network [33]. Here, the perceptron acts as a neuron by receiving input from other neurons, see Figure 6A. The single-layer perceptrons are capable of solving any linearly separable task. However, they are unable to solve more complex tasks, such as exclusive OR problems. This is because of the linearity of the output neurons [34]. This shortcoming can be solved by stacking multiple layers, creating a MLP (Multi-layer Perceptron), see Figure 6B [30]. This is important, because the embryo selection is not a linearly-separable task.

The MLP structure consists of multiple layers stacked on top of each other. A layer consists of neurons $x$ and each neuron is connected to all other neurons in the subsequent layer, resulting in a *dense layer*. The strength of a connection is represented by its weight $w$. The input of the neurons in a layer is the weighted sum $z$ of all values in the previous layer. The weighted sum is converted by an activation function $a$, which will be discussed further in Section 4.3. Additionally, a bias neuron $b$ with a value of 1.0 can be added to a layer. The model parameters include both the bias term and the weights. Computing the weighted sum of a neuron can be summarized as [31]:

1. First, for each neuron in the next dense layer $x_{i+1}$ the weighted sum $z_{x_{i+1}}$ is computed:

$$z_{x_{i+1}} = b + \sum_{i=1}^{n} x_i \; w_i;$$

   where $n$ is the number of input neurons $x_i$ for $x_{i+1}$ connected by the weights $w_i$.

2. The weighted sum is converted using an activation function

$$a_{x_{i+1}} = \quad (z_{x_{i+1}})$$

(a) (b)

Figure 6: Examples of the single and multi-layer perceptrons. The $x_n$ represents the inputs of the input layer.
a) This single-layer perceptron contains only 1 (output) layer, shown in blue. The lines represent the weights, each $x_n$ is connected to all neurons in the next layer. The neurons take the sum of all input values multiplied by the weights of the previous layer and apply an activation to it. The output layer contains 3 neurons, representing the 3 prediction classes [30] (modified).
b) A multi-layer perceptron solving the XOR problem. The XOR problem refers to the classification of binary input data, which cannot be linearly separated. As seen on the left, no single line can separate the 2 classes. The right part shows the MLP network that can solve the XOR problem. Each connection, except for the one where the weights are shown, has a weight of 1.0. The proof can be seen in Appendix A.3. Figure adapted from [30].

Just like the biological brain must learn to function effectively, the network also has to learn in order to achieve good results. This learning is done during the training phase, a training set $D_{train}$ $D$ is part of the full data set $D = f(i_1; o_1); (i_2; o_2); : : : ; (i_n; o_n)g$. Here $(i_n; o_n)$ represents the data sample and its outcome. The test set $D_{test}$ $D$ is used to evaluate the success of the model. During training, the network is fed one or more instance(s) at a time, the number referred to as the batch size. The error resulting from an incorrect prediction is used to update the weights.

## 4.2 Backpropagation

After the creation of MLPs researchers have struggled for years to train MLPs successfully, which changed with the invention of backpropagation [35]. Backpropagation is a technique used to optimize the parameters $\theta$ (weights and biases) of the model in order to fit the network to a particular problem. After each feedforward computation, the error is propagated backwards from the output nodes to the input nodes. The partial derivative of each parameter $w_j$ with respect to the loss is computed using the chain rule, making this step fast and precise. The partial derivative represents the contribution that the parameters have with respect to the loss, noted as [30]:

$$\frac{@}{@_j}\text{Loss}(\ )$$

The partial derivatives are optimized to minimize the cost function, which is generally done by GD (Gradient Descent). The goal is to update the $\theta_j$ until the algorithm converges to a minimum. A partial derivative represents the directions of the greatest ascent (uphill), relative to the loss function. The opposite direction (downhill) is used to update $\theta_j$, because we want to find the local minima; see Figure 7A. Finding the minimum can be challenging, if there are multiple local minima's; see Figure 7B [30].

(a) (b)

Figure 7: Examples of the gradient descent algorithm for optimizing $\theta$. At step 0, the $\theta$ is randomly initialized. The horizontal axis represents the change relative to $\theta$. The vertical axis represents the Loss of the model. Figure adapted from [30].
a) The gradient descent algorithm for optimizing the initial $\theta$. Each backpropagation step brings the $\theta$ closer to the optimal minima $\hat{\theta}$ until it converges. The learning rate is a variable that controls the amount of change of the $\theta$ per step and is proportional to the slope, resulting in a smaller change when the parameter gets closer to the minimum. In this example the Loss does not converge to 0, which is beneficial, because Loss of 0 likely indicates overfitting.
b) Example of gradient descent pitfalls visualized by 2 different random initializations. The left one converges to a local minimum, which is worse than the global minimum. The right one takes a long time to reach the minimum.

The amount by which the weights are updated during GD is determined by the learning rate. This is usually a small value ranging between 0.0 and 1.0. A small learning rate could lead to slower convergence. However, a large learning rate could lead to 'jumping' over the minimum.

The backpropagation step can be done for one training sample; however it is typically performed for multiple training samples, known as mini-batch. This can be beneficial, because it speeds up the training process. It averages the loss for all samples in a batch, reducing the influence of noisy or outlier samples on the gradients [30].

## 4.3 Activation functions

In this project, we make use of the activation functions ReLU (Rectified Linear Unit) and Sigmoid. Activation functions take the weighted sum $z$ as input and apply a non-linear mathematical function to it. The ReLU function is a commonly used activation function. After the ReLU transformation, neurons with a negative weighted sum are deactivated [36]. Nevertheless, ReLU is not suitable for implementation in the output layer, because the output

layer should act as a classi er, which in our case would be binary. The Sigmoid function is a commonly used function for binary classi cation. The Sigmoid is a logistic function that returns values in the range 0{1, which makes it easy to map these values to 0 or 1 for binary classi cation.

$$\mathrm{ReLU}(z) = \max(0; z)$$

$$(z) = \frac{1}{1 + e^{\ z}}$$

## 4.4 Model loss

The Loss function measures the di erence between the predicted output $\hat{o}$ and the actual output $o$. This Loss is a crucial component of the training, because it helps to adjust the model parameters towards the minima through backpropagation. In the eld of deep learning there are several di erent loss functions. For the scope of this project, we are only interested in binary loss functions. Therefore, only the BCE (Binary Cross-Entropy) loss function is addressed.

### 4.4.1 BCE Loss Metrics

After each forward propagation of the inputs, the loss is calculated. The prediction $\hat{o}$ is given by the nal layer of the model and ranges between 0.0 and 1.0. The BCE loss uses the Bernoulli trial entropy $BTE$. This function gives the amount of uncertainty found for a given prediction $\hat{o}$.

$$BTE(p) = \quad \hat{o} \quad \log_2(\hat{o}) \quad (1 \quad \hat{o}) \quad \log_2(1 \quad \hat{o})$$

Ideally, we want an entropy of 0.0 between $\hat{o}$ and the actual true $o$. If a prediction is 0.5, then the entropy is at its maximum, because according to the Bernoulli trial, the prediction outcome is just as in favour of label 0 as of 1 [37]. The Bernoulli trial entropy function is closely related to the BCE function. The di erence is that the Bernoulli trial entropy function only measures the uncertainty, whereas the BCE loss uses this uncertainty for penalizing the model for incorrect predictions. The model will minimize the entropy of the Bernoulli distribution, through the use of BCE loss function.

$$B_{loss} = \quad (o \log(\hat{o}) + (1 \quad o) \log(1 \quad \hat{o}))$$

The aforementioned formula for calculating the BCE loss $B_{loss}$ can be separated into 2 parts: the $o \log(\hat{o})$ part calculates the BCE loss for $o = 1$, and the $(1 \quad o) \log(1 \quad \hat{o})$ part for the $o = 0$ samples. A prediction that di ers a lot from the ground truth is penalized more by the logarithmic function than those that are closer to the truth.

The average BCE $B_{loss}$ for all samples $n$ in data set $D$ is one of the metrics used to evaluate model performance.

$$B_{loss} = \quad \frac{1}{n} \sum_{i=1}^{n} (o \log(\hat{o}) + (1 \quad o) \log(1 \quad \hat{o}))$$

## 4.5 Batch normalization

In the backpropagation section (4.2) we discussed the possibility of training a network using mini-batches. The difference is that the model is not trained on a single instance, but on multiple instances using mini-batches. Research showed that normalization within a batch can be beneficial, because it speeds up training and could potentially improve performance [38].

The batch normalization layer takes a batch $B$ as input: $B = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$. First, the mean $\mu_B$ and the standard deviation $\sigma_B$ of the batch are computed. Second, each batch sample $x_i$ is normalized to $\hat{x}_i$:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Here, a small smoothing term $\epsilon$ is added in case the $\sigma_B$ is zero. After normalizing the samples, the batch has a mean of 0 and a standard deviation of 1. In other words, all samples in the batch are on the same scale. However, it is not beneficial for all batches to have $\mu_B = 0$ and $\sigma_B = 1$, because different batches may have different statistics and distributions. This is where the main strength of batch normalization comes in, by introducing 2 learnable parameters $\gamma$ and $\beta$.

The influence of $\gamma$ is to scale the standard deviation, which is achieved by multiplying the normalized values by a factor $\gamma$. The $\beta$ parameter is then added, which causes a shift to a different mean. This operation gives the batch normalized output $Z_i$ [30].

$$Z_i = \gamma \hat{x}_i + \beta$$

These learnable parameters allow each batch normalization layer to set the inputs to a different scale, allowing the hidden layers to effectively make use of the normalization through backpropagation [38].

## 4.6 CNN Architecture

The embryo data set contains multiple videos, where each video is essentially a stack of images that are displayed at a certain frame rate. If a traditional fully-connected network were to be used to train a model, it would require a large number of additional parameters, resulting in very large networks. This is because a fully-connected network would treat each individual pixel as a neuron in the input layer. The size of these big networks would have as consequence, that it could make optimization too computationally intensive. Another drawback of having too many parameters is that the model is more prone to overfitting, particularly when the data set is sparse, as in our case.

To overcome these limitations, a convolutional neural network is preferred when processing visual input such as images or video data. A typical convolutional neural network structure consists of a series of convolutional layers combined with max-pooling and activation layers, before being connected to a fully-connected layer, see Figure 8. In this section, we describe the notion of convolutional layers [30]. We also introduce 3 successful CNN models: **LeNet**, VGG16 and Google EfficientNet, which are covered later in the Methods section of this manuscript.

Figure 8: Simple CNN architecture. Convolution and pooling computations are performed on the input image with only 1 depth. The resulting feature maps are connected to a fully-connected network. Figure adapted from [30](modi ed).

Embryo images contain useful features, such as: edges, textures, shapes, and patterns. A convolution layer can capture these features using learnable kernels. During the forward-pass of a single channel image, 2D kernel convolutions are applied across the height $x_i$ and width $x_j$ of the image by scanning the entire image. The amount of movement during each scanning step is determined by the horizontal and vertical strides $s_w$, $s_h$.

The area where the convolution takes place is referred to as the receptive eld. The convolution is a mathematical multiplication of 2 matrices, including the receptive eld and the kernel $w_k$. Each operation returns one value and all these values together form a feature map $z$. The number of kernels and their size can vary across convolutional layers. The width and height of a kernel are represented by $f_w$ and $f_h$. The number of feature maps is equal to the number of di erent kernels applied to the input. Each kernel $w_k$ creates the feature map $z_k$.

The resulting feature map is smaller than the input layer, because the kernel center is unable to reach the border pixels while scanning the image. In order to preserve feature information around the border of an image, padding is used around the outer frame of the input layer, keeping the spatial resolution the same, see Figure 9.

Figure 9: The convolution operation is performed on a 2D input layer (bottom), which creates the feature-map (top). The input layer has been zero-padded (shown in gray). The feature-map $i$ $j$ is created using single-kernel convolutions across the height and width using a kernel size of $f_w$=3, $f_h$=3. This results in the feature-map $z_{ijk}$ . The kernel stride was set to 1. The colored boxes of size $i$ $j$ represent the connection between the kernel and the receptive eld of the input layer. Figure adapted from [30].

A feature map corresponds to the activation of di erent part(s) of the image. Each feature map represents a speci c feature or pattern that the neural network has learned to recognize in the input image. By combining multiple convolutional layers on top of each other, the network is capable of learning more complex patterns and concentrate on small low-level features in the earlier hidden convolutional layers.

Above, we discussed convolutions on a single 2D input channel, e.g. a single greyscale image. However, our embryo time-lapse data set contains many (time) channels. A 2D convolution with a channel depth higher than 1 acts more as 3D convolution, because the kernel slides not only over the 2D image with the height $x_i$ and width $x_j$, but also in the depth dimension $x_{k'}$ of the input. This means that the kernel weights also extend in the depth dimension, resulting in kernel weights as 3D metrics with size $w_i \quad w_j \quad w_{k'}$. The sliding windows receptive field size is $f_w \quad f_h \quad f_{n'}$.

A full convolution of $x_{ijk'}$ with the kernel weights $w_{ijk'}$ results in a single feature map of size $z_{ijk}$, see Figure 10. After a full convolution, the bias value $b_k$ is added to the feature map value $z_{ijk}$.

Figure 10: Representation of multiple input channels $x_{ijk'}$ highlighted in blue, the receptive field of the input channels of size $f_{whn'}$ highlighted in blue and red, a filter with the kernel weights $w_{vuk'}$ in orange, and the feature map $z$ in green. Each value $z_{ijk}$ in the feature map is obtained by applying the filter $w_{vuk'}$ to the current receptive field. Figure adapted from [39].

Additionally, the feature maps can be further processed. A useful technique is to reduce the size of these feature maps by max pooling. The pooling operation applies a sliding window approach, similar to convolutional operations. However, instead of a trainable filter, it uses a fixed maximum operator. For instance, a 400x400 image with a 2x2 kernel results in a feature map of 200x200. Pooling results in a reduction of spatial information. This helps to reduce variance and amount of computations. However, excessive pooling has a downside of removing too much information.

### 4.6.1 Existing models

In recent years, researchers have used CNNs to successfully develop complex models [40]. It turned out that the architecture of these models was not necessarily bound to a speci c task. In other words, these models could be adapted, to t a certain purpose, e.g. the embryo implantation classi cation task. In this manuscript, 3 well-known CNNs are used: VGG16, Google E cientNet and U-Net. Before implementing these models, it is essential to review the architecture beforehand. This is done in the section below, where we will discuss the architecture of each model.

### 4.6.2 VGG architecture

The VGG network is widely known for its image classi cation capabilities. It won the ImageNet competition in 2014, with 76.3% accuracy. In this competition, 224x224 colored images are used for object detection and image classi cation. The architecture of the VGG16 network can be seen in Figure 11. The number 16 in its name refers to the 16 layers containing weights, which include 13 convolutional layers and 3 fully-connected layers containing 4096, 4096, and 1000 neurons, respectively. The convolutional layers are grouped into 5 convolutional blocks, each block containing 2-3 convolutional layers, with the following statistics:

- Block 1: 64 lters, 2 convolutional layers, kernel size 3
- Block 2: 128 lters, 2 convolutional layers, kernel size 3
- Block 3: 256 lters, 3 convolutional layers, kernel size 3
- Block 4: 512 lters, 3 convolutional layers, kernel size 3
- Block 5: 512 lters, 3 convolutional layers, kernel size 3

Each block is followed by a max-pooling layer. The last block outputs 512 feature maps connected to the 3 fully-connected layers. Overall, the network includes around 138 million parameters [41].

Figure 11: The architecture of the VGG16 network. The network consists of 5 convolutional blocks, followed by fully-connected layers. The output layer contains 1000 nodes, for the 1000-class classi cation. Figure adapted from [42].

### 4.6.3 EfficientNet architecture

In 2020 Google authored the paper entitled "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" [43]. In this paper, the authors propose a framework for scaling, depth, width, compound and resolution in Convolutional Neural Networks to maximize the overall efficiency of the network. This approach resulted in a network that achieved 77.1% accuracy on the ImageNet data set. At first glance, this does not look very promising. However, the fact that this network consists of only 5.3 million parameters, indicates that the network efficiently makes use of its parameters. Additionally, it has been shown that the network is able to adapt to other data sets (CIFAR-100/Flower data set) using transfer learning, see Section 4.7.

In this paper, a family of "EfficientNet" networks is created; to keep this manuscript compact only the smallest network "EfficientNet-B0" is discussed. The architecture is shown in Figure 12. The "EfficientNet-B0" network consists of in total 237 layers, mainly of MBConv (mobile inverted bottleneck convolution) blocks [44]. In short, the MBConv block adapted in the paper is a combination of several operators in a specific order:

1. Pointwise convolution: "Cheap 1x1 convolution filter is applied to the input to reduce the channels, but preserve the inter-channel relationship. Followed by Batch normalization + swish activation.

2. Depthwise convolution: "A single convolutional filter is applied to each individual channel created by the Pointwise convolution. In this step, a stride of 1 or 2 is used, and padding is set to "same". Feature maps retain their size by having a stride of 1. At last batch normalization and swish activation are performed.

3. Pointwise convolution: "Followed by batch normalization only.

4. (optional) Residual skip connection: If the input used in step 1 has the same size as the output of step 3, it is added to the feature maps [43].

Figure 12: The "EfficientNet-B0" network has "Conv" and "MBConv" blocks to predict images. The "Conv" is a simple convolutional layer, with strides set to 2, while the "MBConv" blocks consist of Pointwise-Depthwise-Pointwise operations. The additional numbers within the "MBConv" block represent the number of Depthwise operations and the size of the kernel. The fully-connected layers for the class classification are not shown. Figure adapted from [45].

### 4.6.4 U-Net architecture

The U-Net is a different type of convolutional neural network compared to the previously discussed networks. Where the previous network tackled a classification task, the U-Net tackles an image segmentation task. An U-Net uses an image-in/image-out approach, instead of image-in/label-out one. Due to its ability to make precise and fast segmentation using a low number of training images, it is currently considered the gold standard for segmentation in the biomedical field [46]. It can locate an object with high accuracy, making it a good fit for our embryo localization task. The U-Net achieves precise predictions through an encoder-decoder structure using skip connections. This model does not include any fully-connected layers, which reduces the number of learnable parameters [47].

The input is being contracted by the encoder through a series of repeated filter convolutions and 2x max-pooling operations. The number of feature channels is doubled at each down-sampling step. The decoder is responsible for expanding the output through 2x up-sampling, and utilizes skip connections. The skip connection adds the corresponding encoder feature map to the decoder feature map via concatenation. Subsequent convolutions are performed on this feature map. The number of feature channels is halved at every up-sampling step. The final layer performs a 11 convolution to reduce the number of channels to the desired number of segmentation classes [47].

The U-Net name is based on the shape of the network. The general features are obtained via the encoder block, which increased the number of feature maps while down-sampling the spatial size. To reintroduce high-resolution features, skip connections between encoder and decoder layer are used. This leads to more precise output [47].

## 4.7 Transfer Learning

Transfer Learning is a technique where a pre-trained model is adapted for another task. Instead of randomly initializing the weights and biases, we use the pre-trained values. The benefit of this approach is that the network does not need to learn from scratch. Therefore, the network can devote less time to learning low-level structures in images, and more time to learning higher-level structures, which is referred to as transfer learning. The use of pre-trained models can speed up the training process and may even reduce the amount of required training data depending on the task [48]. The VGG and EfficientNet are both pre-trained networks on the ImageNet data set, which contains 1,000 classes. These pre-trained weights could be useful for predicting the embryo implantation potential.

## 4.8 Performance Metrics

The performance of trained AI models needs to be evaluated to ensure that the models are working properly. The goal is to avoid over tting or under tting on the training set. We measure the performance throughout the training process using four metrics: the Accuracy, Recall, Precision and the model loss. Accuracy is the simplest, simply dividing the total number of true and false positives (TP and FP) by the total number of samples in the training set. However, focusing solely on accuracy has the disadvantage that it provides no information about the model's bias towards a certain label.

$$\hat{o}_{acc} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\hat{o}_{recall} = \frac{TP}{TP + FN} \tag{1}$$

$$\hat{o}_{precision} = \frac{TP}{TP + FP}$$

To address these issues we also make use of recall and precision metrics. Recall measures the ratio of true positives (TP) against the total number of relevant elements (TP + FN). A high recall represents the model's ability to identify instances of a certain label and vice versa. The precision is the model's ratio between true positives and all the positives. It represents the ability of the model to correctly identify positive cases. The equation for the metrics is given in Eq 1.

# 5 Embryo Data & Incubator

For this project raw videos from 92 embryos were obtained from the embryologists at Leiden University Medical Center. All video material was captured by the Geri® incubator QRTM329, with a resolution of 928928 pixels. The Geri® incubator has 6 individual chambers, with 16 microwells each, see Figure 3. These chambers are equipped with a long-wavelength light source (550-650 nm) and a monochrome CMOS camera with a resolution of 2 pixels per and ve megapixels. The time an embryo was kept in the incubator ranged from 3 to 5 days. The acquisition time interval of the camera frames was set to 5 minutes. All frames together result in time-lapse videos consisting of 800 to 1550 frames. Each frame was captured with di erent focus settings, creating one video for each of the 7-11 focal planes per embryo. In total, 774 videos were obtained from the 92 labelled embryos.

The labelling was carried out individually by the LUMC embryologists: Diego Diaz de Pool and Dr. Gonneke Pilgram. An embryo implantation resulting in an abortion or menstruation receives a negative label (0), while an embryo ending in a pregnancy receives a positive label (1). The 92 embryos labels were not evenly distributed; a total of 35 embryos were labelled as positive and 58 as negative, resulting in a label imbalance of 37% against 63%. All samples were anonymized prior to processing and are shown in Table 3.

Table 3: All 92 embryo samples; each sample is provided with a Z1/M0 status indicating the outcome of the embryo transfer. Unsuccessful pregnancies are labeled with 0, and successful pregnancies with a 1. The data set has a slight class imbalance, with 37% resulting in a pregnancy and 63% resulting in an unsuccessful one.

| | ID | Z1/M0 | | ID | Z1/M0 | | ID | Z1/M0 |
|---|---|---|---|---|---|---|---|---|
| 1 | 4431-1-2021134-well07 | 1 | 1 | 4431-1-2021134-well10 | 0 | 36 | H2549-1-H 2021107-well03 | 0 |
| 2 | 4968-3-2021335-well04 | 1 | 2 | 5264-4-2021210-well07 | 0 | 37 | H2549-1-H 2021107-well06 | 0 |
| 3 | 5241-6-2021171-well15 | 1 | 3 | 5313-4-2021246-well01 | 0 | 38 | H2557-1-H2021122-well02 | 0 |
| 4 | 5448-3-2021173-well02 | 1 | 4 | 5313-4-2021246-well02 | 0 | 39 | H2563-1-2021132-well06 | 0 |
| 5 | 5476- 3 - 2021132-well02 | 1 | 5 | 5330-2-2021298-well13 | 0 | 40 | H2610-1-H2021211-well10 | 0 |
| 6 | 5476- 3 - 2021132-well03 | 1 | 6 | 5607-2-2021168-well03 | 0 | 41 | H2625-1-H2021061-well03 | 0 |
| 7 | 5476- 3 - 2021132-well04 | 1 | 7 | 5705-2-2022036-well05 | 0 | 42 | H936-2-H2021203-well09 | 0 |
| 8 | 5607-2-2021168-well02 | 1 | 8 | 5741-3-2021113-well10 | 0 | 43 | LR1524-5-2021215-well01 | 0 |
| 9 | 5607-2-2021168-well05 | 1 | 9 | 5741-3-2021113-well11 | 0 | 44 | LR1524-5-2021215-well04 | 0 |
| 10 | 5623-3-2021214-well05 | 1 | 10 | 5815-2-2021238-well02 | 0 | 45 | LR1524-5-2021215-well05 | 0 |
| 11 | 5705-2-2022036-well02 | 1 | 11 | 5823-2-2021233-well03 | 0 | 46 | LW2008-7-2021221-well04 | 0 |
| 12 | 5705-2-2022036-well03 | 1 | 12 | 5829-1-2021151-well02 | 0 | 47 | LW2008-7-2021221-well09 | 0 |
| 13 | 5741-3-2021113-well08 | 1 | 13 | 5850-1-2021198-well01 | 0 | 48 | R1515-2-R 2021051-2-well15 | 0 |
| 14 | 5769-1-2021241-well01 | 1 | 14 | 5850-1-2021198-well03 | 0 | 49 | R1566- 2 - R2021046-well04 | 0 |
| 15 | 5806-1-2021121-well04 | 1 | 15 | 5850-1-2021198-well05 | 0 | 50 | R1620-2-R2021055-well05 | 0 |
| 16 | 5850-1-2021198-well04 | 1 | 16 | 5850-1-2021198-well11 | 0 | 51 | R1624-1-R 2021032-well02 | 0 |
| 17 | 5855-1-2021250-well02 | 1 | 17 | 5854-1-2021204-well05 | 0 | 52 | R1624-1-R 2021032-well04 | 0 |
| 18 | 5886-1-2021252-well02 | 1 | 18 | 5874-1-2021232-well03 | 0 | 53 | R1627-2-R 2022094-well01 | 0 |
| 19 | 5935-1-2021316-2-well02 | 1 | 19 | 5874-1-2021232-well05 | 0 | 54 | R1627-2-R 2022094-well05 | 0 |
| 20 | 5965-1-2022079-well03 | 1 | 20 | 5874-2-2022021-well01 | 0 | 55 | R1635-1- R2021047-1-well03 | 0 |
| 21 | H2516-2-H 2021105-well07 | 1 | 21 | 5889-1-2021257-well14 | 0 | 56 | R1635-1- R2021047-1-well05 | 0 |
| 22 | H2563-1-2021132-well02 | 1 | 22 | 5902-1-2021275-well01 | 0 | 57 | R1635-1- R2021047-1-well07 | 0 |
| 23 | H2617-1-2021220-well07 | 1 | 23 | 5933-1-2021310-well08 | 0 | 58 | R1648-1-R 2021071-well05 | 0 |
| 24 | H936-2-H2021203-well01 | 1 | 24 | 6010-1-2022062-well01 | 0 | | | |
| 25 | H936-2-H2021203-well10 | 1 | 25 | H2471-2-H 2021115-well06 | 0 | | | |
| 26 | LD 74 - 2 - 2021169-well06 | 1 | 26 | H2485-2-H 2021072-well05 | 0 | | | |
| 27 | LH912-3-2021114-well02 | 1 | 27 | H2542-2-H2021196-well01 | 0 | | | |
| 28 | LH912-3-2021114-well04 | 1 | 28 | H2542-2-H2021196-well04 | 0 | | | |
| 29 | LR1524-5-2021215-well06 | 1 | 29 | H2542-2-H2021196-well08 | 0 | | | |
| 30 | LW2008-7-2021221-well06 | 1 | 30 | H2547-1-H 2021102-well02 | 0 | | | |
| 31 | R1624-1-R 2021032-well06 | 1 | 31 | H2547-1-H 2021102-well05 | 0 | | | |
| 32 | R1627-1-R 2021035-well02 | 1 | 32 | H2547-1-H 2021102-well08 | 0 | | | |
| 33 | R1627-1-R 2021035-well03 | 1 | 33 | H2547-1-H 2021102-well09 | 0 | | | |
| 34 | R1628-1-R 2021037-well08 | 1 | 34 | H2547-1-H 2021102-well10 | 0 | | | |
| 35 | R1635-1- R2021047-2-well06 | 1 | 35 | H2549-1-H 2021107-well02 | 0 | | | |

# 6 Methods

In order to predict the embryo implantation potential using deep learning, the neural network needs to be fed with correctly classi ed samples, constituting the training set. How well a network learns and generalizes depends heavily on the quality and size of the training data set, and the way the data are pre-processed. Factors such as image thresholding, resizing and image cropping play key roles; they can make or break the learning ability of a data set. In the \Methods" section, we will  rst go through all pre-processing steps developed as part of this project, and in the second part we will focus on the classi cation task and go deeper into the network structures.

Figure 13: The 5 steps of the data pre-processing pipeline. The  rst section focuses on the development of the U-Net for embryo detection. Next, this U-Net is used to crop out the embryo, through pixel thesholding and connected component analysis. This is followed by a sharpness analysis performed on all focal planes. Finally, di erent data sets are generated, varying in the number of time frames, the amount of focal plane depth and augmentations applied to the images like brightness and contrast modi cations.

## 6.1 Work ow - Pre-processing

The  rst part of the methodology consists in the pre-processing. Here, we convert the embryo videos obtained by the Geri incubator to a format that makes machine learning the most e cient. This part is divided into 5 main steps, see Figure 13. In steps 1{3, part of the image background is removed from the raw embryo images by cropping out the ROI around the embryo. This is done by tracking the embryo using a self-made U-Net and manually annotated center locations. In step 4, the quality of each z-slice is determined by a sharpness analysis. In the  nal pre-processing step, di erent training data sets are obtained, varying in time, depth and data size.

## 6.2 Determining the center of the embryo

One of the biggest limitations in training a neural network is the limited capacity of virtual memory in the GPU, which is necessary to  t a training batch along with the neural network itself. The smaller a training samples is, the more samples can  t into a batch. This raises the question: is it possible to make our data set more compact without losing valuable information? First, each video was converted to a set of gray scale images (TIFF). Each image is made up of pixels with gray scale intensities ranging from 0 to 255. Reviewing the data set shows that 928  928 images were captured inside a Geri® dish micro-well. Remarkable is that the embryo occupies only a small portion of this space during the  rst 5 days of development[1]. This leaves a large part of the image unused, resulting in an unnecessarily large batch size (in memory) and additional noise in our data set. In order to tackle this problem, the embryo is cropped out of the image. This is done by determining the center of an embryo using a center detection neural network, and consequent image cropping. Sections 6.2.2{7 will focus on the detection and cutting of the embryo out of the frame.

### 6.2.1 Motivation for rejecting traditional image segmentation techniques

Training of a neural network requires more preparation and background understanding (see Section 4) compared to the more straightforward/traditional techniques that rely more on heuristics such as threshold segmentation, edge-based segmentation, etc. Here, more focus lies on local di erences and pixel gradients in an image. Trial and error showed that these methods would not be suitable for embryo segmentation, because these methods were not consistent enough when applied to di erent samples; resulting in di erent setup per sample, increasing complexity, and negatively a ecting reproducibility. We do not want to claim that this approach would never work, but the amount of  ne-tuning necessary would make it time consuming and di cult to use in practice. Hence, time should be better invested in constructing a robust neural network, also capable of handling new/never seen samples.

### 6.2.2 Neural network: center detection

The  rst step towards creating a neural network able to detect the center was to create a suitable data set for network training. This involved manual annotation of the embryo center.

[1] Throughout the manuscript we will use the term \embryo" as a collective reference name for all the development phases of the zygote.

An image used for annotation is 2-dimensional, so we describe the center of each embryo by its $(C_x; C_y)$ coordinates. Therefore, the $C_x$ and $C_y$ coordinates were used as metrics.

This was done for 13 embryos (available at the start of the project) out of the total 92. A subset of 30 images per sample was selected for annotation. The image subset was evenly (linearly) distributed across the entire time span of the video, because we wanted our model to generalize for all phases. In order to increase stable learning, image values were scaled between [0-1] scale using maximum normalization $I^0 = \frac{I}{\max(I)}$. This results in images with a similar intensity distribution.

The annotation was performed on the central (5th) z-slice, because this slice was found to be the most in focus over the entire video. The images were downsampled by a factor of eight, from 928 928 to 116 116 (see Figure 14a), which bene ted our computational e ciency (reducing computational resources) and batch quantity. The annotation set consisted of a total of 13 30 images, resulting in 390 $(C_x; C_y)$ positions.

The second step is to choose the right neural network architecture to solve this embryo localization problem. An U-Net network would be a great t for this task: not only does it supports an image in/-out approach, but it also has the capability to extract both high- and low-level features through the encoder-decoder architecture, allowing for accurate and precise image-to-image mapping [47]. This approach requires the same input size as the output size. Hence, for each 116 116 input, an output array of 116 116 had to be de ned. Intuitively, the output consisted of a 116 116 black grid, with a single white dot marker in the center. These black and white colors simply correspond to 0 and 1, respectively; see Figure 14b.

(a) (b) (c)

Figure 14: Process of annotating an embryo image. A) The 116 116 embryo image B) The manually annotated center of the embryo image, where the center is represented by a single white pixel. C) The nal result of the annotation of an embryo image. The single white pixel is replaced by the radial gradient circular object.

This single pixel is expected to denote the exact center location. However, it is known that the manual annotation can contain human error(s), meaning that the perfect center location is not always valid. This introduces potential noise and the risk of over tting into the training process

To overcome this issue, a di erent method is chosen for the **116** output array. Instead

of using a single pixel to indicate the center, a radial gradient circular is used around the annotated spot. The radial gradient circular has a radius of 12 pixels. The radial gradient circular uses the center as the brightest spot assigning it an intensity of 1.0, see Figure 15. The brightness values get smaller as you move from the center to the outer layers of the circle, resulting in a radial gradient circular with the brightest spot in the middle, see Appendix A.1.

Figure 15: Representation of the radial gradient circular annotation. From left to right (1) Example of a 116 116 annotation. (2) All 116 vertical lines were plotted. (3) All 116 horizontal lines were plotted.

Occasionally, it occurred that some frames from the acquired mp4 le were corrupted or that there was no embryo present in the well. It would be bene cial to prevent these frames from entering our training data set. The task of detecting these frames could also be learned by the U-Net, each time the network sees a corrupted image it should predict an empty 116 116 array. Manually labelled samples representing such cases were added to the training set.

### 6.2.3 U-Net: training parameter optimization

Like for every other neural network, we need to set several important parameters for the U-Net, such as the loss function, optimizer, learning rate, kernel size, number of CNN lters and batch size. Adam optimizer was used, with the binary cross-entropy loss. The 116x116 output layer used Sigmoid as the activation function. Hyperparameter analysis was used to nd the optimal values for the remaining parameters. Multiple numbers of batch sizes were tested, consisting of [4,8,16,32,64]. The learning rate was set to $10^{3}$ and $10^{4}$.

Table 4: Hypermarameters used for training the U-Net.

| Hyperparameter | Value |
|---|---|
| Epochs | 12 |
| CNN kernel size | 2, 3 |
| Batch size | 4, 8, 16, 32, 64 |
| Learn rate | $10^{3}$, $10^{4}$ |
| Loss function | BCE |
| Optimizer | Adam |
| U-Net | m = 32 : [32, 64, 128, 64, 32] |
| Feature map | m = 64 : [64, 128, 256, 128, 64] |
| Con guration | m = 128 : [128, 256, 512, 256, 128] |

Next, we will describe the changes that have been made to the standard U-Net. Originally, an U-Net contains 4 layer with skip connections. The choice was made to reduce the number of encoder-decoder connections to 2. This smaller U-Net could be more e cient and give more room for experimenting with di erent amounts of feature maps. During training the input is processed through the encode-decoder. The number of feature maps m created across the network varies according to the depth of the U-Net. Which can be seen in Equation 2:

$$\begin{array}{ccccc} m & & & & m \\ & 2m & & 2m & \\ & & 4m & & \end{array} \qquad (2)$$

$$m! \quad 2m! \quad 4m! \quad 2m! \quad m$$

Meaning that after each max-pooling the number of feature maps doubles and after every transposed convolution the number of feature maps halves. The feature maps baseline was set to m = 32, which means a feature maps distribution of:

$$32! \quad 64! \quad 128! \quad 64! \quad 32$$

We have also experimented with 2 di erent feature maps sets: m = 64 and m = 128. An example of the adapted U-Net with m = 32 can be seen below in Figure 16. Table 4 shows all the U-Net hyperparameters.

Figure 16: The U-Net architecture used for the embryo center prediction task. The network takes a 116 116 embryo image as input and outputs a 116116 image indicating the embryo center in the form of a radial gradient circular. The initial number of feature maps for this U-Net was set tom = 32. To obtain the output, the input goes to the encoder part (left), the bottleneck (middle) and the decoder part (right). In the encoder (1) the input is converted into 32 feature maps and halved in size by max-pooling(orange), (2) this is repeated with 64 feature maps. The 2929 maps are passed through the bottleneck layers, resulting in 128 feature maps. In the decoder, the bottleneck output is up-sampled using transposed convolutions and concatenated through the encoder-decoder skip connections (blue arrows). This produces 64 feature maps, which are then subjected to the same process to produce 32 feature maps. Finally, the feature maps are convolved into a single 116 116 feature map.

### 6.2.4 Obtaining embryo location from U-Net output

The next task is to translate the image array output from the U-Net to an $(C_x, C_y)$ center location. We use multiple techniques to do this. The most straightforward approach is to nd the pixel with the highest value and select it as the center, since the brightest spot is expected to be in the middle. However, after a short evaluation the consistency of this method fell short.

Instead, we opted for a connected component analysis (CCA) approach, which is widely used in the image processing eld. This object recognition technique makes it possible to easily identify objects in an image. More precisely, the U-Net output should contain a single radial gradient circular object, which can be converted to a binary object by thresholding.

The CCA method involves scanning the binary image with an 8-connected square neighborhood kernel. To identify groups of connected pixels, each pixel hit will result in a pixel being assigned to a group. This allows for accurate labelling in an image. Before applying the CCA,

each prediction image is converted to a binary image, using an image threshold. Intuitively, by looking at the gure in Appendix A.1, one would attempt to intensity threshold on positive numbers $i > 0$. However, this would only output positive values, because a Sigmoid function outputs between 0 and 1, but never exactly 0.00 or 1.00. This means that all values will be above 0, and therefore positive. Instead, a rather low threshold of $10^{-4}$ was chosen, see Eq 3.

$$
\begin{aligned}
t &= 10^{-4}; \\
B^t(x) &= H(x - t);
\end{aligned}
\tag{3}
$$

Here $H(x)$ is the Heaviside step function. It is expected that the binary image contains groups of pixels at the embryo center point. However, it could also be possible that with a $10^{-4}$ threshold other groups of pixels could be classi ed as an object. To ensure that we have collected the correct object, an assumption is made, which indicates that the object with the highest surface area is most likely to be the center object, and the smaller objects are artifacts of the U-Net prediction.

The next step is to obtain the $(C_x; C_y)$ center coordinate from the selected connected component. This is done by placing a bounding box around the object. The center is determined by taking the mean coordinates of the horizontal and vertical sides, these 2 coordinates represent the $C_x$ and $C_y$ coordinates of the $116 \times 116$ image. The nal step was a simple conversion back to the original image size of $928 \times 928$, which required multiplication of the coordinates by a factor of 8.

### 6.2.5 Embryo extraction & Coordinates smoothing

After the center coordinates were obtained, the embryos could be cut out of the raw $928 \times 928$ image. A $400 \times 400$ box, de ning the region of interest, was placed at the obtained $(C_x, C_y)$ coordinate. This region was cropped out, resulting in a square $400 \times 400$ embryo image. The whole process is illustrated in Figure 17.

Embryos where the center appeared close to the border of the well could not be captured completely in a $400 \times 400$ frame. The constant $400 \times 400$ image size was necessary for training a network, because a neural network assumes a  xed size for the input layer. The replicateborder function from the openCV library was used to extend the border of an image to the required $400 \times 400$. This function replicates the last horizontal/vertical element until the desired resolution is reached. This process is shown in Appendix A.2.

Figure 17: The process for obtaining the cropped images from the original one. This process is illustrated by 8 images, from the left to the right. It includes (1{4) the input, the U-Net prediction, the binary thresholding, object detection using CCA and determination of the center of the object. Next (7{8), a 400 400 box is placed on the original 928928 embryo. Finally, the embryo image is cropped.

Another important information that can be derived from the U-Net prediction is the presence or absence of an embryo in the well. The decision is made by taking the sum of the prediction map: if this value is lower than the threshold $10^4$ then it is con dently classi ed as no embryo present, see Eq 4.

$$\begin{aligned} t &= 10^4 \\ P(\hat{o}) &= \frac{1}{n} \sum_{i=1}^{n} \hat{o}_i; \\ E^t(P) &= H(\hat{o} - t): \end{aligned} \tag{4}$$

When no embryo is detected or the image is corrupted, this frame is replaced by the previous one; e.g. if frame 113 does not contain an embryo, it is replaced by frame 112, which means that frames 112 and 113 will contain the same information.

### 6.2.6 Coordinates smoothing

During time-lapse video examination, a small jerking e ect was observed between 2 consecutive time points, which was a side e ect of human (annotation) error, AI error and scaling of the coordinates. For example, if two consecutive $C_x$ coordinates di er by only one pixel, with values 50 and 51, then rescaling will result in a shift of 8 pixels shift: 400 and 408, respectively.

In order to smooth the time series, the $C_x$ and $C_y$ values are smoothed separately over the entire time span using the Locally Weighted Scatterplot Smoothing (LOWESS) function [49]. The bene t of LOWESS is that it can handle outliers quite well because of the underlying

weighted least squares approach. The LOWESS method makes use of a sliding window, which slides from time-point 0 to the end of a sequence. The sliding window width is a fraction of the total sequence length. Here this fraction was set to 0.02.

This sliding window extracts a local subset. A polynomial is tted to this local subset using a weighted linear least squares regression. The weight function applied to this segment is the tricube function, given in Eq 5 and Figure 18.

$$w(d) = \begin{cases} (1 - |d|^3)^3; & \text{if } |d| < 1 \\ 0; & \text{otherwise} \end{cases} \tag{5}$$

This polynomial ensures that the focal point ($d = (0)$) carries the most weight, and the furthest point carries the least weight. Each point in the window has a certain weight. The closer a point is to the focal point the more weight it has. In other words, a point with a closer distance d to the focal point, has more importance, see Figure 18.

Figure 18: The tricube weight function: $w(d) = (1 - |d|^3)^3$
Distances d closer to the center carry more weight than distances further away.

The aforementioned process can be repeated, in our case we did it 3 times for getting a nice smoothing e ect. As result, the $C_x$ and $C_y$ coordinates are smoothed, which improves the cropping process.

## 6.3 Detection of the sharpest focal plane

In Section 6.2 the shortcoming of limited virtual memory was addressed. It is also necessary to address this shortcoming when processing the multiple z-slices for each image. Each image contains 7{11 z-slices, making the task more memory intensive. As mentioned earlier in Section 6.2.2, not all slices have valuable information. It is imperative that stacks with low information content are removed from the data set. The rst step for tackling this problem is to determine which stacks contain valuable information and which do not. Images with more detail and less blur are considered more useful, for this reason we used sharpness as a measure. The approach we used to obtain a sharpness rating was a traditional gradient method, discussed in Section 6.3.1. Other approaches to train a custom neural network capable of assigning a 0.0 to 1.0 sharpness rating proved to be unstable and were therefore rejected.

### 6.3.1 The average gradient magnitude as sharpness rating

The average gradient magnitude (AGM) was used to obtain the sharpness rating, given in Eq 6. The gradients is the measure of change in an image across $x$ (horizontal) and $y$ (vertical) dimensions. The image gradient is de ned by the  rst derivatives with respect to the $x$ ($\frac{@f}{@x}$) and $y$ ($\frac{@f}{@y}$) coordinates, see Figure 19. The image magnitude $jjr\ Fjj$ represents the strength of intensity change in the whole image. Large intensity changes often correspond to edges in the image. The magnitude vector is calculated as the square root of the sum of the squares of the $x$- and $y$-component gradients $jjr\ Fjj = \sqrt{(\frac{@f}{@x})^2 + (\frac{@f}{@y})^2}$. In order to get the AGM, the magnitude is divided by the number of all magnitudes. This AGM value corresponds to the overall sharpness in our image.

$$jjr\ Fjj = \sqrt{(\frac{@f}{@x})^2 + (\frac{@f}{@y})^2}$$

$$AGM = \frac{1}{n}\sum_{i=1}^{n} jjr\ Fjj \tag{6}$$

Figure 19: Overview of the gradient calculation on the embryo image. From left to right: the cropped embryo image, the gradients with respect to the horizontal direction, the gradients with respect to the vertical direction. The average gradient magnitude of this image is 3.14.

### 6.3.2 Sharpness selection

After obtaining all the image sharpness scores, the distribution over focal planes is analyzed. This gives an insight into how the sharpness values di er between the dimensions. In Figure 20 the distribution of the AGM is shown for a small time interval. Initially, it was expected that the same single focal plane would dominate throughout the whole time-series, because the same focus settings were set. This would allow for a very simple selection of the sharpest frame across the time-series. However, evaluation showed that slice shifts do occur regularly and have to be taken into consideration during this selection.

The intention was to design a data set with the highest possible information density possible, as sharper images contain more reliable information. An obvious step would be to select only the sharpest slice in each time step. This will result in the information reduction from 7{11 slices to a single one for each time frame. The disadvantage of this approach is that in this case we will loose the unique depth information, because only one image slice is picked. Hence, the top 3 sharpest slices are selected for each time.

Figure 20: The AGM for the sample 5313 for the time inverval 642{654. The x-axis represents the 9 focal planes. The y-axis represents the time frames. The lighter color indicates a higher AGM. It appears that the 9th slice is in focus throughout this local fragment as the corresponding values are all above 3. These values gradually decrease as we move away from the 9th focal point.

Figure 21: The before (left) and after (right) results of the top-3 sharpness normalization for the sample 5313. The x-axis represents the 9 focal planes. The y-axis represents the time frames. The top-3 frames in-focus are the focal planes 7, 8 and 9. With 3 irregularities found at frames 642, 648 and 654. These irregularities are resolved after normalization, which is shown by the constant coloring of the top 3 frames.

## 6.4 Sharpness normalization

In order to be used for training a network, the data should not differ too much within the training set, in particular, via unrealistic focus shifts. We want all our time frames to be sampled similarly. It is expected that small frame shifts will occur rarely. Figure 21 shows an unrealistic shift at frame 648.

To deal with small and large outliers, outlier detection was used to identify unrealistic focus shifts in the top 3. A data smoothing approach was implemented to smooth out the irregularities. A frame is classified as an irregularity if the top 3 focal planes are not adjacent (contiguous) to each other, e.g. frame 648 with a top 3 being slices [5,8,9] in Figure 21.

A sliding window approach was used for smoothing. This sliding window technique is used to process the data in a sequential manner. It iterates starting from $T_{n=0}$ until the end of the time sequence, and at each iteration it performs an operation that results in a normalized top 3 selection, see Figure 21.

The slider size was set to 5 in order to examine adjacent time frames from $T_{n-2}$ to $T_{n+2}$. The top 3 of these frames are used to determine $T_n$. The top 3 combination that occurs most frequently in $T_{n-2}$ to $T_{n+2}$ was selected.

## 6.5 Data sets for classi cation

Once we have obtained the sharpest images of the embryo, we can move on to the  nal step in the data pre-processing work ow. This stage involves shaping the data for training. The data is 4-dimensional, with each embryo characterized by the number of time points (1550), focal planes (3) and image size. As a result, the array for all embryos has the shape of (92, 1550, 3, 400, 400). There are several problems with this setup.

1. First, not all videos have the same length. Therefore, the number of time frames is sometimes smaller than 1550. As a result of this inconsistency, empty frames must be added to  ll up this space, because a neural network input layer expects the same shape as input, resulting in a higher likelihood of over tting (due to the neural network simply memorizing empty frames).

2. Secondly, it would be impossible to train a batch with the current resources, as it would not  t into the VRAM memory.

Based on the aforementioned arguments, maintaining the same data structure is not feasible. This raises the question of which structure would be most appropriate for our task. We need to consider the following factors when making this decision:

1. Di erent ages of the samples:

    - An age threshold must be set so that all the samples in the data set have the same length. Only samples of the same shape can be processed by a neural network.

2. Selecting an evenly spaced subset of frames for each sample, as discussed in Section 6.2.2. This process involves the following considerations:

    - Determining the optimal number of time frames for the samples.
    - Accounting for embryo motion, as removing too much data could leave out the important context of a frame.
    - Investigating the possibility of using only the sharpest slice instead of the 3 sharpest slices.

3. Downsizing the (400400) frame to (224 224).

The conclusion from this is that it is not possible to create the right data set in advance, as there are multiple considerations to be taken into account, and there is no one-size- ts-all solution. This is because it is not known a priori which characteristics will work best. It would be naive to expect only one data set will be the best one.

In order to  nd out which data set performs best, exploration must be carried out. The  rst step is to set an age/frame threshold. The shorter samples have around 800 frames. So it would make sense to set the threshold to 800. This measure reduces the data structure from (92, 1550, 3, 400, 400) to (92, 800, 3, 400, 400). Since, not all 800 frames can be used, the subset is linearly subsampled, for this di erent lengths of 10, 50 and 75 are chosen, referred to as $T_p$ .

The disadvantage of this approach is that it removes morphological data, which includes movements before and after the selected frames. To account for this, a data set is created that also includes the 2 frames before and after the snapshot, denoted as +2 and -2, respectively. In this case the context is set to $T_{c=5}$. In Figure 22, an example of a subset with 5 temporal samples and a previous/next context is shown.

We also created data sets with only the single sharpest focal plane, instead of the top 3. In this case the dimension $T_d$ of each $T_p$ in the data set is set to 1. For example, the data set with ($T_{p=50}$ ; $T_{c=1}$ ; $T_{d=1}$) would have the data structure of (92, 50, 1, 400, 400).

Finally, it was decided to downsample the 400 400 frame to 224 224. As a result, the samples take up less VRAM memory, allowing for larger batch sizes. It also speeds up the training process and it could also prevent over tting. The VGG and E cientNet models were also trained with this ratio, which could potentially lead to better transfer learning.

Figure 22: Example of the data representation for a training set including 5 time points with a context of +/- 1 ( $T_{p=5}$ ; $T_{c=3}$ ). Five equally-spaced frames are selected between 0-800, resulting in frames [0, 200, 400, 600, 800]. For each frame, the previous/next frames are also selected, except for the frames at the ends of the time spectrum. The time is represented in HH:MM format.

### 6.5.1 Image transformations and augmentations

One of the major factors when training a neural network is over tting, which is more likely to occur with small data sets, such as ours with only 92 samples. Data augmentation can be applied to the images to create additional variations in the images. This can help the model to learn the patterns and features in the images, rather than learning simple features. Two image transformation approaches are proposed, namely the ColorJitter and random pixel Shu e methods:

The rst approach uses the colorjitter function of the pytorch library to adjust the brightness and contrast of the images. For each training epoch, the brightness and contrast are selected uniformly between a factor of 0.1 and 1.6. Additionally, a random horizontal and vertical ip is added with a probability of 0.5 to ensure that the network does not encounter the same image twice.

The second method involves the use of a random pixel Shu e function, which shu es all the pixel values in an embryo image, resulting in the loss of the structural features while preserving the pixel intensity distribution, see Figure 23. This is done once before the training started. This approach is applied to verify if the models are actually learning the embryo structure and not simply relying on simple features like pixel distribution.

(a) (b)

(c) (d)

Figure 23: Comparison between the original and ColorJitter/Shu e data set.

# 7 Training the proposed models

In this section, we will discuss the neural networks for tackling the embryo implantation classification task. In total, 3 different types of network architectures are proposed:

1. The TimeBlocks network, a network composed of mini-networks dedicated to learning different time phases.
2. Adapted versions of the VGG network.
3. Adapted versions of the EfficientNet.

The TimeBlocks network was built from scratch, the existing VGG and EfficientNet networks required modifications for both input and output layers to fit our 4-dimensional data set. Section 7.1 provides a detailed overview of the models.

The proposed models for implantation potential classification models are trained using an identical approach. The data set is split into a training and test set with a split value of 0.8, resulting in a training set of 73 samples, and a test set of 19 samples. The ratio of (0, 1) labels for the whole data set is (63%, 37%). This ratio was also set within our training and test sets. This results in a distribution for $D_{train}$ of (46; 27) and for $D_{test}$ of (12; 7), both equal to (63%; 37%).

Furthermore, parameters were initialised as follows. The networks were trained for 200 epochs. The learning rate was set to $10^{-4}$, and a batch size of 16 was used. Adam and SGD were used as the optimizers, and binary cross-entropy loss was used to calculate the loss. The implantation potential was determined using the Sigmoid activation function. Table 6 contains all the general parameters used to train the models.

## 7.1 Proposed models

### 7.1.1 TimeBlocks network design

The TimeBlocks network is a network specifically designed to classify time series using a multi-branch structure. In this multi-branch structure, not all time series images are processed by the same convolutional layers, but are instead divided into a number of branches. Each branch presents its own mini-network for a specific subset of time series frames. This mini-network consists of a set of convolutional and linear layers. The outputs of these mini-networks are concatenated and passed through the output layers to produce the final classification.

This is different from a single-branch network like VGG, which processes the input data through a sequence of layers, where all the layers are connected to each other. The mini-networks perform independently of each other, and are therefore able to capture certain time features of the data set. The structure of a mini-block is as follows:

1. A subset of (batch normalized) $T_p \quad 224 \quad 224$ is assigned to a mini-block.
2. Multiple combinations of convolutions, ReLU activation and max pooling are applied to the $T_p \quad 224 \quad 224$ input, resulting in multiple feature maps of size $7 \quad 7$.
3. The feature maps are then flattened, and connected to a 128-vector, representing the subset of frames.

All mini-network's vectors are concatenated to form a large layer, where the classification takes place. In order to avoid overfitting and memory issues, it was not possible for each time frame to have an individual mini-network. For this reason, we experimented with 5, 10 and 20 numbers of mini-networks $T_b$. In total, the mini-network input undergoes 5 convolutions and max pooling.

The size of the mini-networks in the TimeBlocks network depends on the convolution filter configuration $T_f$. Two configurations were tested in the experiments: $T_{f=8}$ and $T_{f=32}$. For a mini-network with $T_{f=8}$ the 5 convolutions consist of [8, 16, 16, 32 and 32] filters and for $T_{f=32}$ of [32, 64, 64, 128, 128] filters.

The concatenation layer had a size of $T_b \quad 128$, where $T_b$ is the number of mini-networks. This layer connects to the final layer, which contains a sigmoid function and a single node. All custom parameters can be seen in Table 9. The whole network can be seen in Figure 24.

Optionally, the $T_p \quad 224 \quad 224$ could be batch normalized before passing a subset to the branches, during training 2 configurations were used: with batch normalization $T_{bn=True}$ and without $T_{bn=False}$.

Figure 24: The di erent parts of the TimeBlocks network, from top to bottom. First, the time frames $T_p$ are assigned to time blocks $T_b$, which in this example were both initialized to $T_{p=5}$ and $T_{b=5}$. Optionally, a batch normalization is performed for all inputs. Second, each time frame is processed by a dedicated Timeblock, which functions as a 'mini-networks'. Through 5 iterations of convolution and max-pooling layers, the 224 224 input is transformed into 128 feature maps of 77. The number of feature maps generated per iteration depends on the $T_f$ selected. In this gure the $T_f$ is set to 32. The feature maps are then attened and connected to a 128-vector. Next, the 5 'mini-networks' are concatenated into a layer of size $T_b$ 128. Finally, this layer is connected to the nal classi cation layer, which contains a sigmoid node to determine the implantation potential.

### 7.1.2 VGG-16 and EfficientNet

In order to use the VGG and EfficientNet for our task, 2 major modifications are made. First, the input layer is replaced to handle $T_p$ channels instead of the standard 3 color channels. Optionally, an optional batch normalization was placed at the start of the network.

Second, the classification layer of these models is made to classify for 1000 classes, whereas our goal is to get a 0{1 value from just a single node. Therefore, we had to modify the output layer to fit our requirements. Apart from these 2 changes, the core of the model remains the same.

The VGG16 network consists of 16 layers. The 16th layer is removed and replaced by a single node layer. This layer is connected to the 15th layer, which consists of 512 values obtained by 2D adaptive average pooling of 7x7 feature maps. The adapted VGG16 network is shown in Figure 25.

The EfficientNet network has 20 layers. The 20th layer is removed, and replaced by a single node layer. This layer is connected to the 19th layer, which consists of 1280 values, obtained by the 2D adaptive average pooling of 7x7 feature maps. The adapted network can be seen in Figure 26. In contrast to the different The model includes a dropout layer with a probability of 0.2. It is also the only model that implements the dropout layer.

Figure 25: The different parts of the adapted VGG16 network. The adaptations are shown in green and the original part is shown in yellow. The first adapted part is the input layer, where the input layer is changed to procces (224, 224, $T_p$) to (224, 224, 64) feature maps. Optionally, a batch normalization is performed for all inputs. The layers 2{15 (yellow) of the VGG16 remain unchanged. The second adapted part is the classification layer, the 4096 nodes from the 15th linear layer are connected to the classification layer, which contains a sigmoid node to determine the implantation potential.

Figure 26: The di erent parts of the adaptedE cientNet network. The adaptations are shown in green and the original part is shown in yellow. The rst adapted part is the input layer, where the input layer is changed to procces (224, 224, $4T_p$) to (224, 224, 32) feature maps. Optionally, a batch normalization is performed for all inputs. The layers 2{19 (yellow) of theE cientNet remain unchanged. The second adapted part is the classi cation layer, the 1280 nodes from the 19th linear layer are connected to the classi cation layer, which contains a sigmoid node to determine the implantation potential.

## 7.2 Hyperparameter tuning overview

This section includes all the training hyperparameters discussed previously, which are presented in Tables 5 to 9. The Table 5 provides hyperparameter for the di erent data sets. Table 6 contains the general training parameters. The Tables 7{ 9 show the parameter speci c to E cientnet , VGG and TimeBlocks

Table 5: Dataset hyperparameters.

| Hyperparameter | Value |
|---|---|
| Time Splits $T_p$ | 10, 50, 75 |
| Number of Dimensions $T_d$ | 1, 3 |
| Frame context $T_c$ | 1, 5 |
| Data Transformations | Default, Colorjitter, Random Pixel Shu e |

Table 6: General training hyperparameters.

| Hyperparameter | Value |
|---|---|
| Epochs | 200 |
| Batchsize | 16 |
| Learn rate | $10^{4}$ |
| Loss function | BCE |
| Optimizer | Adam, SGD |
| Batch normalization $T_{bn}$ | True, False |
| Train/Test split | 0.8 |

Table 7: E cient network hyperparameters.

| Hyperparameter | Value |
|---|---|
| Pretrained | True |
| Image size | 224x224 |

Table 8: VGG network hyperparameters.

| Hyperparameter | Value |
|---|---|
| VGG | 16 |
| Pretrained | True |
| Image size | 224x224 |

Table 9: TimeBlocks network hyperparameters.

| Hyperparameter | Value |
|---|---|
| Time Block $T_b$ | 5, 10, 20 |
| Time Block Filter con guration $T_f$ | 8, 32 |
| Image size | 224x224 |

# 8 Results

In this section, the pre-processing and performances of the classi cation models are presented in 3 parts. First the U-Net model performance on the embryo center detection task is presented. Next, the embryo's trajectory smoothing and detection of the most informative focal planes are presented. In the third part the model's performance for predicting the implantation potential is reported.

## 8.1 Model performance: embryo center detection

The  rst step involved creating models to detect the location of an embryo. This was done by subjecting the U-Net models to hyperparameter tuning to determine the most optimal combination. In total, 48 models were trained and evaluated based on their performance on the training and test set, with the binary cross entropy (BCE) loss used as a metric. The hyperparameters that were adjusted during the experiment were the batch size, learning rate, kernel size and the U-Net feature map con gurations.

Overall, 44 of 48 models converged within the 12 epochs achieving a BCE loss lower than 0.02. Interestingly, the remaining 4 models, which had higher losses between 0.06 and 0.12, all used the same learning rate of 1e-4, and had a low feature map con guration $m = 32$.

Out of the 48 models, 40 achieved a loss lower than 0.010. Speci cally, 23 models had a loss of 0.005, 11 had a loss of 0.006, and the remaining models fell within the range of 0.007 to 0.010. The hyperparameter causing the largest negative impact on the loss was the batch size, with larger batches resulting in higher losses; although this was only observed when larger batches were combined with a low feature map con guration $m = 32$.

Zooming in on these 40 best performing models, we see no major di erences for feature map con gurations, only that $m = 128$ always performed with a consistent 0.005 loss. The in uence of the kernel size, had no impact. Models using kernel sizes 2 and 3 performed similarly, see Figure 27.

Minor di erences were observed between the learning rates. However the 8 worst performing models used a learning rate of 1e-4, which may give a distorted picture, because the better performing models also used a learning rate of 1e-4.

In general, there were no major di erences in the training and test losses across di erent hyperparameter combinations. Most models had similar performances, with no single model standing out as the best. Therefore, the decision was made to select the optimal model based on the average di erence between training and test loss throughout the training process. The model with the least di erence of only 1e-3 between test and training loss was used with a batch size of 16, a learning rate of 1e-4, feature map con guration $m = 64$, and a kernel size of 2, see Figure 28.

Figure 27: This  gure compares the performance of an U-Net model across di erent hyperparameters using training and test loss metrics. The 4 hyperparameters are: batch size, number of feature maps, learning rate, and kernel size. The bars correspond to the loss after the 12th epoch, and the black lines above and below each bar represent the upper and lower bounds of a 95% con dence interval. The upper row shows the performances for each hyperparameter setting. The lower row represents all 48 models. In general, the models performed similarly when using the same settings. The parameters show to be more favourable for low batch sizes and learning rates. Notable is that the U-Net models with a feature map con guration of m = 128 have consistently led to low losses.

Figure 28: The model with the smallest di erence between its training and test losses. This model has the following settings: batch size of 16, feature map con guration of = 64, a learning rate of 1e-4, and a kernel size of 2. As we can see, the training loss values decrease from 0.176 to 0.009 and the test loss values decrease from 0.177 to 0.009 as the number of epochs increases. The model achieved a training loss of 0.009 and a testing loss of 0.009 after 12 epochs.

## 8.2 Trajectory estimation and smoothing

The second step involves obtaining the center coordinates of the embryo, by performing binary thresholding and connected component analysis (CCA) on the U-Net network outputs. Overall, this approach was able to determine the location of the embryo successfully for all samples. Also, it was able to detect abrupt embryo changes. However, further investigation of the embryo's trajectory revealed a small jerking e ect in the embryo's movements. In a small fraction of samples, outliers were found, appearing as abrupt spikes in embryo movement. To address this issue, the LOWESS smoothing function was applied to smooth out the trajectory movement over the time-span. This step was able to correct for coordinates appearing as outliers. The correspoding results can be found for all 92 samples in Appendex A.4.

The predicted and smoothed coordinates are illustrated in Figure 29. This  gure represents what the majority of our data set looks like. The embryo trajectory is tracked over the whole time range, and non-embryonic images were removed. The LOWESS function successfully smoothed out the jerking e ect while still preserving the overall trajectory. Each sample in Figure 29 has the following speci cs:

1. LD74_well06/ 5705well03: Across (almost) the whole sequence the jerking can be seen, shown by the up and down peaks.

2. 4431well07: Outliers around frames 1125, 1250 and 1275 are corrected by smoothing.

3. 5476well03: Around frames 800 a gaps is observed, indicating that no embryo was detected during this time frame.

In a small part of the data set, extreme outliers were observed, as can be seen in Figure 30. These outliers mainly occurred when the sample age was beyond the embryo age threshold, which was set to 800. Figure 30 can be summarized by:

1. 5623_well05: Outliers around frames 825 and 900 are corrected through smoothing.
2. H2563_well06: Outliers around frame 1300 are corrected through smoothing.
3. 5607_well02: Outliers around frames 1225 and 1350 are corrected through smoothing.
4. H2549_well03: The trajectory predictions between frames 125 and 375 are very unstable, with abrupt positional changes observed for both the X and Y coordinates.

Figure 29: The embryo movement in the x and y directions. The horizontal axis represents the age of the embryo, and the vertical axis represents the coordinates in the 928x928 frame. The red vertical line shows the age threshold of 800. In general, the samples in this gure shows that the jerking e ect is corrected by the smoothing function.

Figure 30: The embryo center trajectory movement in the x and y dimensions. The horizontal axis represents the age of the embryo, and the vertical axis represents the coordinates in the 928x928 frame. The normal and smoothed horizontal trajectories are depicted by blue and brown colors respectively, while the vertical trajectories are represented by green and black colors. The red vertical line shows the age threshold of 800. In general, the samples in this  gure show that the jerking e ect is  xed by the smoothing function. However, inconsistencies can be observed in the center determination, for embryo images older than the age threshold.

## 8.3 Detection of the most informative focal planes

In the third step, a sharpness analysis was carried out on all the focal planes. This was done to obtain the 3 most informative focal planes throughout the video sequence. Altogether, it was noted that the sharpness distribution remained stable over extended time periods, with the 3 sharpest focal planes being located next to each other in 98.2% of time. Even after focus shifts the corresponding sequences remained stable. Sharpness consistency and focus shifts can be seen in Figure 31, for 4 samples H2547, 5850, 5607 and R1635. In the samples H2547, 5850, and R1635 small shifts were observed, in contrast to 5607, where over the whole sequence the 3 most informative focal planes distribution was stable.

Figure 31: The focal plane sharpness distribution over the time sequence. The horizontal axis represents the age of the embryo, and the vertical axis represents the focal planes. The 3 sharpest focal planes are depicted in green. In general, the samples in this gure show that the distribution remains stable, as indicated by the steady green line. In addition, there are noticeable shifts in focus, indicated by the abrupt green shifts. Focus shift time intervals can be seen for all samples around frames 575 (day 2.0) and 1150 (day 4.0).

The number of correctly classi ed time frames was 98.2%. Sharpness normalization was applied to correct outliers and smooth the trajectory for the remaining frames. In Figure 32, the e ect of sharpness normalization is shown for two types of samples.

On the left are the ones with minor outliers consisting of samples 5313 and 6010. In 5313 outliers between the frames 600{900 were found and corrected via normalization. In 6010 outliers can be found across most of the time-span, shown by the unstable top-3 pattern. For both these samples the distribution is more stable after the normalization.

On the right side are those with unstable sharpness distributions H936 and H2549. Sample H2549 has no clear focus positioning between frames 100 and 200. Sample H936 seems to be unique in the way that no stable top-3 is found through most of the sequence; meaning that the top-3 consists of non-adjacent focal planes, indicated by the red gap between the green part. For these samples the minor shifts were corrected by normalization, but the major shifts in H2549 in frames 100{200 could not be compensated.

Figure 32: The focal plane sharpness distribution over the time sequence, before and after normalization. The horizontal axis represents the age of the embryo, and the vertical axis represents the focal planes. The 3 sharpest focal planes are depicted in green. The samples all show some kind of irregularity, which is partly corrected by normalization.

### 8.3.1 Temporal sampling

Repeated patterns were found when looking at the average gradient magnitude (AGM) values over the time-span. These patterns repeat themselves after 5 frames. Generally, the pattern begins with the highest AGM value, after which the AGM values subsequently decrease with each frame until the last frame, after the cycle repeats itself. Frame 1, which contains the highest peak AGM value, is classi ed as the keyframe. For our videos, the key frame period is set to 5. In Figure 33, the key frame pattern over the time-span is shown. This  gure shows that this pattern persists throughout the entire time sequence.

Figure 33: The average gradient magnitude (AGM) distribution over the time sequence for the focal planes 1 (blue) and 5 (red). In the top plot, the horizontal axis represents the age of the embryo. The vertical axis represents the AGM value. Overall, it is evident that focal plane 5 is sharper throughout the whole sequence. The bottom plots zooms into the key frame pattern, with the vertical lines representing the key frames at the start of the repeating pattern that occurs every 5 frames. Focal plane 1 shows that the peak height varies between patterns for the 1100-1200 interval. Also, focus shifts are visible around frames 580, 800, 850 and 1150.

## 8.4 Neural network results

In this section the results of training all the neural networks are presented. This is done for the di erent network types: the TimeBlocks network, E cientNet and the VGG16 network. In total 960 models were trained with di erent parameters and data sets. All parameters are reported in Tables 5 to 9.

## 8.5 TimeBlocks network

The  rst type of model we trained was the TimeBlocks network. We trained a total of 672 models on the 3 di erent data sets: Default (224), ColorJitter (224), and Shu e (224).

### 8.5.1 TimeBlocks - optimizer & batchnorm in uence

The performance was mainly determined by 2 factors: (i) the type of optimizer used, Adam or SGD and (ii) the presence of a batch normalization (BN) layer. The following section describes the di erences in performance between these factors.

None of the models trained with SGD were able to learn on the training set: all models achieved a 63% accuracy. After further investigation, we found out that these models had a recall of 1.0 for label 0 and 0.0 for label 1, which explains the consistent 63% accuracy score.

On the other hand, the models trained with Adam and $T_{bn=True}$ performed well on the Default data set achieving an average training loss of 0.006 and a 100% accuracy after 100 epochs. In comparison to the models trained with Adam and $T_{bn=False}$ needed 200 epochs to reach a training loss of 0.065 0.84e-2, and had an accuracy of 98%, see Table 10.

Table 10: The mean TimeBlocks performance per data set after 200 epochs.

| Con gurations | Data sets | Training loss | SD | Test loss | SD | Training acc. | Test acc. |
|---|---|---|---|---|---|---|---|
| opt=Adam $T_{bn=True}$ | Default | 0.001 | 7.51e-4 | 4.27 | 2.08 | 100% | 44% 8 |
| | ColorJitter | 0.456 | 0.13 | 2.04 | 1.76 | 80% 8 | 43% 8 |
| | Shu e | 0.050 | 8.3e-2 | 1.43 | 0.57 | 99% 3 | 52% 11 |
| opt=Adam $T_{bn=False}$ | Default | 0.065 | 0.84e-2 | 2.67 | 1.69 | 98% 2 | 46% 11 |
| | ColorJitter | 0.660 | 1.26e-2 | 0.66 | 1.70e-3 | 63% 0 | 63% 0 |
| | Shu e | 0.643 | 2.87e-2 | 0.67 | 2.83e-2 | 64% 3 | 60% 6 |
| opt=SGD | Default | 0.672 | 2.57e-3 | 0.673 | 2.11e-2 | 63% 0 | 63% 0 |
| | ColorJitter | 0.674 | 1.75e-3 | 0.674 | 1.67e-4 | 63% 0 | 63% 0 |
| | Shu e | 0.670 | 4.09e-3 | 0.669 | 5.20e-3 | 63% 0 | 63% 0 |

In the case of the ColorJitter data set only the models trained with Adam and $T_{bn=True}$ were able to learn, albeit at a slower rate with a gradual decrease in training loss starting from the 50th epoch and  nally reaching 0.456 0.13 at the 200th epoch, with an accuracy of 80%. All other model con gurations trained on the ColorJitter data set achieved a mean accuracy of around 63%, with 1.0 recall for label 0 and 0.0 recall for label 1.

Results were similar for the Shu e data set, where the Adam and $T_{bn=True}$ models were the only ones able to reduce the training loss within 200 epochs. A training loss of 0.350 and a 99% accuracy were achieved.

When evaluating the performance of the models on the test data set, we found 2 outcomes. 1) Models that were capable of learning the training set had on average, higher test loss and lower test accuracy. 2) Models that were unable to learn had the same training and test loss of around 0.67 and similar accuracy of around 63%. All the training and test losses and accuracy are shown in Figure 34.

Another interesting nding was the recall classi cation performance, that can be seen in Figure 35. All models trained with Adam, all started the training with a recall of 1.0 for label 0 and a recall of 0.0 for label 1. During the training the model's the performance increases for label 1, while it remains stable for label 0, for the models capable of learning on the training set.

### 8.5.2 TimeBlocks: parameter in uence

This section investigates the in uence of the hyperparameters over the training process. These hyperparameters are: dimensionality ($T_d$), context ($T_c$) and the number of time points ($T_p$). In contrast to the VGG16 and E cientNet we have 2 additional parameters, which are the $T_f$ & $T_b$, indicating the size and the amount of time blocks used in the network.

In Figures 36{37 the training loss trajectory corresponding to each hyperparameter is shown. This was done only for the models able to learn, consisting of models trained with Adam & $T_{bn=True}$ for all the data sets. For the Default set we also include the models without $T_{bn}$. The greatest impact had the lter con guration $T_f$. Time blocks starting with $T_{f=32}$ lters, decreased faster in training loss than models with $T_f$ set to 8.

Remarkable is that the speed of achieving low training loss for $T_d$ and $T_c$ parameters was in the order[2] of $(T_{d=1}; T_{c=1}) \succ (T_{d=3}; T_{c=1}) \asymp (T_{d=3}; T_{c=1}) \prec (T_{d=3}; T_{c=5})$ for the models trained with the $T_{bn}$, the order was exactly the opposite.

The number of time points $T_p$ also had an in uence; having more time points ensuring faster learning. Models with $T_{p=10}$ converged slower compared to $T_{p=50 \wedge 75}$. This was not the case for models without $T_{bn}$.

For the number of time blocks used in the network, the di erence between $T_b$ was minimal for the Default and Shu e data sets. But for the ColorJitter, there was a di erence for $T_{b=20}$ with 0.55 training loss compared to 0.4 for $T_{b=5}$, after 200 epochs.

[2] Ordered by convergence speed: from slowest to fastest.

Figure 34: The average training performance of the TimeBlocks models over the training trajectory for the models trained with Adam and SGD. The error bar represents the standard deviation. In general, we see that the models trained with SGD (blue) did not improve over the training set. In contrast to the Adam models trained with $T_{bn=True}$ (red), these models were able to reduce the training loss for all data sets with average training losses of 0.001, 0.456 and 0.050 for the Default, ColorJitter and Shu e data sets. The models without batch normalization ($T_{bn=False}$) (blue) were only able to reduce the training loss on the Default data set by 0.065. Unfortunately, the test loss increased while the training loss decreased.

Figure 35: The average training recall performance of the TimeBlocks models for the models trained with Adam and SGD. In general we see that the models trained with Adam (row 1,2) always output a 0 at epoch 1, resulting in having a recall of 1.0 for this label. During training the models improve the recall of the label 0, while the recall of label 1 remains constant. For the SGD, we see that the recall between labels 0 and 1 is approximately equal at epoch 1 but converges quickly in favour of label 0.

(a) (b)

Figure 36: The average training performance of the $T_b$, $T_f$, $T_d$, $T_c$ and $T_p$ for the Adam-trained models with and without $T_{bn}$, on the Default data set. The filter configuration $T_f$ of each $T_b$ had the largest impact on training loss. Interestingly, the Adam models with and without $T_{bn}$ have opposite performance for $T_p$ and $T_d$, $T_c$; favouring larger data set for models with $T_{bn}$, but the opposite for models without $T_{bn}$.

(a) (b)

Figure 37: The average training performances of the $T_b$, $T_f$, $T_d$, $T_c$ and $T_p$ for the Adam trained models with $T_{bn}$ on the ColorJitter and Shu e data sets. For both the data set we see that a larger data set ensures a faster decrease in training loss, see the plots for $T_d$, $T_c$ and $T_p$. This is also the case for larger $T_f$ con gurations, models trained with $T_{f=32}$ decrease faster than models with $T_{f=8}$. Overall, the models trained on ColorJitter were able to achieve a loss of around 0.45 and the Shu e models were able to achieve a much lower loss of around 0.1.

## 8.6 VGG16

A total of 144 VGG16 models were trained on the 3 data sets: Default (48), ColorJitter (48) and Shu e (48). Many of these models achieved very low training losses, while some models were not able to learn at all. To gain more knowledge about the training performance, we noted the number of epochs where the training loss was below 0.35 for each model, which can be seen in Figure 38. These results are discussed in Section 8.6.1.

### 8.6.1 VGG16 - Training loss & Over tting

We start with the 48 models trained on the Default data set. In total, 24 models were trained with $T_{bn=True}$, all of these models reach 0.35 loss within less than 50 epochs, and all achieved accuracy of 100%. The models that were trained with Adam tend to over t twice as fast as those trained with SGD.

Interestingly, the models trained with $T_{bn=False}$ were less likely to achieve low losses, with only 10 out of 24 models achieving a loss of 0.35 after 200 epochs. The other 14 models all performed above the 64% accuracy, with an average accuracy of 80% on the training set.

In contrast to the models trained on the Default data set, performance on the ColorJitter data set was less uniform, with no models achieving a loss below 0.35 and having accuracy around 63%. Further investigation revealed that these models had a recall of 1.0 for label 0 and 0.0 for label 1, which explains the consistent 63% accuracy score. There were only a few exceptions with a training accuracy of at least 78%, which were trained with larger training sets: $T_{p=50\_75}$ with ($T_{d>1}$ _ $T_{c>1}$) using Adam and $T_{bn=True}$, see Table 11.

Table 11: Model parameters that achieved an accuracy of at least 78% for the ColorJitter data set.

| $T_p$ | $T_d$ | $T_c$ | Optimizer | $T_{bn}$ | Training Acc. |
|---|---|---|---|---|---|
| 75 | 1 | 5 | Adam | True | 90% |
| 75 | 3 | 1 | Adam | True | 82% |
| 75 | 3 | 5 | Adam | True | 78% |
| 50 | 1 | 5 | Adam | True | 86% |
| 50 | 3 | 5 | Adam | True | 82% |

The Shu e data set showed the same pattern as the Default data set, which means inability to learn, except for the models trained without $T_{bn}$, with an accuracy of 63%. The remaining $T_{bn}$ models all over t with a 100% accuracy. Compared to the models trained on the Default data set, the (Shu e) models took longer to reach a loss of 0.35.

Figure 38: The number of epochs required to achieve a training loss of less than 0.35. The horizontal axis represents the 48 models for the speci c data set, the vertical axis the number of epochs. The training accuracy is shown above each bar. The colour represents the number of temporal frames in a data set. The top, middle and bottom plots show the 48 models trained on the Default, ColorJitter and Shu e data sets. The Default models with batch normalization all over t within less than 50 epochs, with 100% accuracy. The models trained with Adam over t twice as fast as these trained with SGD. Models without $T_{bn}$ were less prone to 100% over tting. Only one of the ColorJitter model achieved a loss below 0.35, using 75 time points with a $T_{c=5}$ and a $T_{d=1}$. This Adam-trained model performed much better than the other 47 models. The batch normalized Shu e models were over tted with 100% accuracy. However, it took them longer to reach a loss of 0.35 compared to the models trained on the Default data set. The models with $T_{bn=False}$ had an accuracy of 63%.

### 8.6.2 VGG16: batch normalization and test loss

In this section we focus on the models that used $T_{bn=True}$, because these were able to achieve lower loss for the Default and Shu e data set. Figure 39 shows the average training and test accuracy and loss for the Default and Shu e data set.

The accuracy of the Default set shows that the models using the Adam optimizer generally converge faster to a 100% training accuracy. Speci cally, at the 18th epoch, with an accuracy of 99.1 0.013, while SGD had an accuracy of 99 0.005 at the 38th epoch.

However, Adam's training loss trajectory is less stable, with large standard deviations during the training process. In contrast, SGD shows a stable learning pattern. After the 60th epochs, the loss reaches nearly zero for both the Adam and SGD trained models. However, after 115 epochs the Adam models become unstable again.

In the Shu e data set we observed a similar trend where the SGD is more stable during training. However, it is noteworthy that Adam has unstable high training losses between the 58th and 84th epoch, with an average loss of 2.5 2. Another key di erence from the Default data set is that it takes longer to reach accuracy close to 100%. The model trained with Adam reaches an accuracy of 97.49 at the 124th epoch, while the SGD models reaches 1.96 at the 104th epoch.

The models performed the worst on the ColorJitter data set, with no evidence of the model learning. This resulted in a training loss of 0.66 0.017 after 200 epochs for the SGD trained models. Although the Adam optimizer demonstrated some learning ability, its performance was highly unstable. An overview of all model performances can be found in Appendices A.5 to A.8.

While some models show very promising results on the training set, this is not the case for the test set, with low training loss being proportional to a higher test loss. This can be seen in the Table 12.

Table 12: Performance of optimizer and $T_{bn}$ choice for all the data sets.

| Con gurations | Data sets | Training loss | SD | Test loss | SD | Training acc. | Test acc. |
|---|---|---|---|---|---|---|---|
| opt=Adam $T_{bn=True}$ | Default | 0.080 | 0.19 | 8.56 | 6.44 | 96 11 | 53 8 |
| | ColorJitter | 0.897 | 0.65 | 3.59 | 3.5 | 69 6 | 49 11 |
| | Shu e | 0.204 | 0.244 | 3.95 | 3.57 | 91 14 | 53 12 |
| opt=Adam $T_{bn=False}$ | Default | 3.44 | 10.56 | 6.48 | 10.4 | 78 12 | 53 10 |
| | ColorJitter | 0.656 | 5.71e-3 | 0.660 | 4.61e-3 | 63 0 | 63 0 |
| | Shu e | 0.659 | 1.92e-4 | 0.658 | 16.3e-4 | 63 0 | 63 0 |
| opt=SGD $T_{bn=True}$ | Default | 4.91e-4 | 1.07e-4 | 1.63 | 0.472 | 100 0 | 55 11 |
| | ColorJitter | 0.66 | 1.74e-2 | 0.68 | 6.31e-2 | 63 2 | 60 10 |
| | Shu e | 1.75e-2 | 3.21e-2 | 2.08 | 0.86 | 100 7.91e-3 | 48 14 |
| opt=SGD $T_{bn=False}$ | Default | 0.37 | 0.25 | 1.08 | 0.58 | 82 13 | 54 10 |
| | ColorJitter | 0.66 | 3.82e-3 | 0.66 | 6.24e-3 | 63 0 | 63 0 |
| | Shu e | 0.65 | 1.77e-2 | 0.67 | 1.99e-2 | 63 1 | 62 3.04e-2 |

Figure 39: The average training performance of the VGG16 models over the training trajectory for the models trained with batch normalization. The error bar represents the standard deviation. The accuracy and the training loss for the Default data set are shown on the left. All these models reached a training loss of 0.0 around the 80th epoch. The SGD optimised models converge slower compared to Adam. Interestingly, the training loss trajectory of Adam is less stable compared to SGD. Also, after reaching a zero loss, Adam becomes unstable again after the 115th epoch.
Performance of the Shu e set is shown on the right. These models take longer to reach an accuracy close to 100% compared to the Default data set. Remarkable, the Adam models had unrealistically high training losses between the 70th and 90th epoch. Also, the training loss was not able to reach zero. Overall, the test accuracy for all models varied between 40% and 60%.

### 8.6.3 VGG16: parameter in uence on the Default data set

This section is used to investigate the in uence of the hyperparameters during the training. Figure 40 shows the trajectory of the training loss between the hyperparameters $T_d$, $T_c$ and $T_p$. This was done only for the models that were able to learn on the Default data set.

When we examined the models trained with $T_{bn=True}$ and SGD, we found that the values of $T_c$ and $T_d$ impacted the speed of training. Models trained on more complex data sets learned more slowly. We observed that models trained with $(T_{d=1}; T_{c=1})$, $(T_{d=3}; T_{c=1})$, $(T_{d=1}; T_{c=5})$, $(T_{d=3}; T_{c=5})$ needed on average (36, 32, 24, 23) epochs to achieve a training loss of 0.35. The same trend was observed for the time points $T_p$: (10, 50, 75) needed (31, 28, 22) epochs to reach a training loss of 0.35.

For the SGD models trained with $T_{bn=False}$, we observed no di erences between the $T_c$ and $T_d$ con gurations, except for the $(T_{d=1}; T_{c=1})$ with a training loss of 0.043 after 200 epochs, compared to 0.413 for $(T_{d=1}; T_{c=5})$ and 0.51 for $(T_{d=3}; T_{c=1})$ and $(T_{d=3}; T_{c=5})$. The results for the $T_p$: (10, 50, 75) were (0.267, 0.419, 0.423) after 200 epochs.

The model trained with Adam did not show a favourable parameter con guration. All the results of this section are shown in Table 13.

Table 13: Parameter in uence for $T_c$, $T_d$ and $T_p$ for the models trained on the Default data set.

| Con guration | | | Results | |
|---|---|---|---|---|
| $T_{bn}$ | Optimizer | Parameters | Epoch | Training loss |
| True | SGD | $T_{d=1}; T_{c=1}$ | 36 | 0.35 |
| True | SGD | $T_{d=3}; T_{c=1}$ | 32 | 0.35 |
| True | SGD | $T_{d=1}; T_{c=5}$ | 24 | 0.35 |
| True | SGD | $T_{d=3}; T_{c=5}$ | 23 | 0.35 |
| True | SGD | $T_{p=10}$ | 31 | 0.35 |
| True | SGD | $T_{p=50}$ | 28 | 0.35 |
| True | SGD | $T_{p=75}$ | 22 | 0.35 |
| False | SGD | $T_{d=1}; T_{c=1}$ | 200 | 0.043 |
| False | SGD | $T_{d=1}; T_{c=5}$ | 200 | 0.41 |
| False | SGD | $T_{d=3}; T_{c=1\_5}$ | 200 | 0.51 |
| False | SGD | $T_{p=10}$ | 200 | 0.27 |
| False | SGD | $T_{p=50}$ | 200 | 0.42 |
| False | SGD | $T_{p=75}$ | 200 | 0.42 |

(a) (b) (c) (d)

Figure 40: The average training performance of the $T_d$, $T_c$ and $T_p$ for the models trained on the Default data set. On the left side we can see that larger data sets enable faster decrease in training loss for models with $T_{bn}$ and SGD. Interestingly, we see the opposite for the models trained without $T_{bn}$. Here we see that data set with $T_{(d=1}$; $T_{c=1})$ achieve lower losses. This is also the case for $T_p$. On the right side the Adam models are shown. All models with $T_{bn=True}$ achieved losses close to zero within 20 epochs and no di erences between the parameters con gurations could be found. The Adam models with $T_{bn=False}$ were unable to achieve loss close to zero again, no di erence between parameters con gurations could be found.

## 8.7 E cientNet

### 8.7.1 E cientNet: over tting

The last type of model we trained was the E cientNet. We trained a total of 144 models on 3 data sets: Default (48), ColorJitter (48), and Shu e (48). Many models reached a very low training loss. To gain more insights into the training performance, we noted the number of epochs where the training loss fell below 0.35 for each model, which can be seen in Figure 41.

We start with the models that were trained on the Default data set. All the models over- tted on the training set, reaching 100% accuracy at some point during training. How quickly this over tting was achieved depended on the optimizer used. The Adam optimizer was shown to achieve a training loss of less than 0.35 in around 15 epochs. The models trained with SGD achieved this loss within 50 to 120 epochs.

The SGD models were not able to learn on the ColorJitter data set, as none of them reached the 0.35 loss, with the average accuracy of 68%. On the other hand, the Adam models were able to learn with the average accuracy of 95%.

The pattern in the Shu e data set was similar to the Default data set, with all models over tting with 100% accuracy. The Adam models were also faster to reach the loss of 0.35. The Adam models trained on the larger data sets ($T_{p=75}$; $T_{c=5}$; $T_{d=1\_3}$) took more epochs to reach this loss.

### 8.7.2 E cientNet: optimizer in uence

The focus of this section is to explain the di erences in performance between Adam and SGD trained models. Figure 42 shows the accuracy and loss for the training and test set.

Starting with SGD, for the Default and Shu e data sets the training loss shows a stable trajectory. The models achieved a training loss of 0.15 0.05, and an accuracy of 99.2% for the Default data set. The models trained on the Shu e data set achieved a training loss of 0.11 0.07, and an accuracy of 100.7% after the 200 epochs. The models performed less well on the ColorJitter data set, with a training loss of 0.68 0.024, and an accuracy of 0.62 0.03.

In contrast to the mean SGD training loss, Adam was not stable with large standard deviation errors across the training trajectory. Despite this, the models trained with Adam achieved accuracies of 98.9%, 84 11%, 96 9% for the Default, Colorjitter and Shu e data set, respectively.

Figure 41: The number of epochs needed to achieve a training loss of less than 0.35 for the E cientNet models. The horizontal axis represents the 48 models for a given data set, the vertical axis the number of epochs. The training accuracy is shown on each bar. The colour represents the number of temporal frames in a data set. The top, middle and bottom plots show the 48 models trained on the Default, ColorJitter and Shu e data sets. Accuracy of 100% is achieved by all models trained on the Shu e and Default data set. These models also achieve a training loss of less than 0.35. Adam optimized models achieve a 0.35 loss faster compared to SGD. Interestingly, Adam models take longer to reach 0.35 loss, when trained on larger Shu e data sets. For the ColorJitter data set, only the models trained with Adam were able to learn with a mean accuracy of 95%.

(a) (b)

Figure 42: The mean optimizer performances of the E cientNet models over the training trajectory. The error bar represents the standard deviation. On the left the performance for the Adam-optimized models is shown, and on the right is the performance for the SGD-optimized models. The Adam-trained models show an unstable mean trajectory loss with high standard deviation peaks, especially for the model trained on the Default and Shu e data sets. The classi cation accuracy for these models are 98%, 84 11% and 96 9% for the Default, ColorJitter and Shu e data sets. The training loss of the SGD models converges slower than Adam, but SGD has lower standard deviation of 99 2%, 62 3% and 100 7% for the Default, ColorJitter and Shu e data sets, respectively. Unlike the training loss, both optimizers were unable to reduce the test loss.

### 8.7.3 E cientNet: parameter in uence

In order to examine the impact on the di erent parameters, the training loss of all possible con gurations is presented in Figure 43 (SGD) and Figure 44 (Adam).

First, we discuss the SGD models trained on the Default and Shu e data sets. All of these models reach a training loss of 0.35 within 200 epochs. There is a large di erence in learning stability between these two data sets. Models with $T_{c=5}$ have a less stable training loss curve compared to the models with $T_{c=1}$. The combination of $T_{c=5}$ with $T_{d=3}$ leads to even stronger instability. In particular, at late time points $T_{p=50}$ and $T_{p=75}$, the impact of a higher context and dimension make the training loss curve unstable.

On the ColorJitter data set, the network was unable to learn. The cumulative plot for these parameters is shown in Appendices A.10 and A.11. It is worth to note that the models trained on the Shu e data set start with a very high training loss in the rst few epochs, after which it returns to normal.

The models trained with Adam all had an unstable training loss pattern, except for the smallest data set $(T_{p=10}; T_{c=1}; T_{d=1})$. The larger the data set, the more unstable the loss curve was. Interestingly, the models trained on the ColorJitter data set had overall a more stable loss curve; mean losses around 0.25, 0.4, 0.5 were reached for $T_p$ of 10, 50, 75, respectively.

The stability of the models could be ordered from high to low $T_p$ as $= (10; 50; 75)$ and within a $T_p$ as $(T_{d=1}; T_{c=1}) > (T_{d=3}; T_{c=1}) \_ (T_{d=1}; T_{c=5}) \_ (T_{d=3}; T_{c=5})$. This shows that larger data sets result in more instability during training, with the model trained on the largest set $(T_{p=75}; T_{c=5}; T_{d=3})$ exhibiting the highest training instability.

The cumulative plot for these parameters can be found in Appendices A.9-A.11.

(a) (b) (c)

Figure 43: The training loss performance of the E cientNet models optimized with SGD for all the data set combinations. The models trained on the ColorJitter data set were not able to improve the training loss. The models trained on the Default data set were able to learn; in particular, $T_{p=10}$ (blue) reached a loss of around 0.20 in 200 epochs. The training became unstable as more dimensions were added to the data set. The stability could be ordered from high to low as: $T_{(d=1}; T_{c=1}); (T_{d=3}; T_{c=1}); (T_{d=1}; T_{c=5}); (T_{d=3}; T_{c=5})$. These observations also accounted for $T_{p=50}$ (orange) and $T_{p=75}$ (green). However, for these $T_p$ the training loss showed to be less smooth. Especially, for the larger data set $(T_{d=3}; T_{c=5})$, many instabilities were found. The Shu e data set had similar behaviour, with the exception of models $T_{p=75}$ with $(T_{d=1}; T_{c=5})$ _ $(T_{d=3}; T_{c=5})$, which showed slightly higher instabilities. Nonetheless, the models trained on the Shu e data set showed to reach even smaller losses around 0.1 after 200 epochs.

(a) (b) (c)

Figure 44: The training loss performance of the E cientNet models optimized with Adam for all the data set combinations. The models trained on the Default and Shu e data sets were able to reach losses close to zero very quickly. However, with the exception of $(T_{p=10}; T_{d=1}; T_{c=1})$, the training curve of the models was very unstable with large variations. The stability could be ordered from high to low as: $(T_p = 10; 50; 75)$ and within a $T_p$ as $(T_{d=1}; T_{c=1}) > (T_{d=3}; T_{c=1}) \_ (T_{d=1}; T_{c=5}) \_ (T_{d=3}; T_{c=5})$. Larger data set results in more instability during training. The models trained with ColorJitter are more stable, reaching losses below 0.25, 0.4, 0.5 for $T_p$ of 10; 50; 75, respectively; except for the models with $T_{(p=75\_50}, T_{d=3}; T_{c=5})$, which showed higher instabilities during training.

# 9 Discussion

In this study, videos of IVF embryos were used to train 960 machine learning models to investigate whether these models are able to predict the implantation potential. To achieve this, a work ow was developed to extract the most informative content from the videos. These data sets were then used to train the TimeBlocks, VGG16 and E ecientNet networks, among which the TimeBlocks network was speci cally designed to handle longitudinal data.

## 9.1 Pre-processing: work ow

The  rst part of the work ow consisted of cropping the embryo images using an embryo localization approach. The sub-research question posed was: RQ1: Is it possible to localize the position of an embryo in the frame? Training the U-Net with multiple con gurations showed that the U-Net was able to successfully predict the location of the embryo. In addition, the U-Net was also able to successfully detect the presence or absence of an embryo.

The U-Net was trained on 116×116 images downsampled from the original 928×928 resolution. This contributed to the model's generalization as it was only trained on a small subset of 390 images from 13 embryos. Based on Figure 27 we can conclude that creating 48 U-Net models might have been too excessive, as they all performed well; especially, the U-Net models trained with $m = 128$.

The LOWESS coordinate smoothing proved to be e ective, successfully correcting the jerking e ects. After cropping, the average gradient magnitude value was able to give a sharpness rating. Figure 31 provides convincing evidence that this approach is able to compute the sharpness distribution within a sample and can identify the associated focus shifts. What strengthens this argument is that the AGM results were able to detect the keyframe pattern.

## 9.2 Proposed models: over tting

When evaluating the TimeBlocks, VGG16 and E cientNet models, a common conclusion was drawn: models that were able to reduce the training loss performed worse on the test set. The over tting on the training set became more evident as all the 3 network types were also able to perform equally well on the Shu e data set. These  ndings proved that the models were unable to detect patterns between the time points, but instead relied on some form of memorization, like pixel counting. Generally, the models over tted faster on the Default data set, likely due the combination of multiple trivial features like embryo well placement.

Furthermore, the results revealed that label imbalance also plays a role in the likelihood of over tting. Based on Figure 35, we see that the models are mostly  ne-tuned for label 1 (pregnancy). This is not surprising, given that the training set contains only 27 samples for this label. Models trained on such small sets are prone to over tting and lack generalization.

The implementation of dropout layers could potentially prevent over tting and improve model's generalization. However, the dropout present in the E cientNet had no e ect, while this model, with only 5.3 million parameters, should be the most a ected by it. It is likely that, if the dropout would be implemented for VGG, it would be ine cient, due to its 138 million

parameters. Limiting the number of learnable parameters could also be an option to prevent over tting, but this can only be successful if the likelihood of learning these trivial features is reduced.

In addition, the "trivial features" argument could also be the reason that the models over t slower for the ColorJitter data set. In this data set, the images were augmented di erently for each iteration, thus discouraging the learning of "simple" training set patterns.

## 9.3 Proposed models: parameters

In general, two parameters had a large impact on the training loss curve: (i) The addition of the batch normalization layer at the start of the network, and (ii) the choice of optimizer (SGD or Adam). The E cientNet experienced no di erence with or without the batch normalization layer; likely due to the fact that it already includes many batch normalization layers. Only the VGG16 and TimeBlocks network with the batch normalization layer added were able to over t on all data sets, except for the Colorjitter data set. This di erence could be explained by the way a batch is normalized, by tting the data to the mean and variance using the trainable parameters and . Normalizing for ColorJitter data set is a greater challenge due to its varying composition at each iteration.

The choice of optimizer played an interesting role on the 3 networks. The TimeBlocks models trained with SGD were not able to reduce the training loss, as can be seen in Table 10. For the E cientNet and VGG16, the SGD optimizer showed to result in a more stable training loss curve, with less training loss spikes compared to Adam. This di erence could be the result of a too high learning rate of $10^{-4}$ for the models trained with Adam. To prevent this measures like a reduce learning rate function could be implemented.

Based on Figure 44, we can conclude that the models trained on larger (more complex) data sets experience high training loss spikes. The models trained with $(T_{p=10})$ were more stable compared to $(T_{p=50}; T_{p=75})$. Investigation of the dimensionality and context parameters showed the same correlation between data complexity and training stability: the samples trained with $(T_{d=1}; T_{c=1})$ were the most stable, followed by $(T_{d=3}; T_{c=1}); (T_{d=1}; T_{c=5}); (T_{d=3}; T_{c=5})$. This training instability was present to a lesser extent in the ColorJitter data set, which was caused by the introduced additional variations that regularize the model.

Further investigation of the non-technical parameters: time points, focal planes and contexts showed that more complex data sets over t faster; in the particular order for focal planes and context:

$$(T_{d=1}; T_{c=1}) \; ! \; (T_{d=3}; T_{c=1}; \_ \; T_{d=1}; T_{c=5}) \; ! \; (T_{d=3}; T_{c=5})$$

and for the time points:

$$T_{p=10} \; ! \; T_{p=50} \; ! \; T_{p=75}:$$

These results can be explained by the fact the model had more data-rich samples to e ectively over t faster on.

Interestingly, when comparing these statistics against models trained without the batch normalization layer, shown in Figure 36, we see the exact opposite for $(T_p; T_d; T_c)$ in terms of

over tting time. This could be due to the fact that, without this layer, the larger samples become too noisy, making even over tting too di cult.

### 9.3.1 TimeBlocks models: speci c parameters

The TimeBlocks models have 2 additional parameters: the number of time blocks $T_b$ and the ( lter) size of each block $T_f$. The following relationship was found between the number $T_b$ of and the likelihood of over tting: a TimeBlocks network containing 20 mini-networks over ts the fastest, followed by $T_{b=10}$ and $T_{b=5}$. The trend for larger networks to over t faster also holds for the second parameter $T_f$.

This can be explained by the fact that larger models have more parameters and it was intended that these additional model parameters would learn speci c time-interval features. However, due to the limited training size, these additional model parameters were prone to over tting.

## 9.4 Research Question & Sub-questions

In this the study we attempted to give an answer to the research question and the related sub-questions (SQ). The sub-questions are repeated and brie y answered below, based on the aforementioned information:

1. Is it possible to localize the position of an embryo in the frame?
   Yes, the results showed the trained U-Net was able to successfully detect the location of an embryo.

2. Is it possible to identify the most informative focal planes for a given time frame?
   Yes, the AGM technique could identify the focal planes with the most informative content. Furthermore, this robust method was also able to identify focus shifts.

3. What is the impact of the di erent data set con gurations on the performance?
   The in uence of the data set con gurations on the model performance could not be determined, due to over tting. What we did observe is that more complex data sets were able to over t faster when models were trained with $T_{bn=True}$.

4. Is a multi-branch network structure able to learn the features of a speci c time interval?
   The trained TimeBlocks models also over tted, because of this we were unable to determine whether a multi-branch network is able to learn the features of a speci c time interval.

5. Are the traditional models, VGG16 and E cientNet able to adapt to our classi cation task?
   We are unable to answer this question, given the speci c circumstances in which the models were trained. The pre-trained models were unable to generalize and over tted; thereby making it impossible to determine whether the (kernel) features optimized for the ImageNettask could be bene cially applied to our binary task.

### 9.4.1 Research Question: Can an AI model predict the implantation potential of an IVF embryo based on time-lapse developmental videos?

The primary question of this study was whether an AI model could accurately predict the implantation potential of an IVF embryo. After analyzing the results, we conclude that all the 960 models analyzed were unable to deliver on this task. This is due to the aforementioned reasons, including the fact that the data set is prone to over tting due to limited data and label imbalance (63% vs 37%).

## 9.5 Limitations and Future work

The research conducted in this study was limited by several factors. Starting with the initial data set of 820 compressed videos that contain less detail. Since neural network bene t from using high-quality input data, future work should potentially use uncompressed material. If this option is not available, the possibility of processing keyframes in the data set should be explored, since these frames are known to encode the most details. The data set consisted of 92 samples, which proved to be too few to build a binary classi cation model. As this was the main reason for the over tting, future research should include at least hundreds or perhaps thousands of samples.

In addition, the models were trained on a training set using a split ratio of 80% to 20%. This could result in leakage of embryos from the same donor to be in the training and test sets leading to generalization problems. This patient information is currently not available for each sample, nor is the information that strongly correlates with the success, such as patient/oocyte age, ethnicity, hormone levels. These features could potential be combined with the data set bene tting the model.

Furthermore, the data set su ered from label imbalance, with all models favouring the most common label 0 (no pregnancy). The introduction of a weight imbalance parameter in the loss function in future research may prevent or reduce this.

Regarding the pre-processing work ow, image cropping was successfully performed using the U-Net. However, the 400400 output frame may contain large amounts of non-embryo content, or have parts of the well edge, which increases the likelihood of over tting. To address this, future research should include full embryo segmentation. This would also address the issue of older embryos that have reached the blastocyst stage (older than frame 800) that do not t into the 400 400 frame. This makes full segmentation more important for future research, where we would like to include all embryo phases into the pre-processing. This segmentation approach would also bene t the AGM sharpness distribution, as background fragments are no longer included in the sharpness score.

The data sets in this research were built by sampling a small number of video frames because it was infeasible to train deep networks on all video frames. This resulted in some crucial frames being missed, which could be prevented by the use of methods like cumulative temporal clustering algorithms to identify important frames.

Alternatively, the network can be trained on a dynamic data set, rather than using a fixed set of frames [0, 200, 400, 600, 800] for with $T_{p=5}$. At each iteration, a frame can be uniformly selected between [0 and 200] for the first index. This approach would ensure that the network sees unique samples each time and that all the data can be seen by the network.

# 10 Conclusion

In this thesis we tried to find an answer to the research question:

- Can an AI model predict the implantation potential of an IVF embryo based on time-lapse developmental videos?

To answer this question, we successfully developed a pre-processing framework, capable of locating the embryo, obtaining the embryo's ROI and computing the sharpness distribution within the corresponding image stack.

Our 960 network training experiments on various data sets showed that these models were not able to learn the features related to the implantation potential. The results showed that the models overfitted on the training set. This fact was confirmed by the fact the models were able to learn on the Shuffle data set, which did not contain any structure except from the shared pixel intensity distribution. This provided evidence that the models were only learning the "trivial features". We found that the cause of the overfitting can be traced back to the following facts: 1) The data set contained too few samples; and 2) The data set was imbalanced. Additionally, we conclude that a too complex data set caused faster overfitting and training instability.

Whether an AI model is able to predict the implantation potential remains an open question. Further research and more embryos would be needed to definitely answer this question.

# A Appendix

Figure A.1: The radial gradient circular representation with a radius of 12. On the left the version between 0{255. On the right the normalized version. Referenced in Section 6.2.4.

Figure A.2: Representation of the embryo border replication process. Due to the embryo being to close to the well border, the 400400 crop can not be completed fully, allowing the maximum size of 384 400. Using the border replication, the outer last (horizontal) element is replicated until the 400 400 condition is satis ed. Referenced in Section 6.2.5.

$$w_{1;1} = 1\,; \quad w_{1;2} = 1\,; \quad b_{1;1} = \quad 1{:}5$$
$$w_{2;1} = 1\,; \quad w_{2;2} = 1\,; \quad b_{1;2} = \quad 0{:}5$$
$$w_{3;1} = \quad 1; \quad w_{4;1} = 1\,; \quad b_{2;1} = \quad 0{:}5$$

$$\begin{aligned}
x_1 &= 0 \\
x_2 &= 1 \\
z_{1;1} &= x_1 w_{1;1} + x_2 w_{2;1} + b_{1;1} \\
&= 0 \quad 1 + 1 \quad 1 + 1 \quad ( \quad 1{:}5) \\
&= \quad 0{:}5 \\
z_{1;2} &= x_1 w_{1;2} + x_2 w_{2;2} + b_{1;2} \\
&= 0 \quad 1 + 1 \quad 1 + 1 \quad ( \quad 0{:}5) \\
&= 0{:}5 \\
a_{1;1} &= \quad (z_{1;1}) \\
&= \quad ( \quad 0{:}5) \\
&= 0 \\
a_{1;2} &= \quad (z_{1;2}) \\
&= \quad (0{:}5) \\
&= 1 \\
z_{2;1} &= a_{1;1} w_{3;1} + a_{1;2} w_{4;1} + b_{2;1} \\
&= 0 \quad ( \quad 1) + 1 \quad (1) + 1 \quad ( \quad 0{:}5) \\
&= 0{:}5 \\
a_{2;1} &= \quad (z_{2;1}) \\
&= \quad (0{:}5) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
x_1 &= 1 \\
x_2 &= 0 \\
z_{1;1} &= x_1 w_{1;1} + x_2 w_{2;1} + b_{1;1} \\
&= 1 \quad 1 + 0 \quad 1 + 1 \quad ( \quad 1{:}5) \\
&= \quad 0{:}5 \\
z_{1;2} &= x_1 w_{1;2} + x_2 w_{2;2} + b_{1;2} \\
&= 1 \quad 1 + 0 \quad 1 + 1 \quad ( \quad 0{:}5) \\
&= 0{:}5 \\
a_{1;1} &= \quad (z_{1;1}) \\
&= \quad ( \quad 0{:}5) \\
&= 0 \\
a_{1;2} &= \quad (z_{1;2}) \\
&= \quad (0{:}5) \\
&= 1 \\
z_{2;1} &= a_{1;1} w_{1;3} + a_{1;2} w_{2;3} + b_{2;1} \\
&= 0 \quad ( \quad 1) + 1 \quad (1) + 1 \quad ( \quad 0{:}5) \\
&= 0{:}5 \\
a_{2;1} &= \quad (z_{2;1}) \\
&= \quad (0{:}5) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
x_1 &= 0 \\
x_2 &= 0 \\
z_{1;1} &= x_1 w_{1;1} + x_2 w_{2;1} + b_{1;1} \\
&= 0 \quad 1 + 0 \quad 1 + 1 \quad ( \quad 1{:}5) \\
&= \quad 1{:}5 \\
z_{1;2} &= x_1 w_{1;2} + x_2 w_{2;2} + b_{1;2} \\
&= 0 \quad 1 + 0 \quad 1 + 1 \quad ( \quad 0{:}5) \\
&= \quad 0{:}5 \\
a_{1;1} &= \quad (z_{1;1}) \\
&= \quad ( \quad 1{:}5) \\
&= 0 \\
a_{1;2} &= \quad (z_{1;2}) \\
&= \quad ( \quad 0{:}5) \\
&= 0 \\
z_{2;1} &= a_{1;1} w_{3;1} + a_{1;2} w_{4;1} + b_{2;1} \\
&= 0 \quad ( \quad 1) + 0 \quad (1) + 1 \quad ( \quad 0{:}5) \\
&= \quad 0{:}5 \\
a_{2;1} &= \quad (z_{2;1}) \\
&= \quad ( \quad 0{:}5) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
x_1 &= 1 \\
x_2 &= 1 \\
z_{1;1} &= x_1 w_{1;1} + x_2 w_{2;1} + b_{1;1} \\
&= 1 \quad 1 + 1 \quad 1 + 1 \quad ( \quad 1{:}5) \\
&= 0{:}5 \\
z_{1;2} &= x_1 w_{1;2} + x_2 w_{2;2} + b_{1;2} \\
&= 1 \quad 1 + 1 \quad 1 + 1 \quad ( \quad 0{:}5) \\
&= 1{:}5 \\
a_{1;1} &= \quad (z_{1;1}) \\
&= \quad (1{:}5) \\
&= 1 \\
a_{1;2} &= \quad (z_{1;2}) \\
&= \quad (0{:}5) \\
&= 1 \\
z_{2;1} &= a_{1;1} w_{3;1} + a_{1;2} w_{4;1} + b_{2;1} \\
&= 1 \quad ( \quad 1) + 1 \quad (1) + 1 \quad ( \quad 0{:}5) \\
&= 1{:}5 \\
a_{2;1} &= \quad (z_{2;1}) \\
&= \quad ( \quad 1{:}5) \\
&= 0
\end{aligned} \tag{7}$$

Figure A.3: The XOR calculations for the 4 inputs (0, 1), (1, 0), (0, 0) and (1, 1). The prove that MLP can solve nonlinear tasks. Referenced in Figure 6.