INSAT – National Institute of Applied Science and Technology

University of Carthage

Networks and Telecommunications Engineering

Embryo development stage onset detection and blastocyst quality assessment by time-lapse monitoring

Nour El Houda Araar,

Elyes Oueslati,

Taha Yassine Tizaoui

7. June 2022

Supervisor: Wided Miled Souid

Reviewer: Rabaa Youssef

Acknowledgments

This endeavor would not have been possible without Pr. Wided Miled. Her support, guidance and feedback were very valuable.
Also, Special thanks to Dr. Nozha chakroun and Dr. Sana Chtourou from Aziza Othmana hospital for giving the opportunity to work on this project. Their assistance and patience helped us tremendously in our task.
Big thanks to Pr. Rabaa Youssef, for reviewing this project.

Abstract

In vitro Fertilization is a complex process that involves retrieving eggs from ovaries and manually combining them with sperm in a lab for fertilization. When the fertilized egg divides, it becomes an embryo. Typically, 3 to 5 days after fertilization, the embryo with the highest quality is transferred into woman's uterus.

Continuous embryo monitoring with Time-Lapse Imaging (TLI) enables time based development metrics alongside visual features to assess an embryo's quality before transfer. Tracking embryonic cell development provides valuable information about its likelihood of leading to a positive pregnancy. Our project end-of-your project falls within this scope and deals with the 2-cell division timing as it is one of the most important criteria in the decision making process. To provide our solution, we focus on the use of neural networks, more precisely the ConvLSTM model, to detect this important onset first cleavage stage.

key words: IVF, Embryo selection, Cell cleavage, Deep learning, CNN, LSTM.

Contents

List Of Abbreviations

- **AI** Artificial Intelligence
- **BBox** Bounding Box
- **WHO** World Health Organization
- **CFAC** Canadian Fertility and Andrology Society
- **CNN** Convolutional Neural Network
- **LSTM** Long Short Term Memory
- **CPU** Central Processing Unit
- **ConvLSTM** Convolutional Long Short Term Memory
- **RNN** Recurrent Neural Network
- **GPU** Graphical Processing Unit
- **PoC** Proof of Concept
- **IVF** In vitro fertilization
- **TLM** Time-lapse Monitoring
- **SGD** Stochastic gradient descent
- **ConvLSTM2D** 2D Convolutional LSTM.

Introduction

Artificial intelligence (AI) is a field whose goal is to create machines capable of learning and improving themselves in an autonomous way. This technology is proving to be useful in all intellectual tasks. The concept of (IA) was then extended to encompass several subfields, including image classification, which has made considerable progress in recent years. This progress is due to numerous works in this field and to the availability of international image databases that have allowed researchers to report the execution of their approaches. This direction of research has resulted in the emergence and evolution of Deep Learning (DL), with the advent of Convolutional Neural Networks (CNN), a particular type of neural network whose architecture of connections is inspired by that of the visual cortex.

In the same trend, the use of artificial intelligence (AI) techniques is being intensively researched in the field of *In-vitro* fertilization (IVF), which is the most common fertility treatment. Systematic selection of embryos with the highest implementation potentials is a necessary step toward enhancing the effectiveness of IVF. Morphokinetics, i.e. analysis of the dynamics of cleavage divisions and processes such as compaction and cavitation, has provided both biologists and clinicians with a new set of data regarding embryonic behaviour during preimplantation development and its association with embryo quality. An automated system based on artificial intelligence could therefore improve IVF success rates by assisting embryologists with their decision and ensuring more consistent results.

As the morphology of the embryo at Day 2 has been the most used parameter for more than 25 years in the process of evaluating qualities of fertilized ovocytes and selecting the best embryo available, our end of year project falls within this scope aiming to develop a proof of concept (PoC) Computer Vision solution to detect the first embryo's cell division from Time-Lapse images. It consists of data collection and preprocessing, then using the latest state-of-the-art deep neural network architectures best known for image data to predict the stage onset first cleavage patterns.

This project is conducted in collaboration with the *reproductive Biology and Cytogenetics Laboratory* of Aziza Othmana Hospital. The laboratory is equipped with a Time-Lapse Embryoscope incubator for the culture of embryos. Using embryo image sequences acquired by this system, at different stages of development, the idea of our project is to assist the embryologists to annotate important embryo stages development, mainly the fist cell cleavage.

This report summarizes and presents the different steps of the project. It is organised as follows:

- The first chapter 1 introduces the scope of our project. We will first present the process of IVF. Then, the problem statement and the main goals of our work are mentioned.
- Chapter 2 is an opportunity to explain basic concepts on deep learning and image classification.
- Chapter 3 presents the used Dataset, its specifications and the preprocessing we have made to improve its quality.
- Chapter 4 presents the spatio-temporel prediction model we adopt, some implementation details and obtained results.

We conclude this report by a general conclusion and some future work perspectives.

1 Scope of the Project

Introduction

This chapter aims to define the overall context of the project. First, we present introductory information about *In vitro* fertilization. Then, we introduce the problem we aim to solve as well as the steps taken to build our solution.

1.1 Problematic

1.1.1 Facts and numbers

According to the World Health Organization (WHO), one in every four couples in developing countries suffers from infertility. In vitro fertilization (IVF), a process that aims to help people with fertility problems having a baby, is one of the most common infertility treatments that emerged about four decades ago and practiced over one million times annually around the world. During IVF, the fertilized eggs (refers to as embryos) are cultured for about 5 days inside an incubator to develop into blastocysts. According to the Canadian Fertility and Andrology Society (CFAC), only 33.1 percent of embryo transfer cycles led to a clinical pregnancy in Canada in 2017. Several studies suggest that the timing and the synchronicity of the *first few cleavages* during the early human embryonic development correlate with an embryo's potentials for developing into a healthy baby.

1.1.2 Stage onset detection of the first embryo cleavage

The beginning and duration of each embryonic cell stage provide valuable insight into embryo quality. For this project, we focused on detecting the time of the first cleavage at day 2 in the

process of the *In Vitro* Fertilization.

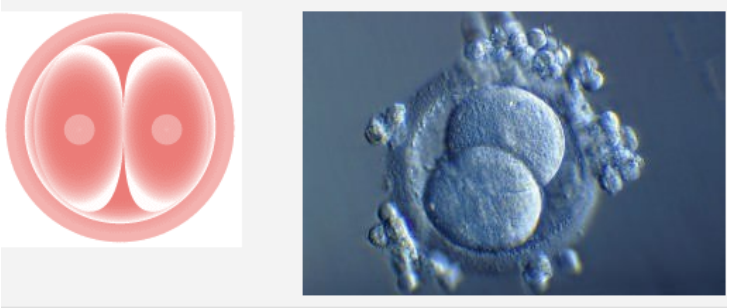


Figure 1.1: 2 cells stage, 30 hours after fertilization

Currently the tools available for making the decision of an embryo selection are limited, highly subjective, time-consuming, and often extremely expensive. As a result, embryologists often rely on their experience and observational skills when choosing which embryos to implant, which can lead to a lot of variability.

1.2 Purpose of the project

Approaches to reliably predict the developmental potential of embryos and select suitable embryos for blastocyst culture are needed. The development of time-lapse monitoring (TLM) and artificial intelligence (AI) may help solve this problem. The purpose of our project is to implement a proof-of-concept solution of a Computer vision system that helps medical professionals interpret Time-Lapse Images to make a decision about the embryo quality.
Our solution takes as input the frames of a Time-lapse sequence, and generates a classification to detect the frame of the first embryo's cell division as well as predict the time at which the division had occur.

1.3 Project steps

The main milestones of our project are:

- **Exploring possible data sources**
- **Selecting and applying Computer Vision techniques**
- **Embryo Stage Onset Detection and results evaluation**

These steps are a rough outline of the main tasks performed throughout our project, but the further detailed steps, their order and iterations are more related and dependent on the project methodology we will adopt.

1.4 Environment Setup and main software requirements

1.4.1 Environment Setup

One of the biggest challenges we encountered during our project was the limited computational resources. In fact, If we want to spend our time efficiently testing and learning across problem statements we need a GPU-based server. However, these GPU-based systems on platforms like Google Cloud Platform, Amazon Web Services, Azure can be very expensive.

But thanks to Kaggle and *Google Colab*'s democratization of deep learning, we were able to overcome this problem by running our python notebooks on the cloud with a GPU quota of 30 hours per week.

1.4.2 Main Libraries

The main libraries used over the course of the project are:

- *Tensorflow*: a deep learning framework that provides a rich set of workflows to build and develop deep learning models in python. [1]
- *Keras:* Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. [2]
- *Pillow:* a Python Imaging Library that adds image processing capabilities to your Python interpreter. It provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. [3]
- *OpenCV:* OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common

[1] https://www.tensorflow.org/
[2] https://keras.io
[3] https://pillow.readthedocs.io/en/stable/

infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products [4]

- *Matplotlib:* Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. [5]
- *NumPy:* a fundamental package for scientific computing in Python [6]

conclusion

Throughout this chapter, we have presented the problem statement, and defined the objectives of our project, as well as as the motivation behind it. Before going into deeper details, we will use the next chapter to explain the main basic concepts we will later use.

[4] https://opencv.org/about/
[5] https://matplotlib.org/
[6] https://numpy.org/

2 Preliminary study

Introduction

In this chapter, we first set and define some of the important concepts in AI and Computer Vision that we used throughout our project. Finally, we describe existing solutions for embryo selection.

2.1 Key concepts of AI and Computer Vision

2.1.1 Convolutional Neural Network

Convolutional Neural Network (CNN) are powerful programming models that allow image recognition by automatically assigning to each input image a label corresponding to its class. The architecture of the Convolutional Neural Network has a convolutional part upstream and therefore includes two distinct parts:

1. - A convolutional part: Its final objective is to extract the characteristics of each image by compressing them in order to reduce their initial size.
2. - A classification part: The CNN code obtained at the output of the convolutional part is provided as input in a second part, made of fully connected layers called Multi Layers Perceptron (MLP). The role of this part is to combine the characteristics of the CNN code to classify the image

2.1.2 Recurrent Neural Network

RNN is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic

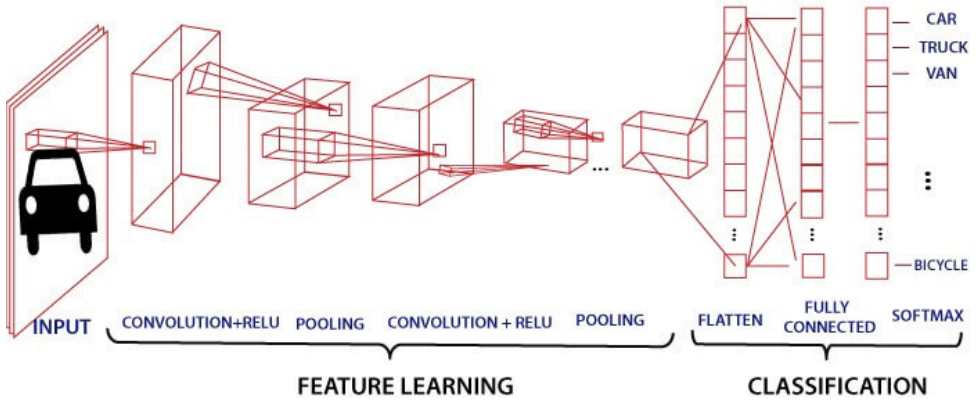


Figure 2.1: A sample CNN architecture

behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. In our work we used the Long Short Term Memory networks(LSTM) which are a special kind of RNN, capable of learning long-term dependencies.

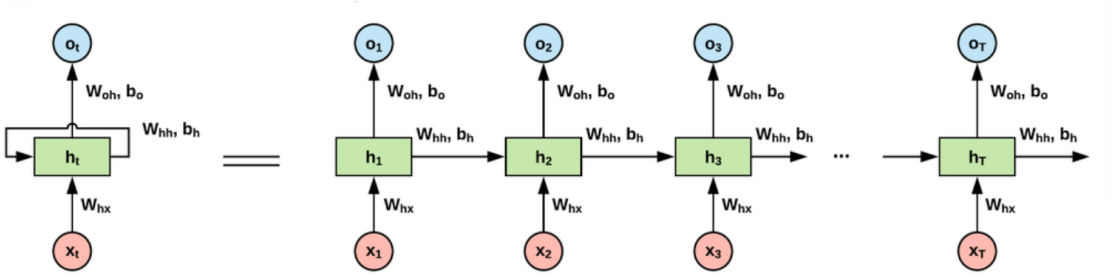


Figure 2.2: A sample RNN architecture

2.1.3 Image classification

Image classification is the task of predicting the class of an input image by calculating the likelihood/probability of the image belonging to a particular class. In other words, it is the process of assigning one label to an input image from a set of categories.
The first part of our work essentially comes down to classifying the medical images into two categories: t2 (2-cell cleavage timing) or not t2.

2.2 Related work

Some works propose a simplified version of the embryo selection problem with fewer classes by focusing on the phases between 1 and 5 cells and propose to model only two classes: blastocyst and non-blastocyst. Others argue that reducing the number of classes allows for sharper annotations, as there is much less intra- and inter-operator variability when only these two classes are modeled. Models such as AlexNet , a set of residual dilated U-Nets or Xception are used as classifiers.

Other works propose to deal with a reduced number of phases, namely the first 6 phases of development. The context obtained by using several neighboring frames has also been exploited. Most of the works propose a late fusion by computing the same features on several successive frames, then aggregating the information to produce a prediction. For example, it has been proposed to concatenate the feature vectors of the images computed by the same convolutional neural network (CNN), or to aggregate them using maxpooling.

It has also been shown that it is advantageous, when several images are used, to predict the label of neighboring images using the features of the current image. Finally, others use an LSTM to take into account the features extracted at each time step.

The different developmental phases are strictly ordered: the embryo can sometimes skip a phase but it is very rare that it returns to a previous phase. This constraint is often exploited by dynamic programming to make the prediction consistent over the whole video.

Conlusion

In this chapter, we introduced some key concepts on AI and Computer Vision, used in our solution. Then, a present a brief review of existing methods dealing with the problem of embryo stage development detection.

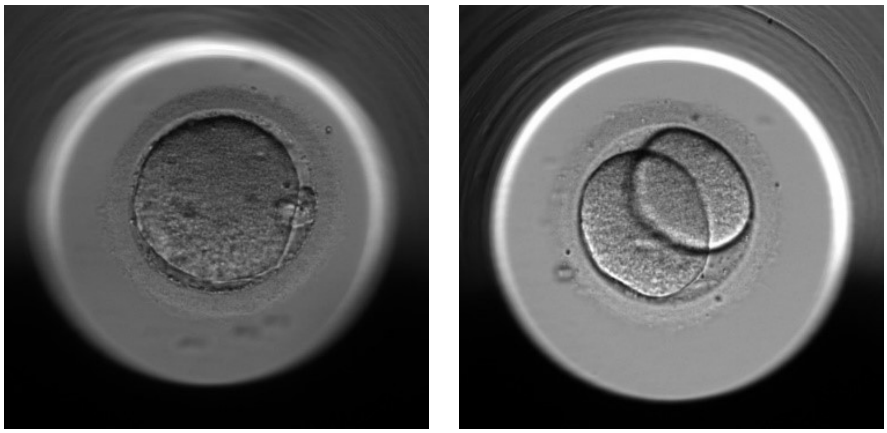
3 Data Collection and Preprocessing

Introduction

In this chapter, we're going to discuss the specifications of our dataset, and detail the different preprocessing steps we applied before feeeding our sequences to the model.

3.1 Dataset Specifications

The dataset we used in this work is comprised of 756 videos obtained from the IVF Laboratory of the University Hospital of Nantes. Each video contains between 300 and 600 frames in gray scale with a resolution of 500×500 pixels [3]. The actual time between two images is 10 to 20 minutes. One video shows the development of a particular embryo. An experienced biologist notes the start and end time of each phase of the embryo's development. Each image of each video has therefore a class, which corresponds to the phase seen in the image.



(a) (b)

Figure 3.1: Embryo stages development (a) before first cellular (b) first cell division

Given the fact that we worked on predicting the start of the phase t2 (First cellular division)

we eliminated every frame which goes beyond the end of this particular phrase. Due to limited computational resources, we set the number of images in each sequence to 50. For each sequence, we stored the time to frame correspondences in a csv file to use it later for predictions

Figure 3.2: sequence example

3.2 Data preprocessing

3.2.1 Object Detection and Cropping

Before feeding the sequence of images to our model, we cropped the images to conserve only the cell which represents our primary focus and gain in memory efficiency. To achieve this task, we used Transfer Learning through the TensorFlow Lite Model Maker package. Transfer learning is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

This high-level library [4] simplifies the process of training a TensorFlow Lite model using custom dataset. Retraining a TensorFlow Lite Object Detector model with our own custom dataset reduces the amount of training data required thus shortening the training time. Another key point behind our choice was its fast execution as it was designed to work with mobile devices and edge computing.

This model leverages the power of YOLO (You Only Look Once) [2] algorithm which is state of the Art Algorithm for Real-Time Object Detection System. It is based on regression, instead of selecting the interesting part of an image, it predicts classes and bounding boxes for the whole image in one run of the algorithm.

To understand the YOLO algorithm, we must first focus on what is being predicted. Finally, we want to be able to forecast an object's class and the bounding box that defines its placement. Four descriptors can be used to describe each bounding box:

1. Center of the box (bx, by)
2. Width (bw)
3. Height (bh)
4. Value c corresponding to the class of an object

We also forecast a real number p_c, which is the likelihood of finding an object within the bounding box. Then, instead of looking for interesting parts in the input image that potentially contain an item, YOLO divides the image into cells, usually in a 19x19 grid. After that, each cell is in charge of predicting k bounding boxes.

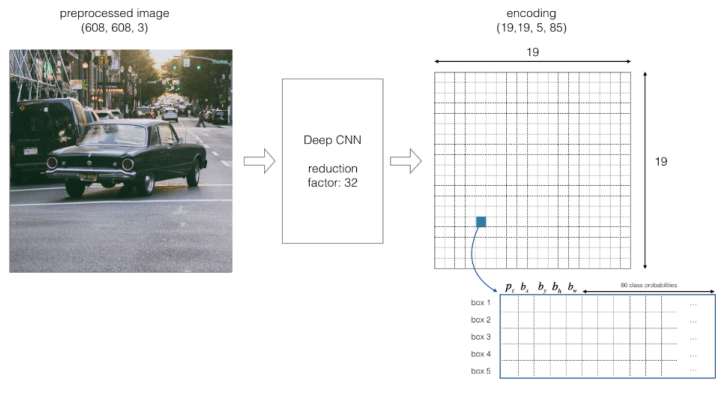


Figure 3.3: Bounding Boxes Mechanism

Only if the anchor box's center co-ordinates are in that cell is an object regarded to be in that cell. The center co-ordinates are always determined relative to the cell because of this attribute, whereas the height and width are calculated relative to the entire image size.

During the one pass of forwards propagation, YOLO determines the probability that the cell contains a certain class. The equation for the same is :

$$score_{ci} = p_c * c_i$$

The class with the greatest likelihood is picked and allocated to that grid cell. All of the grid cells in the image go through a similar process. This graph depicts the results of forecasting the class probabilities for each grid cell before and after. After predicting the class probabilities, the following stage is Non-max suppression, which aids the algorithm in removing unneeded anchor boxes, as shown in the image below.

Figure 3.4: unneeded anchor boxes

To resolve this problem Non-max suppression eliminates the bounding boxes that are very close by preforming the IoU (Intersection over Union) with the one having the highest class probability among them.

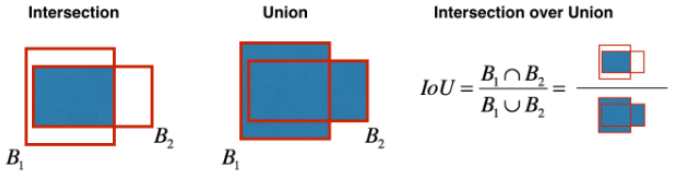


Figure 3.5: IoU operation

Once done, algorithm finds the bounding box with next highest class probabilities and does the same process, it is done until we are left with all the different bounding boxes.

Figure 3.6: Non-max suppression

Figure 3.7 shows an example of a cell detection applied on an embryo image of our dataset.

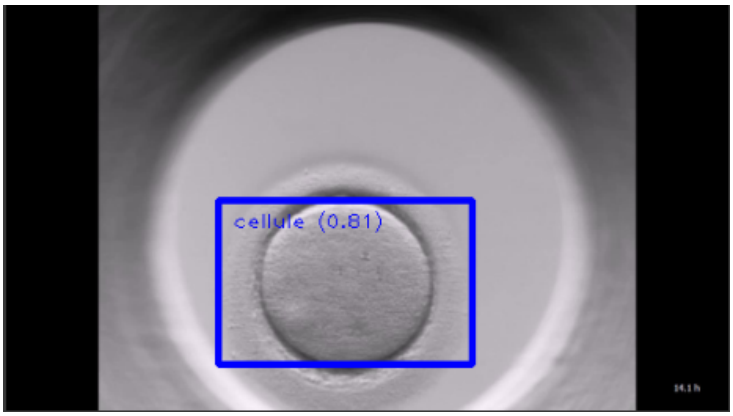


Figure 3.7: cell detection with 0.81 confidence

3.2.2 Contrast Adjustment and Normalization

As each pixel value can vary from 0 to 255 with each number representing the color intensity, feeding an image directly to our neural network will certainly result in complex computations and a slow training process given the high numeric values. To address this problem, we proceed to normalize the values between 0 and 1 by dividing by 255. Thus, the numbers will be smaller and the computation faster.

A part from normalization, we carried out two additional operations called Equalization and contrast normalization [5]. The first operation consists of equalizing the image's histogram by applying a non-linear function that creates a uniform distribution of pixel values. The second one consists of normalizing contrast by writing a function that creates a histogram of the input picture (or mask region), subtracts the cutoff percent of the brightest and darkest pixels from the histogram, then remaps the image so that the darkest pixel is black (0) and the lightest pixel is white (255).

This transformation had a significant impact on the performance of our model as levels of contrast varied tremendously from sequence to sequence which prohibited the model at first from capturing the essence of our data.

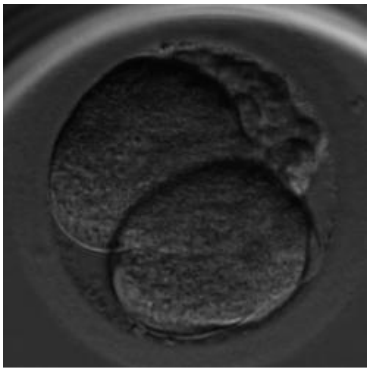
```
img = Image.open(os.path.join(os.path.join(BASE_DIRECTORY,directory),filename)).convert('L')#test grayscale
img=PIL.ImageOps.autocontrast(PIL.ImageOps.equalize(img)) #magic
numpydata = (np.asarray(img))/255 #normalizing data
```

Figure 3.8: Code snippet of normalization and contrast adjustment

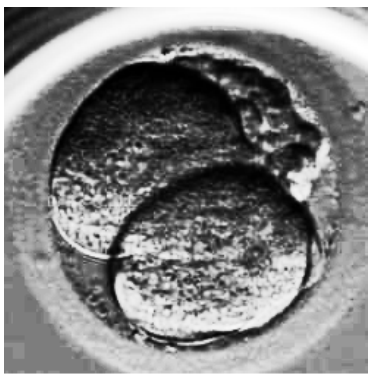
(a) Original bright Image

(b) Transformed Image

Figure 3.9: An example of a bright image transformation



(a) Original dark Image



(b) Transformed Image

Figure 3.10: An example of a dark image transformation

3.2.3 Sequence Preparation

In practice, downloading multiple videos to RAM to feed the model at the same time is simply not feasible. When working with images, such a memory problem is quite prevalent. A solution is to utilize a custom generator, which allows us to feed the model sample per sample instead of all at once. For each sequence of 50 images, we proceeded to convert each image to a numpy array along with its corresponding sequence of labels

For our label encoding we choose to encode the pre-t2 phase with 1 and the t2 phase with 0. Our result is a (numframes, numchannels, imgHeight, imgWidth) vector representing a sequence of images and (numBranches, 1, numFrames, 1) vector representing our label vector.

numframes	50
numchannels	1
imgHeight	250
imgWidth	250

numbranches	2
numFrames	50

Down Below are the code snippets responsible for loading the sequences and labels:

```
#with PIL
def Load_seq(BASE_DIRECTORY,TARGET_PATH,length,sorting_func):
  i=0
  for directory in os.listdir(BASE_DIRECTORY):
    filenames=sorted(os.listdir(os.path.join(BASE_DIRECTORY,directory)),key=sorting_func)
    stack=[]
    for filename in tqdm(filenames):   # tqdm is for displaying progress bar
      img = Image.open(os.path.join(os.path.join(BASE_DIRECTORY,directory),filename)).convert('L')#test grayscale
      img=PIL.ImageOps.autocontrast(PIL.ImageOps.equalize(img)) #magic
      numpydata = (np.asarray(img))/255 #normalizing data
      numpydata = np.expand_dims(numpydata,axis=-1)
      #numpydata = np.moveaxis(numpydata, -1, 0)
      stack.append(numpydata)
    output=np.stack(stack, axis=0)
    np.save(TARGET_PATH+'/scene_{}.npy'.format(i),output)
    print(get_t2_start(directory)-70)
    Load_labels(get_t2_start(directory)-70,i,length,TARGET_PATH)
    i=i+1
```

Figure 3.11: Sequence Loading Function

```
def Load_labels(t2_time,j,length,TARGET_PATH):
  array=np.ones((length,1))
  array2=np.zeros((length,1))
  for i in range(t2_time):
      array[i]=0
      array2[i]=1
  array=np.expand_dims(array, axis=0)
  array2=np.expand_dims(array2, axis=0)
  category=np.stack([array2,array],axis=0)
  #category=np.expand_dims(array, axis=2)
  np.save(TARGET_PATH+'/category_{}.npy'.format(j),category)
```

Figure 3.12: Labels Loading Function

Conclusion

To conclude, in this chapter we tried to introduce dataset specifications, then we performed data selection keeping only the necessary frames of the embryo development stage (until first cellular division). This step was followed by object detection to crop images which have then been processed by adjusting contrast and brightness. Lastly, we had the sequences ready for training and inference.

4 Methodology

Introduction

In this chapter, we will explore our model architecture. We will also present our hyperparameters and why we chose them. We will showcase in the end our results and discuss them.

4.1 Model Architecture

4.1.1 Convolutional LSTM Network

ConvLSTM is a spatio-temporal prediction recurrent neural network with convolutional structures in both the input-to-state and state-to-state transitions. The ConvLSTM uses the inputs and past states of its local neighbors to predict the future state of a cell in the grid. Using a convolution operator in the state-to-state and input-to-state transitions is a simple way to accomplish this 4.1. The image is sent through the convolution layers, and the output is a set of features flattened to a 1D array. When you apply this method to all of the photos in the time set, you get a set of characteristics that change over time, which is the LSTM layer input.

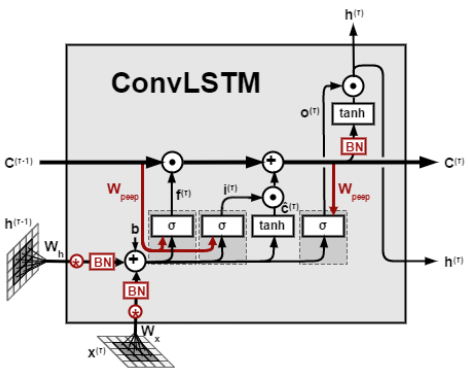


Figure 4.1: ConvLSTM layer

4.1.2 Proposed model for cell division detection

In the figure below we'll find the architecture of the model used in detection first cell division. The architecture begins with two ConvLSTM layers, each with a BatchNormalization and a MaxPooling layer in between. It breaks into 2 branches, one for each category. All of the branches start with one ConvLSTM layer and then a MaxPooling layer. The output is then connected to a Dense network that is fully connected. Finally, the last layer is a single-cell Dense.

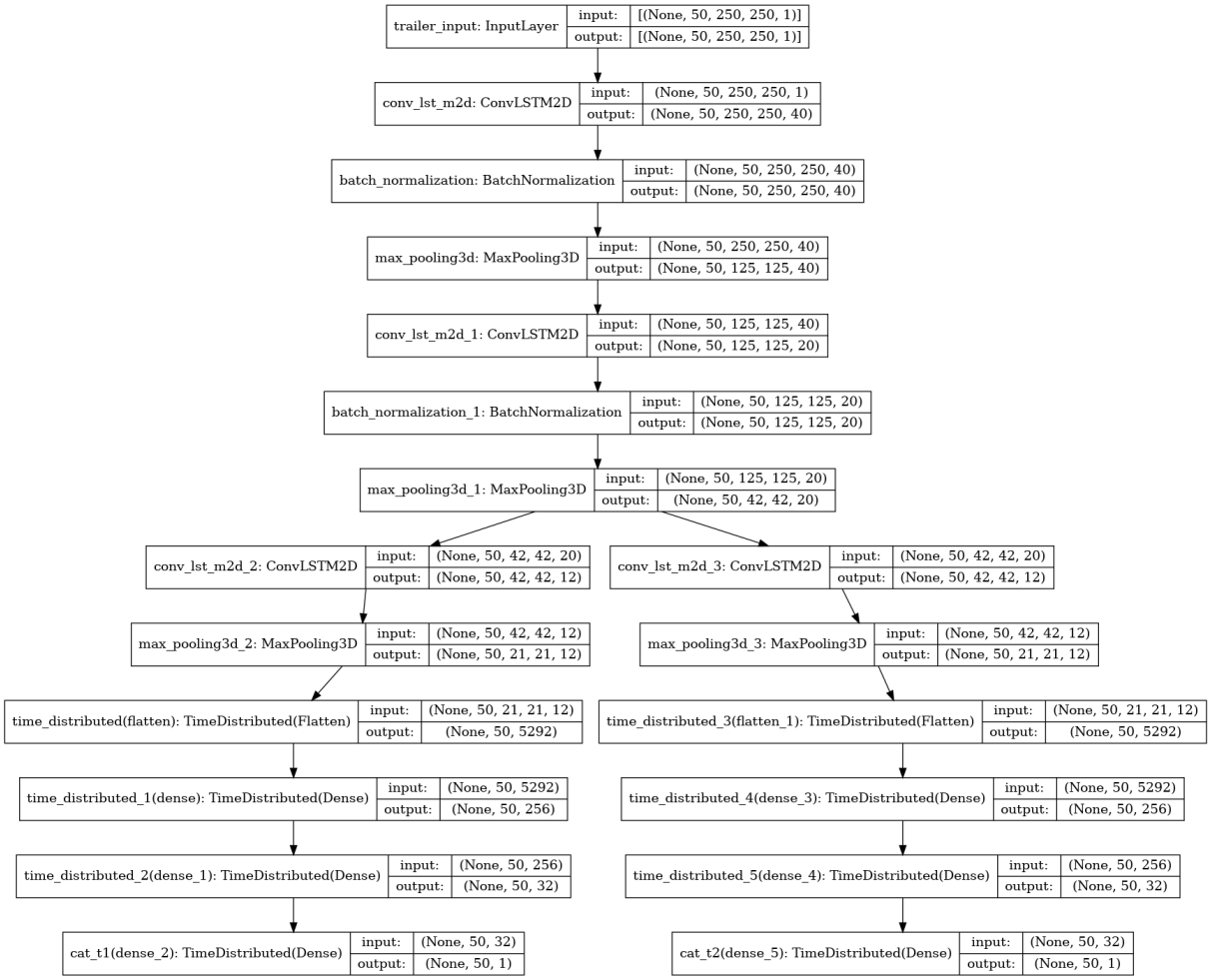


Figure 4.2: Model Architecture

Input layer

The input to the LSTM cell is a set of data over time, in the form of a 3D tensor with shape (samples, time steps, and features). The input to the Convolution layer is a set of images in the form of a 4D tensor with shape (samples, rows, cols, channels). A set of images over time as a 5D tensor with shape (samples, time steps, rows, cols, channels) is the input of a ConvLSTM.

ConvLSTM 2D layer

Convolution and LSTM outputs are combined in the ConvLSTM layer output. If return sequences = True, it returns a sequence as a 5D tensor with shape (samples, time steps, rows, cols, filters), just like the LSTM. If return sequences = False, however, only the last value of the sequence is returned as a 4D tensor with form (samples, rows, cols, filters).

MaxPooling

By aggregating the existence of features in areas of the feature map, pooling layers provide a method for down sampling feature maps. Average pooling and maximum pooling are two popular pooling approaches. In this model we're using MaxPooling.

Branching

In order to detect multiple phases of the embrayo development, we use multiple branches with each one detecting its corresponding phase. The branches have 2 convLSTM2D layers in common but then each branch get its own convLSTM2D layer and its own feature maps. This will allow the architecture extract features for each phase and keep the computational and memory needs to a minimum at the same time. In the Scope of this project, We're only detecting 2 phases so we will be using 2 branches. This can be used later in predictions part to select the most certain prediction.

Time Distributed Dense layer

In the Dense layers part in the model, We apply the TimeDistributed Keras wrapper. The result is time distributed dense layers. This allows you to apply the layer operation on each timestamp. Otherwise when we flatten the data all the image instances will be combined and the time dimension will be lost.

4.2 Hyper-parameter tuning

4.2.1 Activation functions

To make the network dynamic, we need to apply an activation function [11] and give it the ability to retrieve sophisticated and intricate information from input as well as represent nonlinear convoluted random functional mappings.

Recurent activation function

For the final output layer, we use hard sigmoid activation. The Hard Sigmoid is an activation function used for neural networks of the form: $f(x) = \max\left(0, \min\left(1, \frac{(x+1)}{2}\right)\right)$. We notice that transformation from t1 to t2 is fast and takes only between 1 to 4 frames. Hard sigmoid will allow the recurrent network to adapt faster than simple sigmoid activation function.

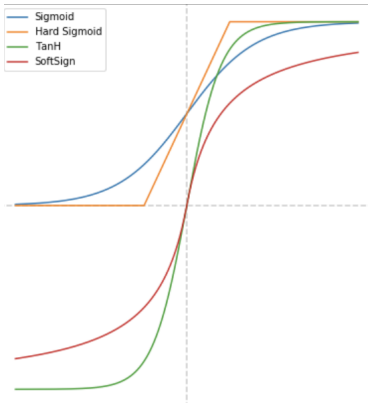


Figure 4.3: Hard sigmoid function compared to other activation functions

output layer activation function

For the final output layer, we use sigmoid activation function for each image classification. It is the most widely used activation function as it is a non-linear function. Sigmoid function transforms the values in the range 0 to 1. It can be defined as: $\frac{1}{1+e^{-x}}$

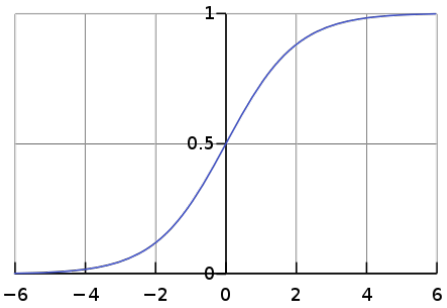


Figure 4.4: Sigmoid function

4.2.2 Dropout

Dropout is a regularization approach for simulating concurrent training of a large number of neural networks with varied topologies. During training, a certain number of layer outputs are rejected at random. The value of dropout in each convlstm2d layer is 0.01.

4.2.3 Optimzer : SGD

Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. Stochastic gradient descent performs a parameter update for each training example :

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Adaptive optimization approaches such as Adam, Adagrad, or RMSprop have been proven to generalize poorly when compared to SGD, despite higher training results [6]. These approaches perform well in the early phases of training but are outperformed by SGD as the training progresses. For this reason, We chose SGD to get better results in test dataset.

4.2.4 Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. We used 3 types of callbacks :

```
early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss", patience=10)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=5)
checkpoint = keras.callbacks.ModelCheckpoint('./checkpoint', monitor='loss',save_best_only=True, mode='min')
```

Figure 4.5: model callbacks

Early Stopping

Early stopping [7] is a type of regularization that is used to avoid overfitting when using an iterative method like gradient descent to train a learner. With each iteration, such methods update the learner to make it better fit the training data.

Reduce Learning rate

When learning becomes stagnant, models often benefit from reducing the learning rate by a factor of 2-10. This callback [8] monitors a quantity and reduces the learning rate if no improvement is seen after a 'patience' number of epochs.

Checkpoint

The ModelCheckpoint [10] callback is used in conjunction with model.fit() training to save a model or weights (in a checkpoint file) at a predetermined interval, so that the model or weights can be loaded later to resume training from the saved state.

4.3 Loss function and precision

We're looking for measurements or a function that will help us improve the performance of our model. The loss function indicates how well your model predicts. The Loss will be the smallest if the model predictions are closest to the actual values, and the maximum if the predictions are completely different from the original values. Binary crossentropy loss function is used in this model.

4.3.1 Binary cross-entropy

The binary crossentropy loss function [12] is utilized in binary classification problems. These are assignments in which there are just two options for answering a question (in our case t1

and t2). Binary cross-entropy can be calculated as:

$$Loss = -(y \log(p) + (1 - y) \log(1 - p))$$

4.3.2 Metrics

Accuracy

Accuracy is the proportion of the total number of predictions that were correct. In our case, accuracy will be the proportion of correct prediction image by image.

Cell Division frame accuracy

Another metric to calculate the performance of our model will be frame accuracy. A correct prediction will be predicting the first t2 classified frame correctly. For example, if the cell division (t2) begins at the frame 25 the accurate prediction will be classifying the first 24 frames as t1 and the rest as t2. To understand more our model performance, will give also the frame accuracy in a given interval of frames. If we take an interval of 3 frames, predicting the first t2 frame before or after 3 frames will be considered correct. The results below 4.1 are obtained with choosing the branch that is more certain in its prediction. This is achieved with calculating the mean entropy of every branch prediction where the two branches does not have the same class prediction. Mean Entropy can be calculated as :

$$Entropy_{mean} = -\sum_{n=1}^{N} p_n log_2(p_n) + (1 - p_n) log_2(1 - p_n)$$

Where N is the number of frames where the 2 branches does not give the same output. The branch with the minimum mean entropy (more certain), where the two branches don't agree, is chosen.

4.4 Data generator

A custom data generator was used. This generator feeds the model a sequence at a time. This can be later modified to apply data augmentation without loosing memory space.

```
def generate_arrays(available_ids):
    from random import shuffle

    while True:

        shuffle(available_ids)
        for i in available_ids:

            scene = np.load('/content/drive/MyDrive/PFA/training/scene_{}.npy'.format(i))
            category = np.load('/content/drive/MyDrive/PFA/training/category_{}.npy'.format(i))
            yield (np.array([scene]), [category[0],category[1]])
```

Figure 4.6: Data generator function

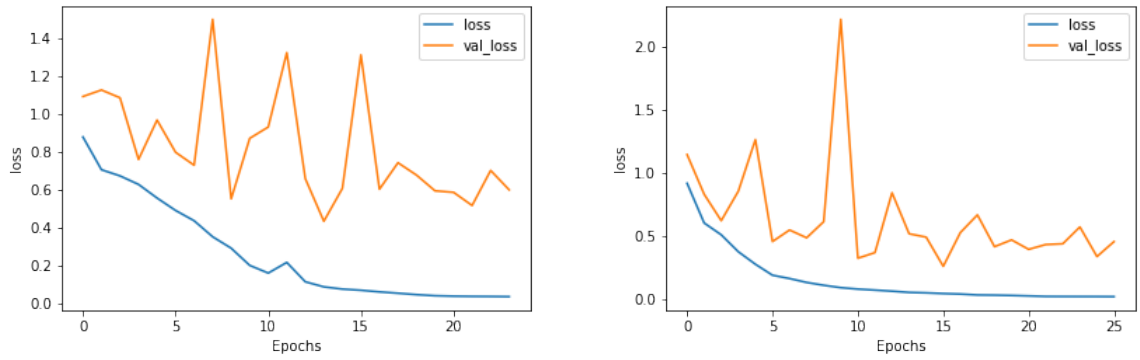
```
def generate_arrays_augmented(available_ids):
    from random import shuffle

    while True:

        shuffle(available_ids)
        for i in available_ids:

            scene = np.load('/content/drive/MyDrive/PFA/training/scene_{}.npy'.format(i))
            category = np.load('/content/drive/MyDrive/PFA/training/category_{}.npy'.format(i))
            for seq in DataAugmentation(scene):
              yield(np.array([seq]),[category[0],category[1]])
            yield (np.array([scene]), [category[0],category[1]])
```

Figure 4.7: Augmented Data generator function

4.5 Experimental results and discussion

In this section model v1 will be the model before applying the preprocessing techniques of contrast and brightness correction. Model v2 will be the model that trained on preprocessed images. The models will train on 130 sequences of 50 images totaling 6500 image.Later on, the model will be tested on 30 sequences

4.5.1 Model Loss

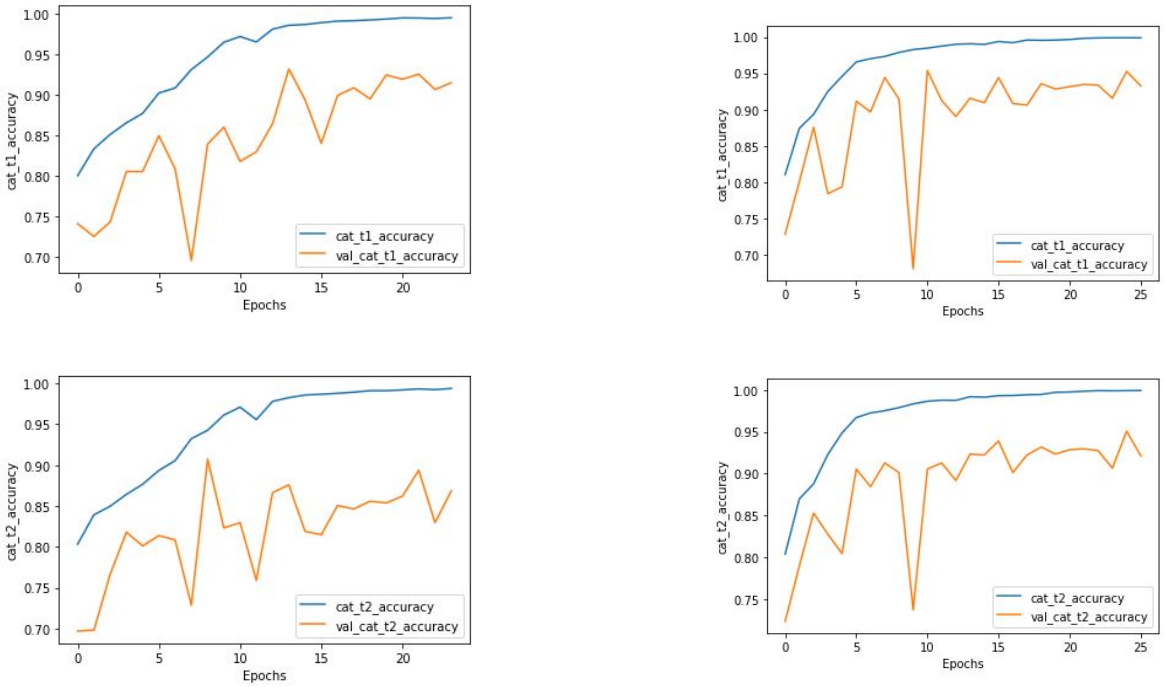


(a) modelv1 loss

(b) modelv2 loss

Figure 4.8: Loss across the versions of our models

4.5.2 Model accuracy



(a) modelv1 loss

(b) modelv2 loss

Figure 4.9: Accuracy across the versions of our models

4.5.3 Precision in Frames

As the goal is to predict cell division time. The results below show the accuracy of our model in terms of predicting the first frame of the cell division (t2).

Frame Diffrence	modelv1		modelv2	
	Training	Test	Training	Test
0 (precise)	18.46%	13.33%	70.76%	16.66%
$\leq 1frame$	47.69%	36.66%	93.07%	53.33%
$\leq 2frame$	57.69%	43.33%	94.61%	73.33%
$\leq 3frame$	62.3%	50%	96.15%	93.33%
$\leq 4frame$	66.15%	50%	96.92%	93.33%
$\leq 5frame$	70.76%	56.6%	97.69%	96.66%

Table 4.1: accuracy in an interval for division time detection

	modelv1		modelv2	
	Training	Test	Training	Test
mean diffrence (in number of frames)	4.48	7.79	0.692	2.4

4.6 Discussion

Although the loss and accuracy in modelv2 didn't see a great improvement, We see that the model became very precise in determining the time of cell division. We see a 50% jump in cell division for 0 frame and 1 frame interval accuracy. Modelv2 approaches 100% frame interval accuracy for an interval of 5 frames which means it rarely misses the division. In testing we see reduced accuracy in both models. However modelv2 does not miss the division in most cases even in testing (93% accuracy in 3 frame interval compared to only 50% accuracy in modelv1). We can say that modelv2 generalizes well.

Conclusion

To conclude, we introduced our model architecture that gave good results with the help of hyperparamater tuning and data preprocessing.

Conclusion and Future Work

At the end of this report, we draw a complete assessment of our work suggested the embryologists of the IVF Laboratory at CHU Aziza Othmana. The project entitled " Embryo development stage onset detection and blastocyst quality assessment by time-lapse monitoring" constitutes a step forward towards the automation of the detection of the different embryo cell stages.

This application can indeed be of great help to embryologists and save huge amounts of time spent on manual detection of the embryo development stages. In addition, the proposed framework is capable of predicting cell-stage and detecting cells by an accuracy rate of about 93% in an interval of 3 frames, this helps infertility treatment techniques reach their maximum potential, thus solving the problem of many childless couples.

In terms of work, we, first defined a pipeline encompassing the different preprocessing steps starting by the selection of relevant data, then the execution of object detection, the image processing to lastly get the sequences ready for training and inference.

The model is comprised of one common branch that expands to 2 branches (and more in the future). Every branch is specialized in predicting a single category. The more certain prediction is then chosen to predict the cell division time.

Finally, it is worth-noting that our application can be further improved. Indeed, we intend to work on increasing the predictive power of our model through the following methods. First, carrying out an exhaustive and automated hyper-parameter tuning. This method requires significant computational resources which are not currently available for us. Second, we can also increase the training data size to enhance the generalization capacity of our model by lowering estimation variance and overfitting. Third, we will try to further expand the detection capacity of our model so as to identify subsequent stages like 4-cell stage, 8-cell stage and the blastocyst stage. Last but not least, we plan to test new neural network architectures like Resnet3D and some others which proved to have great and promising results.

Bibliography

[1] YOLO architecture
*https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_*

[2] YOLO algorithm explained *https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4*

[3] Apprentissage profond appliqué à la classification d'images microscopiques embryonnaires *Tristan Gomez, Harold Mouchère, Thomas Fréour, Magalie Feyeux. Apprentissage profond appliqué à la classification d'images microscopiques embryonnaires. Rencontres des Jeunes Chercheur×ses en Intelligence Artificielle (RJCIA 2020), Jun 2020, Angers, France. hal-02882052*

[4] TensorFlowLite object detection
https://www.tensorflow.org/lite/models/modify/model_maker/object_detection

[5] Equalize and contrast normalization
https://pillow.readthedocs.io/en/stable/reference/ImageOps.html

[6] Improving generalization performance by switching from adam to sgd *Keskar, Nitish Shirish and Socher, Richard*, arXiv preprint arXiv:1712.07628

[7] Early Stopping documentation *https://keras.io/api/callbacks/early$_s$topping/*

[8] Reduce learning rate documentation *https://keras.io/api/callbacks/reduce$_l$r$_o$n$_p$lateau/*

[9] Checkpoint documentation *https://keras.io/api/callbacks/model$_c$heckpoint/*

[10] Checkpoint documentation *https://keras.io/api/callbacks/model$_c$heckpoint/*

[11] Activation functions in neural networks *Sharma, Sagar and Sharma, Simone and Athaiya, Anidhyar*, towards data science

[12] Binary cross entropy with deep learning technique for image classification *Ruby, Usha and Yendapalli, Vamsidhar*, Int. J. Adv. Trends Comput. Sci. Eng

List of Figures