UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTERS' THESIS

MASTER IN SIGNAL THEORY AND
COMMUNICATIONS

DESIGN AND IMPLEMENTATION OF AN AUTOMATIC CHANGE DETECTION SYSTEM OVER UNMANNED AERIAL VEHICLE'S SEQUENCES WITH TENSORFLOW

VÍCTOR GARCÍA RUBIO

2019

MASTERS' THESIS

Title: Design and implementation of an automatic change detection system over unmanned aerial vehicle's sequences with TensorFlow.

Author: Víctor García Rubio

Tutor: Juan Antonio Rodrigo Ferrán

Rapporteur: José Manuel Menéndez García

Department: Departamento de Señales, Sistemas y Radiocomunicaciones.

Grupo de Aplicación de Telecomunicaciones Visuales.

TRIBUNAL:

President:

Vocal:

Secretary:

Substitute:

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

MASTERS' THESIS

MASTER IN SIGNAL THEORY AND COMMUNICATIONS

DESIGN AND IMPLEMENTATION OF AN AUTOMATIC CHANGE DETECTION SYSTEM OVER UNMANNED AERIAL VEHICLE'S SEQUENCES WITH TENSORFLOW

VÍCTOR GARCÍA RUBIO

2019

Resumen

En este documento, el resultado del desarrollo de un sistema capaz de detectar cambios entre dos secuencias de vídeos obtenidas de Vehículos Aéreos No Tripulados (VANT) es descrito.

Para ello, un modelo de inteligencia artificial has sido desarrollado. Este está basado en redes neuronales convolucionales, un tipo de red neuronal multicapa concebida para el procesamiento de imágenes mediante inteligencia artificial.

Para realizarlo, diferentes implementaciones que conforman el estado del arte actual han sido revisadas para obtener las características básicas de un sistema óptimo.

El siguiente paso ha conllevado la exploración de múltiples conjuntos de datos para la detección de cambios entre dos secuencias de video. Esto se debe a que no contábamos con los recursos suficientes para poder generar uno independiente que se ajustara a nuestras necesidades concretas. Tras esto, se han seleccionado los dataos que más se adecúan a nuestro propósito. Una vez obtenidas las imágenes etiquetadas, hemos modificado y reordenado la estructura de los datos para poder entrenar nuestro modelo de inteligencia artificial.

Posteriormente, hemos diseñado la arquitectura del modelo y establecido los parámetros iniciales. La siguiente fase del proyecto ha consistido en el entrenamiento del modelo con los datos confeccionados, junto con el análisis de los resultados de dicho proceso. En base a estos resultados, se han ajustado los distintos parámetros del modelo para incrementar su rendimiento. Una vez obtenido el modelo final, hemos comparado sus resultados con las implementaciones del estado del arte mencionadas anteriormente.

Finalmente, las conclusiones obtenidas del proceso completo de desarrollo del proyecto han sido descritas. Además, hemos remarcado las principales líneas de futuro trabajo del sistema.

Palabras clave

Change detection, image processing, deep learning, TensorFlow, Keras, OpenCV, Python.

Summary

In this document, the result of the development of a system able to detect changes between two video sequences obtained from Unmanned Aerial Vehicles (UAV) is reported.

For this purpose, a deep learning model has been created. It is settled on convolutional neural networks, a multilayer neural network conceived for image processing with artificial intelligence.

To do so, state-of-the-art implementations have been reviewed to obtain the base features for an optimized system.

The next step has involved the exploration of datasets for change detection between video sequences, as the resources to generate one where not available. The most suitable datasets for the system architecture and have been selected. Once we have a collection of labeled images, we have modified and reordered the structure of the mentioned dataset in order to train the deep learning model.

Afterwards, we have designed the architecture of the model and established the initial hyperparameter. The next phase of the has consisted on training the model with the dataset obtained, alongside with the analysis of the outcomes of this process. From these results, the adjustment of the different parameters of the model has been performed. After a final model has been obtained, we have compared its performance with the state-of-the-art implementations analyzed previously.

Finally, the conclusions of the entire development process of the project have been remarked. In addition, we have highlighted the main future work's lines of the system.

Keywords

Change detection, image processing, deep learning, TensorFlow, Keras, OpenCV, Python.

Acknowledgements

I would like to thank my family for its unconditional support along my educational career. Moreover, I am grateful to my company, Visiona I.P. to bring me the opportunity to develop this project. Finally, I wish to thank my tutor and rapporteur for their guidance alongside this work.

Contents

List of Figures

List of Tables

List of Equations

1 INTRODUCTION AND OBJECTIVES

1.1 INTRODUCTION

Unmanned Aerial Vehicles (UAV) have supposed a revolution on the surveillance field. The capability of this devices to cover significant amounts of land and to access difficult locations have affected significantly this sector. However, security tasks require a deeper analysis in order to avoid dangerous situations. The need for automation in these activities has increased the importance of change detection methods. These methods are usually based on images acquired by drones. The drone image acquisition adds a considerable milestone for these methods: the movement of the camera. This is the main challenge of the algorithms, as they have a variable background. Furthermore, the weather conditions as well as the precision of GPS position influence the relation between the acquired frames and the location of the UAV.

In our case, we have focused on the detection of changes between two sequences of an UAV's route. Our purpose is to modify the behavior of this vehicle when modifications appear on the environment.

To do so, we have developed a solution based on a combination of traditional and deep learning-based image processing. Our system will extract the images from two different sequences, use traditional image processing to align them as geolocational information is not available in our data. After that, both images are segmented into smaller sections. The mentioned sections are introduced into our deep learning model.

Finally, the model generates bounding boxes which contain the regions where changes have been produced, alongside with a binary image with the modified regions remarked.

1.2 OBJECTIVES

The aim of this project is to detect and indicate the changes produced between to sequences of images acquired from a drone's flight based on the prediction generated by a deep learning model. Our final objective is to create a system able to perform automatic change detection on moving cameras to prevent security risks.

This goal includes smaller tasks such as:

- Image alignment and segmentation into smaller sections.
- Creation of a deep learning model architecture based on Convolutional Neural Networks.
- Dataset obtention for training the mentioned model.
- Analysis of results from the training process.
- Comparison with state-of-the-art implementations on change detection.

2 TECHNOLOGIES AND STATE OF THE ART

2.1 DIGITAL IMAGE PROCESSING

Digital image processing refers to a certain group of algorithms that aim to obtain information from a digital image, as well as transform it to obtain a different representation of it which may include additional data.

A scheme of the multiple applications can be found in Figure 2.1.

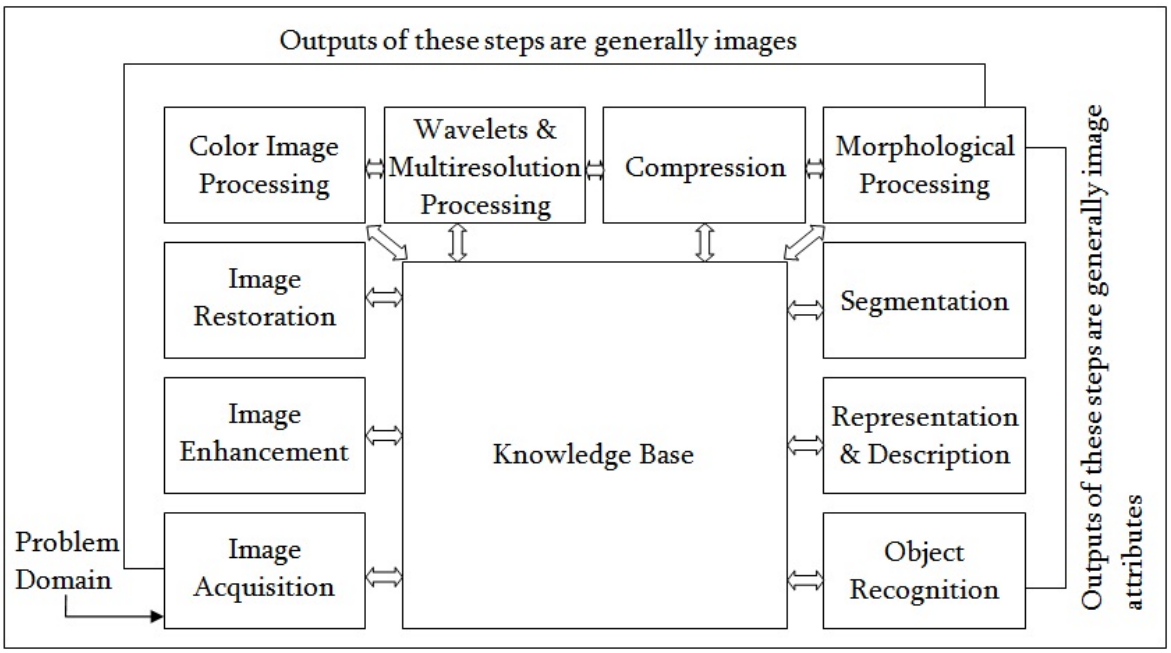


Figure 2.1 Image processing applications

Source: [1]

2.1.1 Image descriptors

Visual information and metadata contained in image and video sequences are organized in structures defined as descriptors. These visual descriptors can be automatically obtained and used in image processing or computer vision processes.

These descriptors can be separated into two groups:

- General descriptors: Provide data about basic features, such as brightness, color, shape, corners, regions, texture, position, movement.
- Specific descriptors: Reflects about the objects and events that are included in the scene, such as face recognition.

The combination of the information provided by general descriptors can conform more complex information about the image. Different methods, such as [3], [4] and [5], have combined multiple of these values to obtain relevant information from the images.

2.1.2 ORB descriptor

ORB (Oriented FAST and Rotated BRIEF) descriptor [4] is one of these image descriptors. Its name comes from the two different descriptors it is based on: FAST (Features from Accelerated Segment Test) [6] and BRIEF (Binary Robust Independent Elementary Features) [3].

ORB extracts points with valuable information by applying the FAST algorithm. This estimation is based on the Harris corner measure [7]. As FAST algorithm does not retrieve any information about the orientation, so this information must be obtained from another method. To do so, ORB employs a rotation matrix based on the computation of the intensity centroid [8] from the moments.

The moments are obtained using:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

Equation 1 Moments

Where *I(x, y)* represents pixel intensities, *x* and *y* denote the coordinates of the pixel and the variables q and p indicate the order of the moments. The centroid is obtained from the previously calculated models as:

$$C = (\frac{m_{10}}{m_{00}} \frac{m_{01}}{m_{00}})$$

Equation 2 Centroid

Finally, a vector from the corner's center to the centroid is constructed. The orientation is calculated using Equation 3:

$$\theta = atan2(m_{01}, m_{10})$$

Equation 3 Orientation from ORB

Where the *atan2* operator is the arctangent's quadrant-aware version given by Equation 4:

$$atan2(x,y) = \begin{cases} arctan(y \cdot x^{-1}) & x > 0, \\ arctan(y \cdot x^{-1} + \pi) & y \geq 0, x < 0, \\ arctan(y \cdot x^{-1} - \pi) & y < 0, x > 0, \\ \pi/2 & y > 0, x = 0, \\ -\pi/2 & y > 0, x = 0, \\ undefined & x = y = 0. \end{cases}$$

Equation 4 Arctangent's quadrant-aware version

With this approach, the ORB descriptor is able to perform two orders of magnitude faster than other descriptors with orientation information such as SIFT (Scale-Invariant Feature Transform) [5].

2.1.3 OpenCV

OpenCV [2] constitutes one of the most well-known libraries for computer vision and image processing. In recent versions, it has included several machine learning and deep learning methods for previous technologies.

This library contains interfaces for multiple programming languages such as Python. C, C++ and Java.

OpenCV is composed of different packages aimed for different processing objectives: *features2D* for extracting image information, *calib3D* for 3D image and video processing, *ML* and *DNN* modules for machine learning and deep neural networks processing, etc.

2.2 DEEP LEARNING

Deep learning is a machine learning technique based on the use of a concrete mathematical concept named Neural network. Due to its potential in a variety of applications, deep learning has revolutionized the technological industry.

2.2.1 Neural networks

Based on an explanation from [8], neural networks are the main mathematical structure of the deep learning technology. They are built on the aggrupation of nodes, with interconnections among each one. These groups of nodes are defined as layers. Each layer consists in a group of nodes with a predefined operation. The size of the layer varies depending on its position on the global architecture, its purpose and other different factors.

The neurons or nodes are the minimal computational unit of computation of deep learning. In these units, the input data is modified by a group of parameters defined as weights. The purpose of weights is to adapt its value to obtain the desired output. An additional element of this unit is the bias, an independent term of the input samples. The combination of the product of the weights and the input values with the bias value are the inputs for the function which will obtain the output of the node. This function is known as the activation function, which is different depending on the purpose layer and thus on the node's objective.

In Figure 2.2, a detailed diagram of a node is depicted.

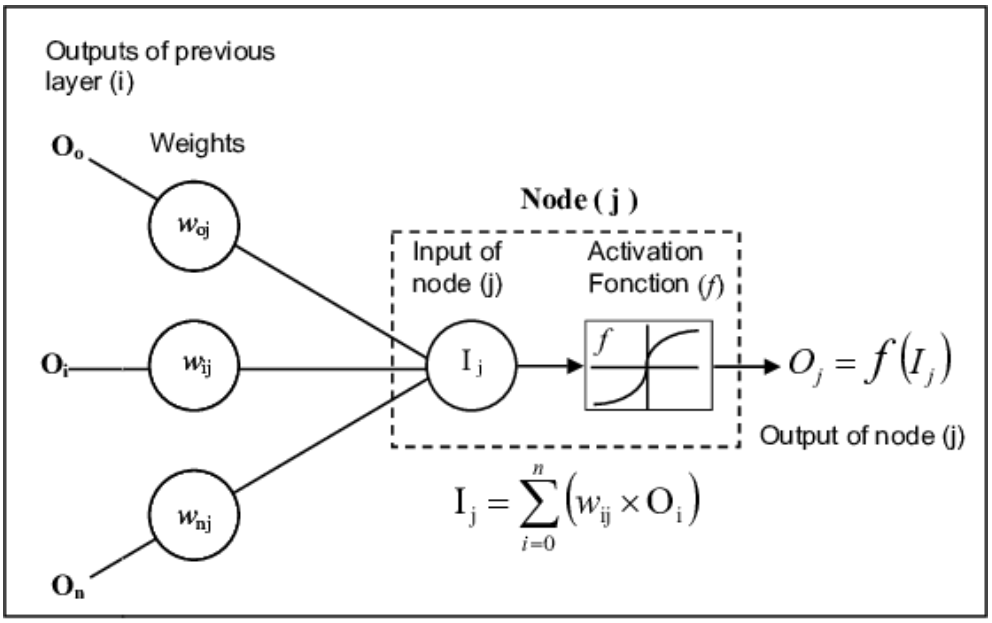


Figure 2.2 Neural node diagram

Source [34]

As stated before, nodes are grouped in layers. The output of a layer is the input of the next layer in the neural network's architecture and its output will be the input for the following one and so forth. The concatenation of these layers composes a neural network. Figure 2.3 depicts an elementary neural network in order to illustrate the explanation.

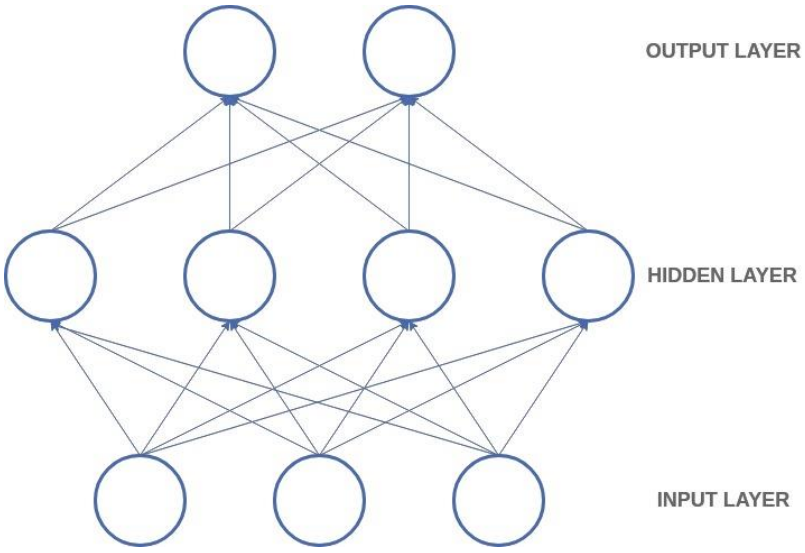


Figure 2.3 Neural Network schema

In the next paragraphs, different deep learning concepts used along the document are going to be explained in detail.

Loss

The base objective of the deep learning problem is to minimize the value of a specific function as an optimization problem. This function is referred to as the loss function. This function calculates the difference between a prediction performed by the algorithm and the known value of the output of this sample, defined as the label or ground truth value. As an example, Equation 5 illustrates the binary cross entropy function. This function is commonly used in deep learning models which try to solve a binary classification problem. In our case, we want to find the changed and unchanged zones in an image. Therefore, this function is an interesting option to be used in our implementation.

$$Loss = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

Equation 5 Binary cross entropy loss

Where:

- N is the number of samples
- y_i is the known output value for the i-esim input sample
- $\hat{y}_i$ is the model's prediction for the i-esim input sample
- *log* is the natural logarithm operation.

Optimizer

The purpose of the optimizer is to apply an additional updating operation to neural network's weights. The objective of this operation is to avoid an excessive adaptation the mentioned parameters to the data. Therefore, the model would obtain a more general relation of the features.

Different optimization algorithms are used based on the problem's condition. For example, Stochastic Gradient Descent [9] is commonly used in optimization.

In our case, we have selected the Adam optimizer [10]. This selection is based on previous experience with other deep learning implementations, along with the performance improvement from other optimizers. A visual comparison between different optimizer is shown in Figure 2.4.

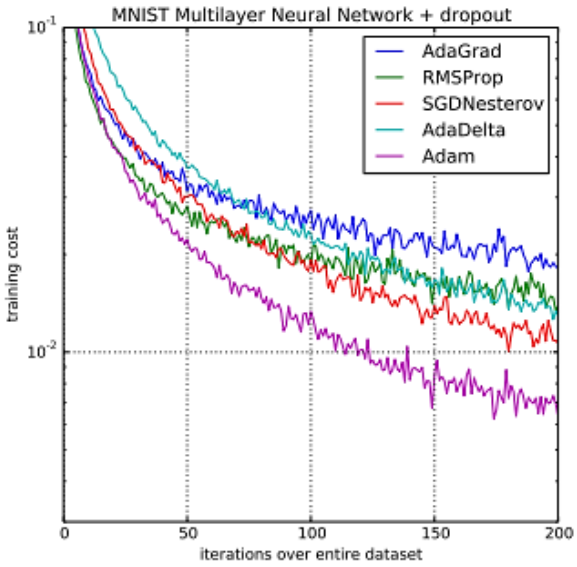


Figure 2.4 Adam optimizer comparison

Overfitting/Underfitting

The concept of overfitting denotes a circumstance where a machine learning model is very precise for the training dataset, although it provides low accurate results for any other data. This phenomenon could occur for multiple factors:

- Small dataset with similar samples.
- Excessive number of layers and neurons for the target problem.
- Excessive importance of a certain feature.
- Absence or lack of effectivity of the regularization terms.

And many other factors. For each problem mentioned on the previous list, a solution can be listed as follows:

- Increase data and perform data augmentation to add variety to the data.
- Decrease the complexity of the model and observe the results.
- Add normalization layers.
- Test different optimizers.

This problem is a very common one on most of the deep learning projects. As the goal is to obtain a model as precise as possible for a variety of situations, the management of overfitting is a crucial factor to obtain a valuable development.

In case of underfitting, the model does not have enough information to obtain significant features to predict successfully. In this case, we can find factors such as a very varied dataset with small amount of data, a model with a simple architecture which cannot provide insights from data.

Dropout

Based on the concept detailed in [8], dropout layers conform a regularization technique. Regularization processes try to avoid the overfitting and underfitting situations described before. In case of dropout, the system deactivates a percentage of the neurons from a specific layer. These neurons are randomly selected after each iteration to keep a uniform regularization. As a result, the weights do not overfit to the data as they are able to extract only a part of the whole information.

A scheme of a dropout layer in a simple neural network is defined in Figure 2.5.

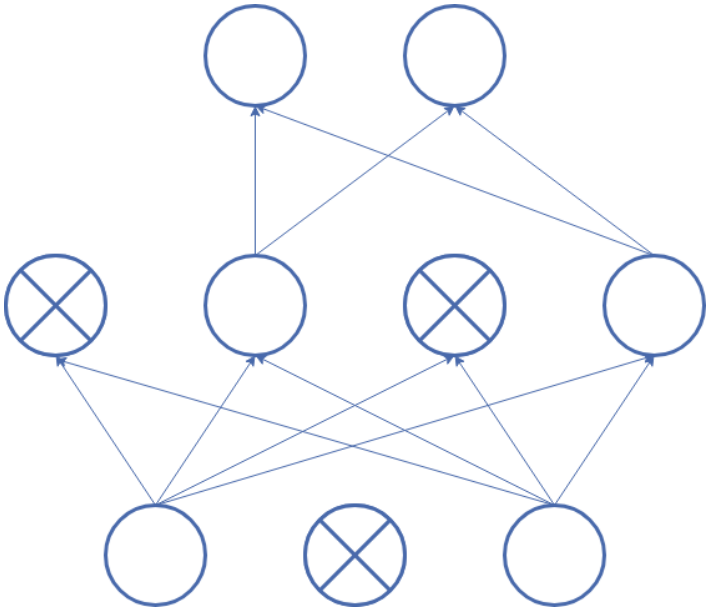


Figure 2.5 Dropout

Batch Normalization

As the previous structure, batch normalization [9] aims to avoid an overfitting situation. In this case, the normalization is performed by subtracting the mean of the input values and dividing the results by the standard deviation. The equation of the batch normalization is as follows:

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

Equation 6 Batch normalization

Denoting μ_b as the average value of the batch, σ_b as the standard deviation of the batch and ϵ as a constant added for numerical stability.

This layer can be combined with a dropout operation to improve the model generalization. As can be observed from the two previous points and the current one, the overfitting phenomenon has a notable relevancy in the deep learning field. Other techniques have been studied to avoid this situation, although they are out of the scope of this document.

Batch size

We can define this value as the number of input samples that are stored in memory at the same time during a prediction, training or testing process. Each batch iteration represents a step of training. The objective is to maximize the batch size to reduce the training time of the model. The variations on this parameter could affect the model, although it depends on the problem and has not been studied for this project.

Epochs

This parameter represents the number of times the model goes through the complete training and test datasets. Based on the datasets size and the computational resources of the development, the number of epochs could vary.

Metrics

These parameters represent the different formulas to measure the model's performance. Metrics can vary depending on the process and purpose. For training, simpler metrics such as the loss function or the accuracy are used. In case of a deeper study, such as reporting the results of the model on specific conditions, more complex and concise measures can be made.

Early stopping

Due to the implications of the epochs' value in terms of time and computational resources, several strategies have been developed in order to optimize both. One of the mentioned strategies is early stopping [10], which is a widely used technique. It is based on the definition of a monitor metric such as the explained before. During the training process, an analysis of the results of the established metrics is made. If after a defined number of iterations, the parameters do not improve a certain threshold value, the system decides that the training process is finished as the improvements made are not significant. Although it can be a very useful technique, the thresholding parameter should be conservative to prevent a premature end of the training phase.

2.2.2 Convolutional Neural Networks (CNN)

As stated in [12], CNNs are a specific structure of neural networks. The main feature of these networks is that they apply the convolution operator to the input samples.

The CNN is formed by three different types of layers: convolutional, activation and pooling layer. In the following paragraphs we are going to explain in detail the purpose of each layer.

Convolutional

In this layer, a dot product operation is performed between each region of the input value and a filter of a defined sizes. These filters, also known as kernels, defined the dimensions of the region of application. By applying this operation, the layer obtains the most significant features of each zone of the input image. The number of filters, along with its size, vary on each architecture and its position on it. This is because, in deeper levels, we aim to obtain higher level features, while in the first layers, the lower level information is extracted. The resultant output depends on the padding parameter defined for the operation. Padding is defined as the process of filling the border regions of the input with different values such as zeros, mean values of near zones or do not add any value and modify the output size by the kernel's size.

Activation

Activation layers apply a function to their input, as explained in previous sections. These are the most common types of layer in deep learning, as they are the main component of a densely connected layer, which is the most used structure in neural networks. The aim of this layers is to characterize the features obtained from previous layers to extract the most relevant values. This process is applied using different functions which may vary depending on the purpose.

Pooling

As the final part of the CNN structure, the pooling layer downsamples the input data by a factor defined and with a certain criterion. Both parameters depend on the architecture. However, the kernel for performing the downsampling operation usually has size 2x2. For the criterion, the CNNs oriented for image processing usually employ the max pooling, where the output value is the highest from the kernel size. The purpose of this layer is to reduce the size of the input data, while maintaining the most information as possible to perform the next operation with the highest accuracy.

The concatenation of CNN allows the system to learn features from different complexity levels of the image. Therefore, a more precise prediction than traditional methods can be formed as they obtain a more detailed information. As a result, the use of CNN has supposed a significant change in the digital image processing field.

In Figure 2.6, a schema of a CNN is depicted.

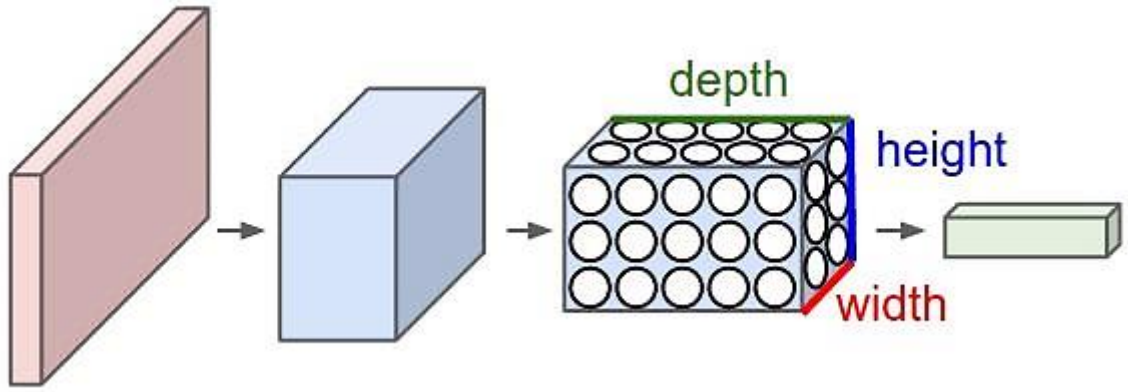


Figure 2.6 Convolutional Neural Network

2.2.3 TensorFlow

Developed by Google, TensorFlow [16] is a deep learning framework for building neural networks architectures based on the previous mathematical structures and graph theory. They are based on the concept of tensors, which are structures that can encapsulate elements such as scalars, vector matrixes and other multidimensional arrays.

TensorFlow is mainly used in Python language, although its API is built for multiple languages such as Java, C++, and JavaScript.

As the computational cost required for the mathematical operations performed by the library is high, most of the methods are implemented in high-performance binaries implemented in C++.

As can be deducted from previous sections, the neural networks contain high computational cost operations with multidimensional arrays as inputs. Therefore, TensorFlow is a framework designed for deep learning due to its features.

2.2.4 Keras

Keras [14] conforms a high-level Python API for deep learning development. This API aims to simplify the development process of the neural networks as the use of backends such as TensorFlow or CNTK increases the complexity for the developers. Keras has been integrated in the TensorFlow API for Python in 2018 to convert TensorFlow into a complete framework for deep learning developers with different grades of experience.

A simple scheme of the different components executing a Keras based deep learning algorithm on a computer is depicted in Figure 2.7.

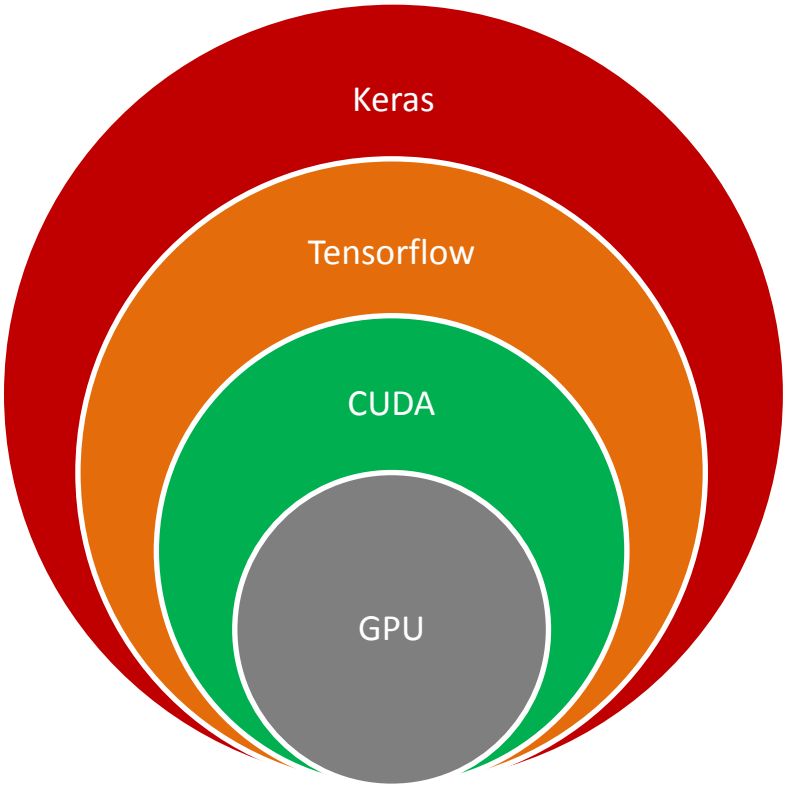


Figure 2.7 Deep learning computational schema

From the previous graph we can observe that two elements are on a lower computational level than TensorFlow: CUDA and GPU.

1. GPU stands for “Graphical Processing Unit”. As a critical component of computers, is related to the computation of information normally related to displays and other matrix information that contains a significant amount of information that could not be efficiently processed by a sequential system and requires to be allocated in memory.

2. CUDA stands for "Compute Unified Device Architecture". Developed by NVIDIA since 2006, this platform is oriented to parallel computation. Due to the benefits of parallel processing for the intense operations of deep learning, GPUs have increased their importance for this type of developments. Therefore, an intermediate component to connect the programming backends, such as TensorFlow with the physical processing devices has been required. CUDA optimizes the computational procedures of the neurons taking advantage of the parallel processing system of the GPU.

2.3 STATE OF THE ART

An exhaustive revision of different state-of-the-art implementations for detecting changes between two images has been made. A significant differentiation of the algorithms reviewed is the technology to develop them. First, we have analyzed systems based on traditional image processing techniques. These implementations have set the state-of-the-art for many years. Nowadays, they still have competitive results versus more complex implementations such as the ones based on deep learning.

Among the mentioned, we have reviewed implementations based on a Gaussian Mixture Model [15] as this concept is widely used. This method is founded on the intensity regions to estimate the background probability. Other interesting traditional methods are based on Kernel Density Estimation [18] .These methods are not able to model frequent elements. As mentioned before, the conditions of the drone's flight imply a dynamic background, where these methods perform poorly. More actual implementations such as SuBSENSE [18] and Pixel-Based Adaptive Segmenter (PBAS) [19] have been studied as they have supposed a significant improvement from other traditional image processing methods for change detection. As they lack higher-level information from images, the performance in complex environments of these methods is not sufficient for our purpose.

As stated on previous sections, convolutional neural networks have supposed a significant improvement in terms of image processing. Two methods have been studied in depth to obtain relevant insights about the most efficient architectures that conform the state-of-the-art nowadays.

In [16], both background and foreground images are sliced in patches with lower dimensions. The two patches are concatenated in the depth axis, as it has been studied that for comparison purposes such as ours, the convolution applied to images stacked depth-wise produces more appropriate results than in any other dimension. Concatenated patch is introduced to a CNN-based architecture, which returns a binary patch with foreground zones in white color and background zones in black. However, the restriction of having a moving camera is not solved as the system is meant for static cameras.

In [17], a combination of a VGG [18] architecture connected to a deconvolutional neural network is applied to obtain an image of the same dimensions as the input picture. To do so, the number of deconvolution layers is equal to the CNN layers in VGG. As a result, the model returns a binary image with the same size of the input image. The stated method does not employ a background image, only the foreground and, for training purposes, the ground-truth images. This has two different consequences: first, without a background reference, the system can be employed for moving cameras if the dataset is properly conformed. Secondly, the algorithm has to learn a different background for each scene. Therefore, the system will have to learn on each new scene, which slows processing time in real world applications. From the results of both methods it may be concluded that CNNs have improved considerably the accuracy of traditional change detection algorithms.

3 PROJECT DEVELOPMENT

In this section the complete development of our system is illustrated. In each subsection, a different component of the project is explained in detail. First, the image processing techniques employed along the project is explained. Then, the process to obtain our dataset is detailed. After that, the deep learning model used on the system is described. Finally, the post-processing methods applied to the model's output are detailed.

3.1 IMAGE PROCESSING

In this part, we will illustrate the diverse proceedings we have applied to the images in order to obtain an adequate input for our deep learning model. As we have pointed out on previous sections, traditional image processing is one of the two main components of the system in combination with our artificial intelligence implementation.

At our initial stage, we have two different video sequences which corresponded to two time spaced flights of a drone. It should be noted that both image sequences are not perfectly aligned neither in time nor in space. The former is a consequence of the fact that the UAV's image acquisition process is not related to any geolocational information and it's activated manually. The latter is a result of the environmental effects to the drone's flight, such as the influence of wind.

From the previous analysis we can deduct that our images will not be completely aligned in a real scenario.

To solve this difficulty, we have used image descriptors such as ORB [1] to obtain the relation between both images and perform an image geometric transformation to generate an aligned version of the image where we want to detect changes.

3.1.1 Obtainment of image descriptors

As mentioned before, we have based our alignment algorithm in the obtainment of the ORB descriptor in both the foreground image and the background image. To do so, we have applied different methods from the OpenCV library. Our first step is to transform our image into a grayscale color space, i.e. modify our images to have only one-color channel instead of the three original channels of the RGB format.

Subsequently, an ORB descriptor detector is created using the *ORB_create* method from the library. The only parameter specified for this object is the number of features. To limit the computational cost of the computation, the parameter has been defined as 500.

After that, the detector is applied to each image independently, using the *detectAndCompute* method from the mentioned class. With that, we obtain the points with the most significant information in terms of the ORB descriptor. Our next step is to find the "matches" between both descriptors. That

is to say, the points that are common between the two pictures. For this purpose, we have employed another object from the OpenCV library, the *DescriptorMatcher*. Following that, we have sorted our matched points in terms of distance. As a result, we have obtained a list of descriptors. These descriptors are represented in both pictures and ordered by the distance between. Finally, we have filtered our list in order to obtain only a percentage of the complete list. With that, we have used only the most representative descriptors of both images. The objective of that decision is to reduce the computational cost of future transformation while losing only the points with less information.

3.1.2 Image alignment

To obtain the aligned foreground image from the points containing the descriptor of the previous subsection, a geometric transformation of the image is needed. More precisely, we need a perspective transformation known as homography. This comes from the fact that the orientation of the drone's camera could have changed from one flight to another. Therefore, the perspective of the same zone could have been modified and a correction of that is needed.

From the OpenCV documentation: "The functions find and return the perspective transformation H between the source and the destination planes, so that the back-projection error is minimized".

$$s_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Equation 7 Perspective transform

Where the original points are defined by *(xi, yi)* and the output points by *(x_i', yi')*.

The back-projection error is defined by Equation 8:

$$\sum_i \left(x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

Equation 8 Backpropagation error for perspective transform

Where each h_{ij} element represent the value of the element at row *i* and column *j* of the matrix *H*.

The documentation from OpenCV points out how the refinement of this operation is performed:

"However, if not all of the point pairs fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor.

In this case, you can use one of the two robust methods. Both methods, RANSAC and LMeDS, try many different random subsets of the corresponding point pairs (of four pairs each), estimate the homography matrix using this subset and a simple least-square algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC.

Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more."

After this complete processing is applied to our points, we have obtained a matrix H which let us transform our image into a version of it aligned to its reference, i.e. the background image. To apply this transformation, the function *wrapPerspective* from OpenCV has been used.

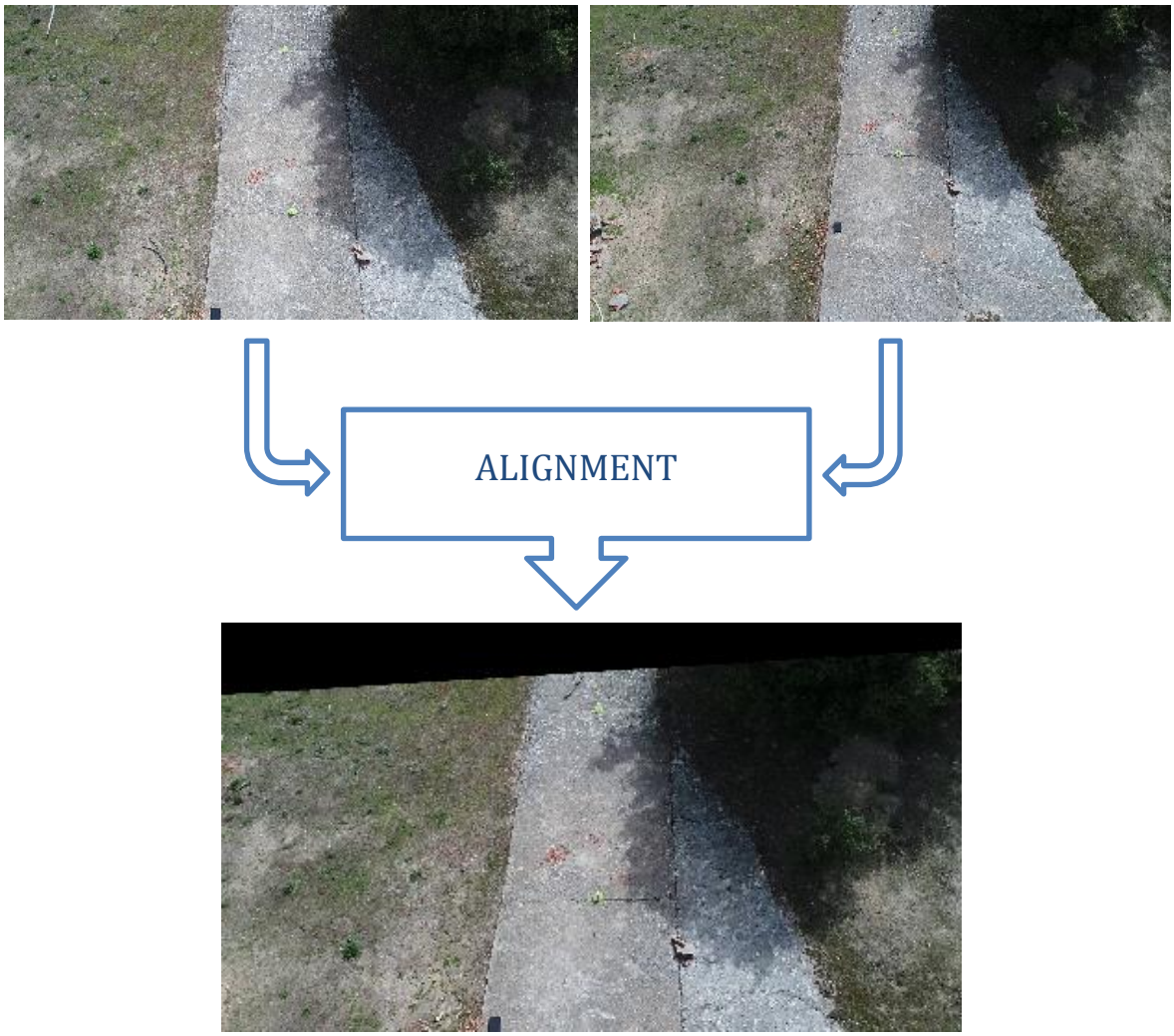


Figure 3.1 Alignment process

3.1.3 Image segmentation: Sliding window

The image provided to our system could vary in size depending mostly on the drone's camera device or the time processing requirements. For instance, the processing requirements of a real time detection systems are different than those of a post flight analysis. Moreover, deep learning models usually struggle to work with very high-resolution images.

Our approach to solve these problems is described in this subsection. First, we have defined the maximum image size to be a parameter of the system. If the input images overcome the defined dimensions, they are resized to the specified size.

In order to have samples with appropriate dimensions for our neural network, able to obtain a pixel-level precision of the change produced, we have to divide our images into smaller regions.

To do so, we have employed an algorithm named sliding window. This algorithm iterates over the image's dimensions, retrieving a matrix of a specific size (window) which contains a region of the original image. The dimensions of this window are equal in width and height in order to avoid any further complication of the segmentation process.

Another parameter of this algorithm is defined as the step. The step of a sliding window algorithm is the distance, in each dimension, from the starting point of a window in iteration *i* to the starting point of the region in iteration *i+1*. Adaptive to each dimension, the step is a crucial factor in terms of computational cost, so is the dimension of the window. If, for example, our window size is *SxS* and our step is defined by *S* for each dimension, then, no overlapping between windows is produced and therefore, the computational cost of this algorithm is the minimum for a given window size *SxS*. On a different case, if we have the same window *SxS* and our step is now *S/2 x S/2*, the computational cost of applying the algorithm to the same image has increased by a factor of 4 (2 for each dimension). The benefit of overlapping is the increase of precision on the analysis, as four predictions are made

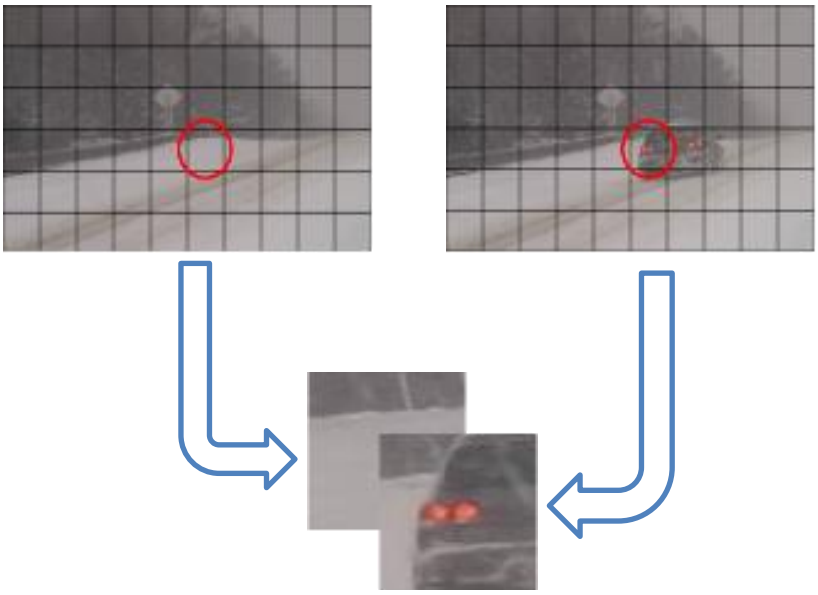


Figure 3.2 Sliding window example

of the most part of the image, except the limits of each dimension. As a conclusion, the variation of both parameters is a very useful tool that can modify the algorithm's performance in order to adapt it for different purposes: real time segmentation, maximum precision prediction or sample generation.

3.2 DATASET ADAPTATION

The development of this project has lacked the necessary resources to generate a dataset. The main reason is the absence of an available UAV device, along with the appropriate licenses to perform flights for image acquisition.

To solve this, we have used the CD2014 dataset [2], adapting it to our specific circumstances. This dataset contains images separated in different categories:

- Baseline
- Dynamic Background
- Camera Jitter
- Intermittent Object Motion
- Shadows
- Thermal
- Bad Weather
- Low Framerate
- PTZ
- Night
- Air Turbulence

As can be deducted from their names, each category contains a set of pictures for a specific condition. In our case, we have employed the categories for training and testing processes: Bad Weather and Dynamic Background. We have also used the Intermittent Object Motion category for our metrics.

The election of these categories has been based on the fact that the represented conditions are the ones that approximate the most to our real conditions. In case of Bad Weather category, adverse environmental conditions such as a snowfall or a blizzard are represented. In case of the Dynamic Background, the conditions depicted are ideal for our case, as the image depict variable backgrounds. Finally, the Intermittent Object Motion category contains images where a displacement of a specific object is represented.

An example of background and foreground image from each category is represented in Figure 3.3 (Bad Weather: Blizzard subcategory), Figure 3.4 (Dynamic Background: Canoe subcategory) and Figure 3.5 (Intermittent Object Motion: Sofa subcategory).

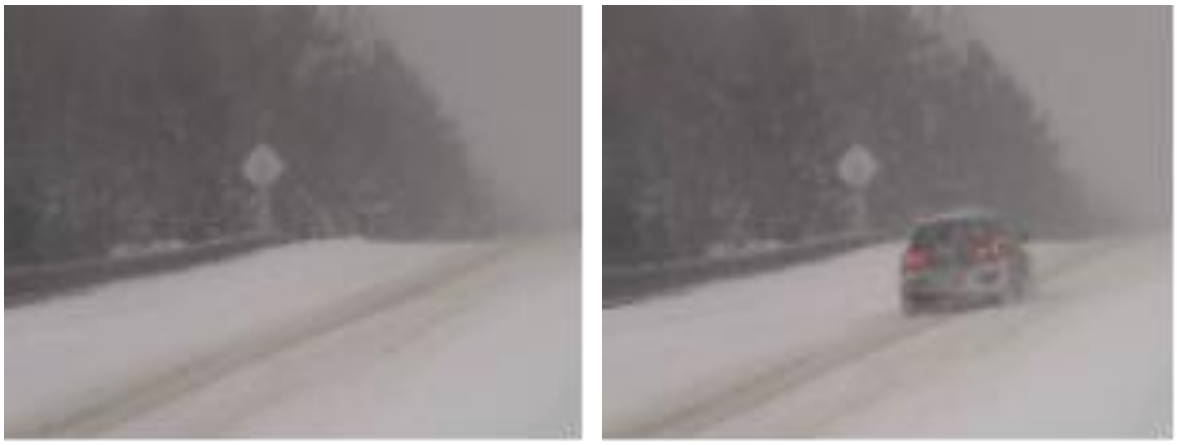


Figure 3.3 Bad weather category

Figure 3.4 Dynamic background category

Figure 3.5 Intermittent Object Motion category

The dataset contains input images, shown in previous figures, which may or not contain a change. In addition to that, each input image has a ground truth binary image such as the one represented in Figure 3.6.

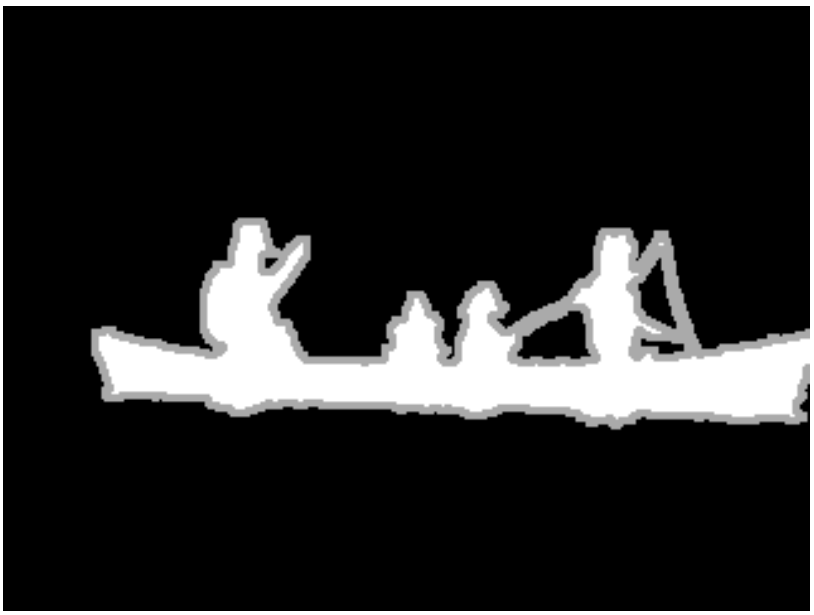


Figure 3.6 Ground truth image

However, not all ground truth images contain the labeled changes, probably for testing purposes for the challenge related to the dataset creation. In our project, we have only selected a set of pictures which had a labeled ground truth image.

One of the core insights of our project is the idea of having a reference video for the background. The original dataset does not contain a set of background or reference images. To solve this issue, we have selected a picture to act as a reference. This picture is selected based on the fact that its ground truth image does not include any white or modified region. The elected image is replicated for each foreground image, with the same identification number to keep a correspondence between each pair of images. In case of the Dynamic Background category, the Canoe subset of images contains a background scenario with very slight variation. This is an ideal condition for training our model, as small changes such as variations of leaves should be ignored. In the rest of categories, the reference camera is on a fixed location. Therefore, a unique image is replicated for each foreground image.

In order to test different segmentation sizes, we have defined our model to have a multiple of 2 as value for each dimension of the segmented regions. Consequently, images should be resized in order to avoid any error in the application of the sliding window algorithm described in section 3.1.3.

The condition of having small sized images, i.e. 64x64 or 128x128 pictures, implies that our dataset images should be segmented. To do so, we have to obtain patches of each image. We define patch as a smaller region, with a size of the window specified for the sliding window algorithm, which has been contained into an independent image file. These patches should be properly named based on their location in each image, along with the name of the image they belong to. Following the

nomenclature: filename_slidingWindowLocation.jpg, each patch is uniquely identified and the position in an alphabetical order of each folder is the same for the foreground and background patches.

To perform this process without any error, we have resized our images to make sure that their dimensions are a multiple of the input size of the model, in our case 64x64.

The final distribution of the images, their final size and the total number of patches for each category used are represented in Table 1.

Category	Images	Size	Total Patches
Blizzard (BW)	195	1216x704	40755
Skating (BW)	177	1216x704	36993
Snowfall (BW)	192	1216x704	40128
Canoe (DBG)	210	640x384	12600

Table 1 Dataset

3.3 NEURAL NETWORK

3.3.1 Architecture

The complete code to generate, train and test a model is encapsulated on a unique script.

To do so, we have employed the Keras Functional API [2]. This section of Keras' code allows the developer to generate more complex architectures, such as multiple input, Siamese networks, etc. Moreover, it permits lower level modifications to the network.

The structure of the architecture is defined as follows:

- First, we define an input layer to handle the samples introduced. As we have two images as input (foreground image and background image), we need to combine them in order to let the model find the differences between them. In our case, we have established an input with the next shape:

  [Image width, Image height, 6]

  As can be easily deducted from section 3.2, both input images have the same dimensions. Then, both images are combined in the depth dimensions, which represents the color

channels. As the input images are RGB, this union results in a 3-dimensional tensor with the shape defined above.

- After the input layer, a sequence of four Convolutional Neural Networks as the ones detailed in section 2.2.2 are applied to the input samples. Following the structure defined in the mentioned section, each convolutional network contains the next elements:
    - Convolutional layer: With an increasing number of filters for each subsequent CNN in order to obtain information for higher level features progressively. The range of filters are 24 for the first layer, 48 for the second, and 96 for the third and fourth layers. The election of this values has been done based on other state-of-the-art implementations such as [3] and [4].
    - Activation layer: With a Rectified Linear Unit (ReLU) function [5], this layer removes the values which contains the less information in order to perform the dimensional reduction of the pooling layer without losing significant information from the features extracted at the convolutional layer.

$$R(z) = max(0, z)$$

Equation 9 ReLU activation function

    - Pooling layer: As we have said in previous sections, the aim of this layer is to reduce the size of the matrix to decrease the computational cost of the following operations, such as the next convolutional layers or the densely connected layers.
- The output tensor of the last CNN has a size of 4x4x96. To perform the next operations, we need to convert it into a one-dimensional vector. To do so, we use the Keras Flatten layer.

When we have our vector, we apply two different regularization [6] techniques.

- First, we apply a dropout layer with a 50% of activation [7]. With this layer, our goal is to reduce the number of activated neurons. Therefore, the model will tend to avoid overfitting the training data [8], as part of the neurons do not get any information.
- After the dropout layer we apply a batch normalization layer [9] with the structure described in section 2.2. The purpose of this layer is to normalize the input values, in order to avoid that any outlier weight could corrupt the performance of the model and, as mentioned before, overfit the model to a specific feature.
- Finally, our desired result is a one-channel image such as the ground truth images shown in previous sections. As we have defined that our output will be a 64x64 image, we are going to establish our output according to this image, yet it has been defined as a one-dimensional version with a length resultant of the multiplication of the sizes for each dimension:

$$Output\ length = 64x64 = 4096$$

To obtain our result, we have employed a sigmoid activation function, such as the defined in section 2.2, with an output size of 4096, as can be easily deducted from the previous analysis.

The resultant vector will be postprocessed, following different methods that will be explained in further sections.

Table 2 gives a more visual perspective of the model's architecture.

In addition, we have included Figure 3.7 to provide another visual view of the model.

Name	Type	Input size	Kernel	Filters	Stride	Output size
Conv2d1	Convolutional + ReLU activation	64*64*6	3*3	24	1	64*64*24
Pool1	Max-pooling	64*64*24	-	-	2	32*32*24
Conv2d2	Convolutional + ReLU activation	32*32*24	3*3	48	1	32*32*48
Pool2	Max-pooling	32*32*48	-	-	2	16*16*48
Conv2d3	Convolutional + ReLU activation	16*16*48	3*3	96	1	16*16*96
Pool3	Max-pooling	16*16*96	-	-	2	8*8*96
Conv2d4	Convolutional + ReLU activation	8*8*96	3*3	96	1	8*8*96
Pool4	Max-pooling	8*8*96	-	-	2	4*4*96
Flatten	Vectorization	4*4*96	-	-	-	1536
Bn	Batch normalization	1536	-	-	-	1536
Dp	Dropout	1536	-	-	-	1536
Final	Sigmoid	1536	-	-	-	4096

Table 2 Description of model's layers

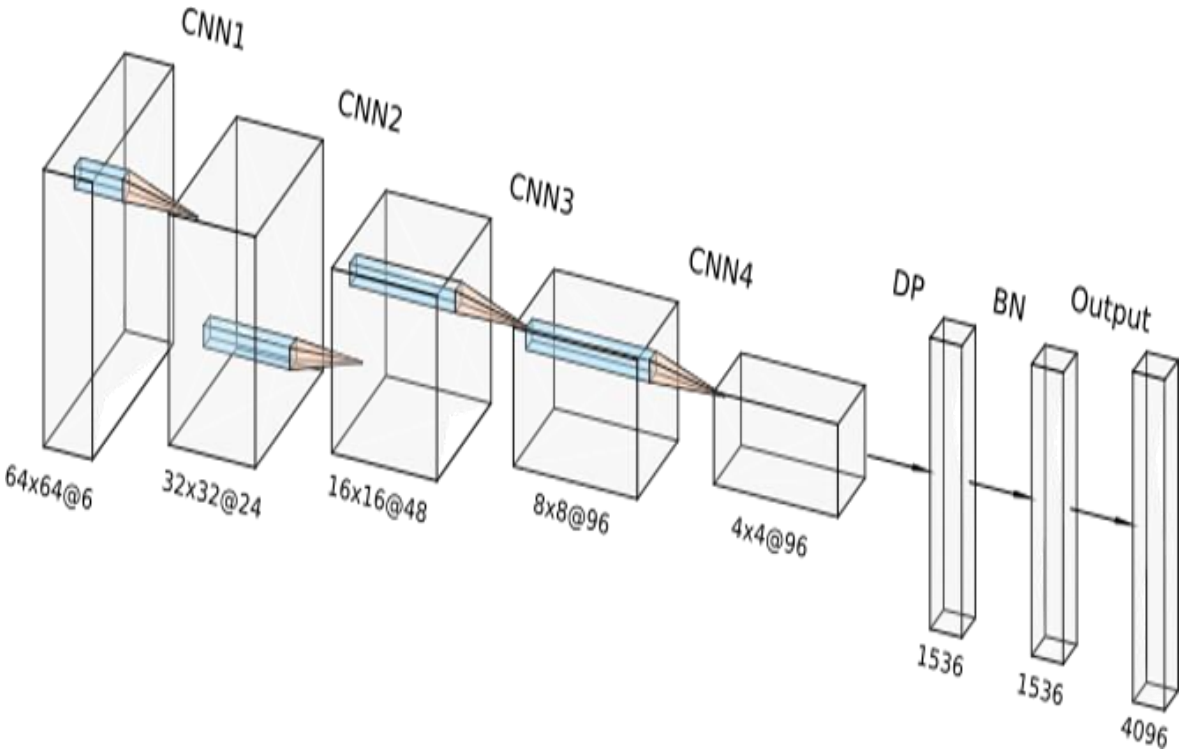


Figure 3.7 Model architecture in 3D

3.3.2 Training and test processes

To train a new model, we have to define the next parameters:

1. Dataset folder: Path to the root directory which contains the three different categories: foreground, background and ground truth images with size 64x64, resultant of the processing described in section 3.2.
2. Test split: Percentage of the complete dataset to be used for testing purposes. We have established that, due to the size of our dataset, a 10% of the complete dataset is destined for testing. However, to have this value as a parameter has granted us the capability of trying with different test sizes in order to obtain the best results.
3. Batch size: As mentioned in section 2.2, defines the number of samples to be loaded in memory for each training/testing step. As said, the higher this value gets, the faster the training will be. However, as we have to load 3 different images (2 for input, one for output) for each training iteration, batch size modification affects dramatically the resources used for training.
4. Epochs: Number of times we want our model to iterate over the complete training and test datasets.
5. Augmentation: A common target of deep learning projects is to obtain the most quantity and variety of data possible. If we desire to improve both, Keras provides a class named ImageDataGenerator for the purpose. This class permits to apply augmentation to data in

order to obtain better results [10]. Data augmentation consists on employing different transformation methods to our samples in order to retrieve modified versions of the images, increasing the size of our data and its variety. In our case, the following data augmentation techniques have been employed:

a. Rescaling with 1/255 proportion for normalization of pixel values.
b. Rotation up to 72 degrees from the original position.
c. Horizontal and vertical flip of the images.

These groups of modifications have been selected for a main reason: As our input and output are images, the application of these augmentations has to be moderate in order to maintain the location of the changes in the ground truth images consistently along the data.

6. Monitor: Metric to evaluate the performance of the model after each epoch. In our case we have 4 different metrics to use: training accuracy, training loss, test accuracy and test loss. It is recommended to use the test metrics as they provide a more unbiased perspective of the model's performance. Depending on the metric chosen, we will want to maximize the value (in case of accuracy) or minimize it (when we are using loss as monitor).

After these parameters are defined, the model is ready to be defined using the architecture established in section 3.3.1. Nonetheless, we have to define the preprocessing needed to feed the correct samples to the model. In this architecture, we have to obtain the correspondent 64x64 foreground image for each 64x64 background image. Due to the data generation methods described in section 3.2, the samples are ordered perfectly. Despite that fact, Keras employs a specific method to train with large datasets, named as *fit_generator*. This function requires a generator object, which is a very common structure in Python development [11]. This object iterates over a larger list of elements, retrieving a group of them of a specific size in order to avoid issues related to memory overload.

We generate one generator for each folder: foreground, background and ground truth. Each one of these generators has been assigned to have the same seed. As a result, the random selection of samples, which is necessary to avoid overfitting to the dataset, is the same for the three cases. Consequently, the order of the samples stays consistent and the randomization is applied without any problem.

Nevertheless, we have to define a unique generator for the training process. This generator is a combination of the three generators explained previously.

To do so, we are going to iterate over the three generators at the same time using the *itertools* library from Python [12]. We are going to have an input number of samples equal to the batch size parameter defined at the beginning of the script. Our final input sample, as mentioned before, is a concatenation of the foreground and background image on the depth or color channel dimension. To obtain it, we

apply the *concatenate* method from the NumPy API [13]. Finally, we have to define our output or labeled batch of samples.

From section 3.3.1, the output of our neural network has been defined as a vector of 4096 elements. However, our initial output is a grayscale image of size 64x64 containing the changed regions in white and the unmodified parts in black. To obtain our desired vector, we also employ the NumPy API, applying its *flatten* method to convert the matrix into a NumPy array of one dimension. To retrieve our generator, we just have to concatenate our input and output into a tuple and define them as the output of our generator.

After that, we can establish our generator to be used as a parameter of the *fit_generator* function in order to begin with the training process. As stated before, we have defined a monitor parameter to obtain the model with better performance. In addition to that monitor, we have also used the Tensorboard package from TensorFlow. This package is composed by a web application which offers a visual analysis of the training process. It should be noted that we are able to check the performance of the model during the training or after the process has been completed by using this application. Therefore, the combination of the monitor with Tensorboard allows a complete control of the training process of the model. A sample view of the Tensorboard interface is depicted in Figure 3.8.

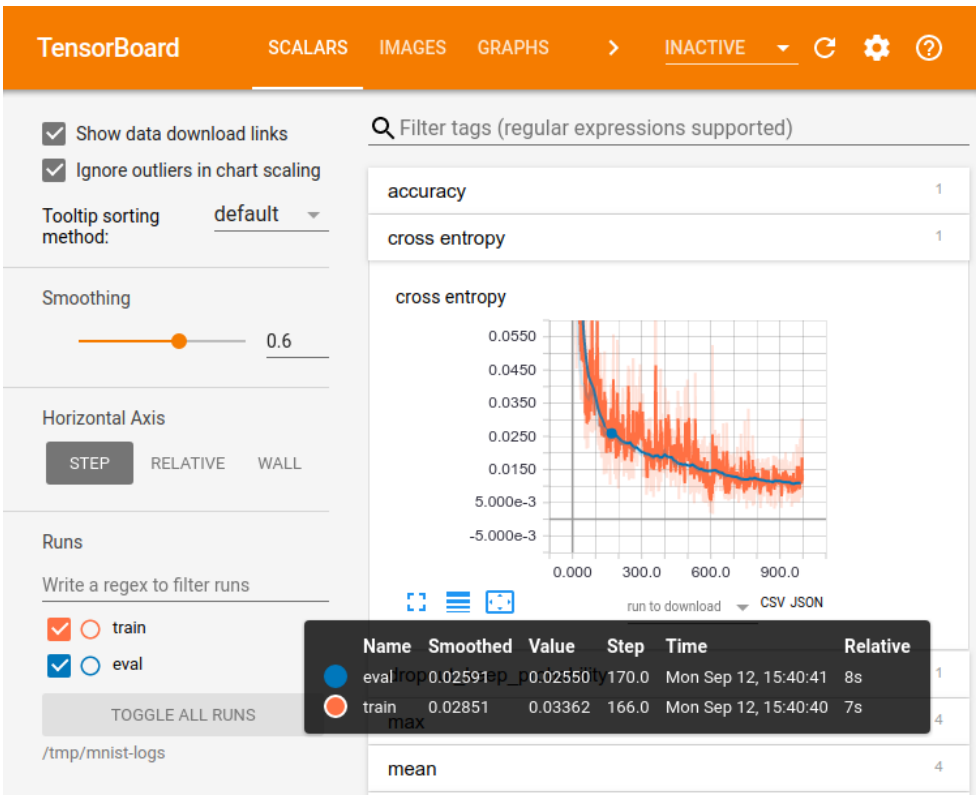


Figure 3.8 Tensorboard

Another important process is the storage and actualization of the model's weights after an improved version is retrieved from the training process. To perform this, we have employed the H5PY format, a file extension which is part of the Hierarchical Data Format [14]. The use of the monitor mentioned on previous sections is employed by a Keras callback named ModelCheckpoint. This function acts as callback, such as the Tensorboard application. This function obtains the results of the model at the end of each epoch. Then, compares it with the ones obtained by the best weights in terms of the monitor value, which have been stored in a .h5py file format. If the current weights are better than the best weights, we have different ways to update our weights. On the one hand, we can overwrite the file in order to reduce the occupied space and obtain a unique file after each training process. On the other hand, we can save the weights after each update in different files. However, this last practice is not employed as the use of testing metrics provides an unbiased indicator of the model's performance. If we had to use the training metrics, the mentioned practice could be useful as the best results on training could not imply directly the best results on other data.

The architecture of the model, i.e. the definition of layers and their sizes, is saved at the beginning of the training process using a JSON file format [15].

3.4 POSTPROCESSING

In section 3.1.3 we have explained the process the system applies to each image in order to split them into smaller patches which will be used as input to our model. However, our intention is to obtain a binary image of the complete area represented by the foreground image. To do so, we have to apply various postprocessing techniques to have our desired output.

As in training, the model predicts the images in batches of 32 input samples at most due to computational limitation. This batches are stored keeping their order of obtainment. This is a crucial fact in order to maintain the disposition of the image and avoid any error derived from a mistaken acquisition process. After that, the batches are introduced to the model, which predicts a batch of output vectors with the same size as the input batch size. These vectors are stored maintaining the sequential order, which is necessary to reassemble the image in a future step of the process.

As we have defined in section 3.3.2, the ground truth images are transformed into vectors using the NumPy API. Now, we have to perform the reverse action: patches of 64x64 are required to obtain the desired grayscale image with the modified regions. To do so, we apply the reshape method from NumPy, as it is defined as the inverse operator of the flatten mentioned before.

Finally, we iterate over the complete list of predicted batches, and complete a blank image with the same size as our input images to obtain the grayscale output with the predicted values for each pixel.

As can be deducted from the sigmoid function graph defined in section 2.2, the predicted values are in a range of [0, 1]. Therefore, the output image obtained from the previous process is a normalized grayscale image. Due to environmental conditions, such a temporal displacement which decreases

the precision of the alignment, noise on the image as a result of compression, disturbance during drone's flight, etc., the resultant values are filtered. To do so, a threshold function from the OpenCV library is applied. A value of pixel intensity is defined as a parameter to this threshold function. In our case, we have established that values higher than the threshold obtain the maximum intensity value, i.e. white. Pixel intensities lower than the mentioned threshold value are assigned with the minimum intensity value, i.e. black. As a result, we obtain a binary image containing the most significant changes predicted by our neural network.

Having this threshold as an adjustable parameter of the system provides flexibility to the implementations, providing the capability to adapt to different scenarios with variable conditions.

Because of the thresholding process, some zones, such as the border regions of changes could have been wrongly included as a background region. To solve this issue, we apply a morphological transformation to the image known as dilation [16]. These morphological operations are the result of an interaction between the original image and a structuring element or kernel. In this case, the dilation consists in giving a value of 1 if at least one pixel of the structuring element is 1. An example of these operation is depicted in Figure 3.9.

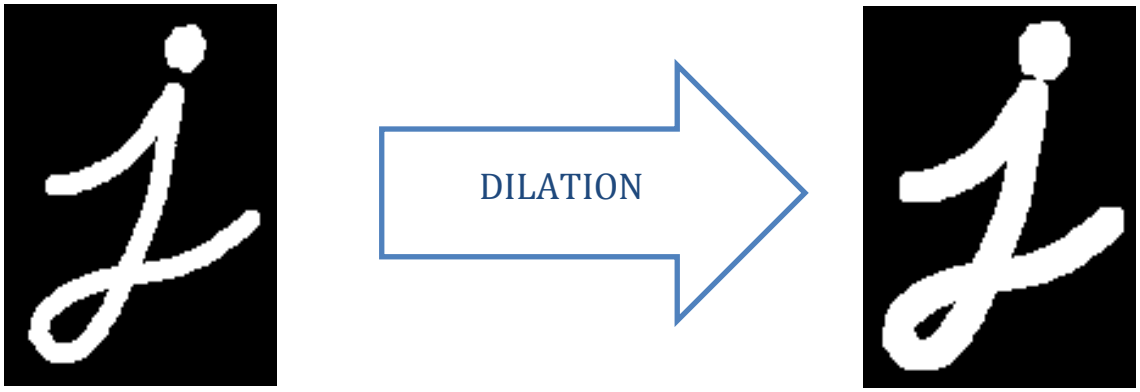


Figure 3.9 Dilation example

The dilation is defined by the next equation:

$$(f \oplus b)(x) = \sup_{y \in E}[f(y) + b(x - y)]$$

Equation 10 Dilation formula

Where f(x) represents an image, and the structuring function is defined by b(x). E is the Euclidean space into the set of real numbers.

In order to obtain a more visual representation of the modified regions, bounding boxes are employed. First, we obtain the coordinates and dimensions of the different bounding boxes from the results of the binary image. Then, these bounding boxes are drawn in our aligned foreground image.

4 RESULTS

In this section, the process followed to create the final deep learning model is detailed. In addition to that, the results from the training and test process are presented together with additional metrics to show a complete perspective of the model's performance. Finally, we have compared the results obtained with a selection of the state-of-the-art implementations studied on previous sections.

4.1 FINAL SYSTEM: CREATION AND PERFORMANCE

4.1.1 Final system creation

To create our final model, we have made several approaches to obtain a result that could compete with the state-of-the-art implementations. These methods are described in the next points.

Scene detection model

Our first approach is based on a deep learning model conformed by convolutional neural networks. In this case, we built a model to detect if an image corresponded to a certain scene or not. We made this approach to delimit the different background scenarios to perform the change detection. However, due to the nature of the system, the background scenario will be dynamic as it is going to be applied on multiple drone's flight. Therefore, we have to retrain it for each scenario, which reduces the flexibility of the implementation. In addition to that, because of the complexity of the scenarios, the system was not able to distinguish to similar scenes such as two different extensive cultivation areas. For these reasons, we did not used this approach as it did not fit our purposes.

Reconstruction model

During our extensive state-of-the-art review, we addressed that most of the implementations do not contemplate a scenario with a dynamic background. One of the few that considered this case was [31], where a reconstruction system is created in order to obtain a background representation for each scene. After the representation is built, using CNN and deconvolutional CNN, which performs an inverse operation from the previous. The objective of these types of CNN is to generate information from surrounding zones to obtain missing data. After the representation is produced, both the foreground original image and the generated background image are introduced into a CNN model which will define if the image contains a change or not.

We have built a similar approach to this one as the idea of obtaining the background from the original image would allow the system to be more flexible and independent. However, the deconvolutional operation generates images with an absence of sharpness. Therefore, the precision of the model would be focused on a system that would create representations with a significant lack of information. In addition to that, the component in charge of labeling the image as a modified region or an unchanged region obtains a unique label for the complete region. Consequently, the regions

must be small in order to avoid errors in parts that contain both changes and original regions. Finally, the use of a combination of two different neural network models increases the computational cost of our system. As the implementation could be used for real time detection processes, we have to optimize it to have the lowest computational cost as possible. For the aforementioned causes, we have put aside this approach.

Final system

After the previous approaches, we implemented our final system. It is based on the idea of having a reference sequence of the scenario where the changes are going to be detected as in [32]. This comes from the fact that the model needs to have a complete idea of the background scenario to detect the modified parts in a new sequence. As we have explained before, reconstruction implementations do not fit our purpose of having a detailed representation of our reference sequences. Therefore, we established that a background sequence must be acquired before detecting the changes. As mentioned before, drone's flights are subject to environmental conditions and other different factors that could affect the acquisition of image. As an example, the wind variations and the acquisition in a manual mode by using a human pilot could result into slight differences between a reference and a sequence with changes on it.

To overcome the previous difficulties, we have made a component whose objective is to align each foreground image from the new sequence to its correspondent frame on the reference sequence. Consequently, the acquisition of both sequences must be as synchronized as possible to reduce the influence of noise at the transformation in the foreground sequences. In our case, we lack from geolocation information of the sequences, which is contemplated in [32]. To perform the alignment, we have followed the procedures explained in sections 3.1.1 and 3.1.2. As mentioned on those points, we have obtained information from the most relevant zones of the image using the ORB descriptor. After that, we have applied a matching algorithm to collect the most similar zones of both images. Finally, we perform a geometric transformation to obtain the final aligned version of the new image.

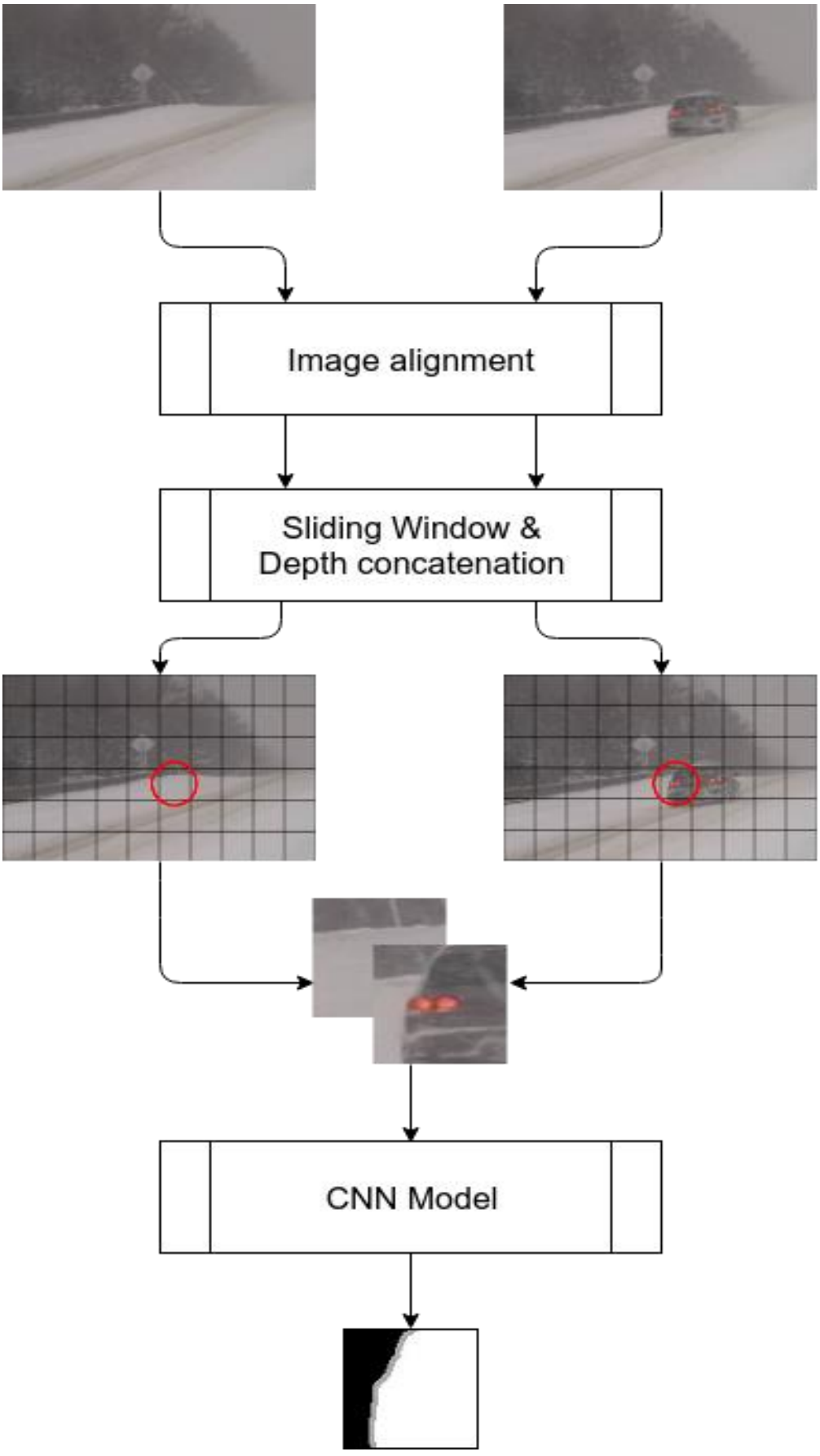
As we desire to acquire a precise detection of the changes, we have divided our image in smaller regions using the sliding window algorithm explained in 3.1.3. This algorithm is applied to both images at the same time to obtain the regions in the correspondent order.

After a smaller region from both sequences is retrieved from the sliding window algorithm, the two images are concatenated in the depth or color channel axis. Because of the image structure, the depth axis concatenation allows us to conserve all the information from both images and perform an analysis of their features at the same time. This united multidimensional array has a size of 64x64x6 as defined in section 3.3.1. The structure introduced to the architecture described on the section mentioned before. From this model, we obtain a vector of 4096 elements with values in range [0, 1]. Following the procedures detailed in section 3.4, the result of the deep learning model is processed

to obtain a 64x64 grayscale image. After the sliding window has iterated over the complete images, all the grayscale sections are collocated into a new image at the same position they have on the original foreground image. As a result, we have a grayscale image containing the levels of change for each pixel defined by the intensity value.

To avoid misclassification of different zones, we have defined a threshold filter for the resultant image. This value is variable for each sequence to be adaptable for multiple scenarios. The filter defines that the intensity values greater than the mentioned threshold are assigned with the maximum intensity value, i.e. white. The values with lower intensity than the threshold are assigned with the minimum intensity, i.e. black. Consequently, we obtain a binary image. Because of the filter's effect as explained in section 3.4, some regions could be abruptly separated. Therefore, we have performed a dilation operator to obtain more homogenous zones. Finally, we create a bounding box for each changed zone. The purpose of this is to remark the modified regions on the aligned version of the foreground image.

In Figure 4.1, a complete scheme of the system's architecture is depicted.



.

Figure 4.1 System's diagram

In Figure 4.2, a visual representation of the model input and output values is represented

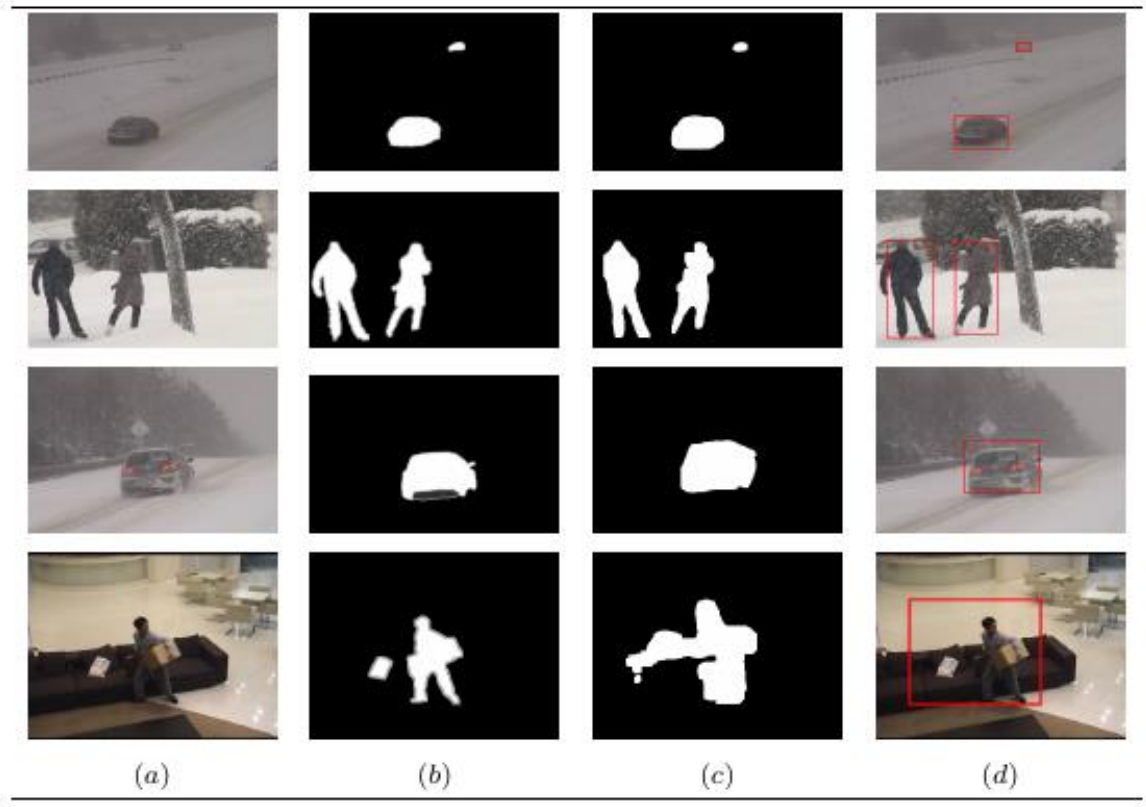


Figure 4.2 Visual results of system

Where (a) column represents an input foreground image for the categories: Blizzard, Skating, Snowfall from Bad Weather category and Sofa from Intermittent Object Motion, (b) illustrates the ground truth labeled image of each foreground picture, (c) depicts the output obtained by our proposed system and (d) represents the bounding boxed changes in the original image.

4.1.2 Image alignment effect on model's predictions

As we mentioned on previous sections, the component of image alignment has supposed a significant advance from the state-of-the-art implementations. The following figures depict the difference of results when the alignment system is used, as in Figure 4.3, and when it not is employed, as Figure 4.4.

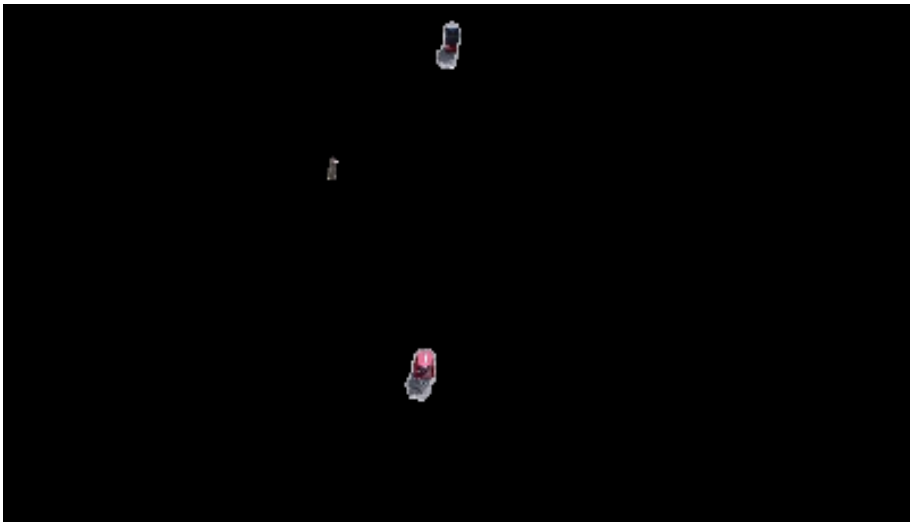


Figure 4.3 Aligned prediction

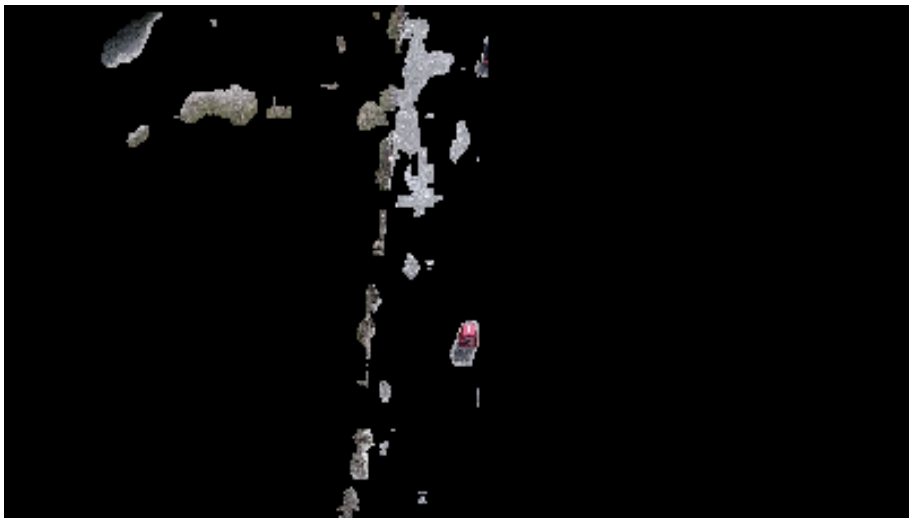


Figure 4.4 Unaligned prediction

4.1.3 Training and test results

In this point we are going to reflect the results obtained by the model's architecture described in 3.3.1 and the training and test procedures detailed in 3.3.2.

In Figure 4.5, the evolution of the loss results during the complete training processes is depicted.

Figure 4.5 Loss evolution in training

As can be seen, the loss parameter gets to low values after approximately 100,000 iterations. Moreover, the last iterations depict a slight increment of the loss function. Consequently, the training has been stopped due to the early stopping concept explained in section 2.2.1. Therefore, the model obtained the weights from iterations around the 450,000 steps, where minimum is achieved.

In Figure 4.6 and Figure 4.7 the results of the final training model acquired are represented.

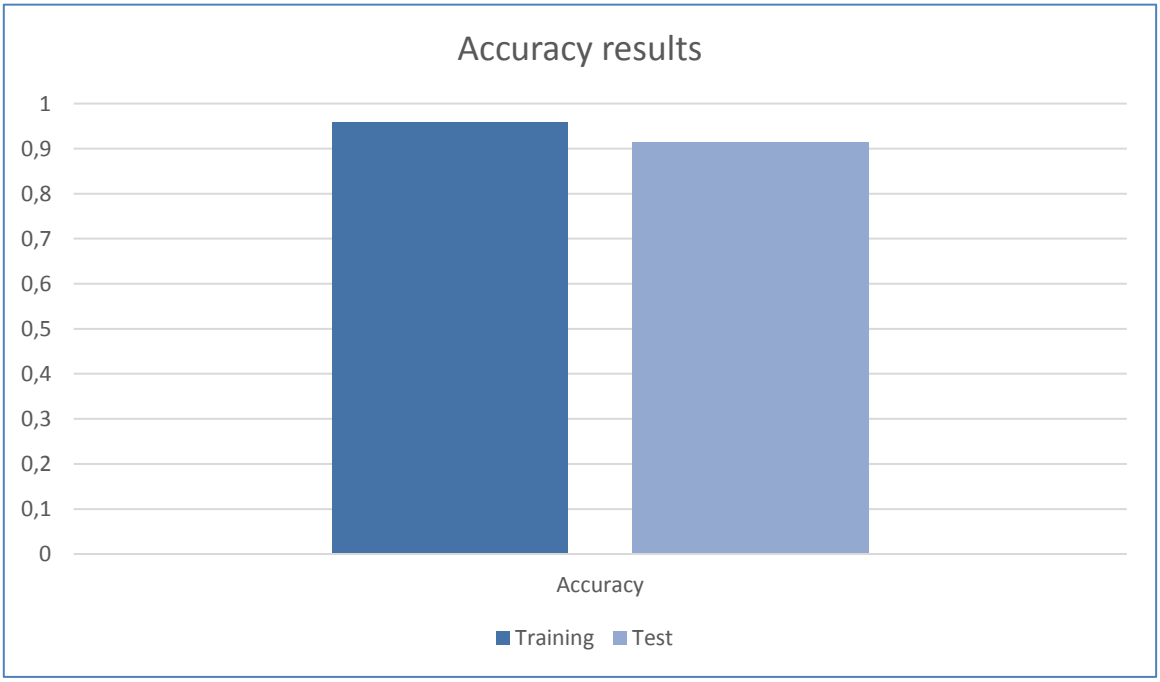


Figure 4.6 Accuracy results

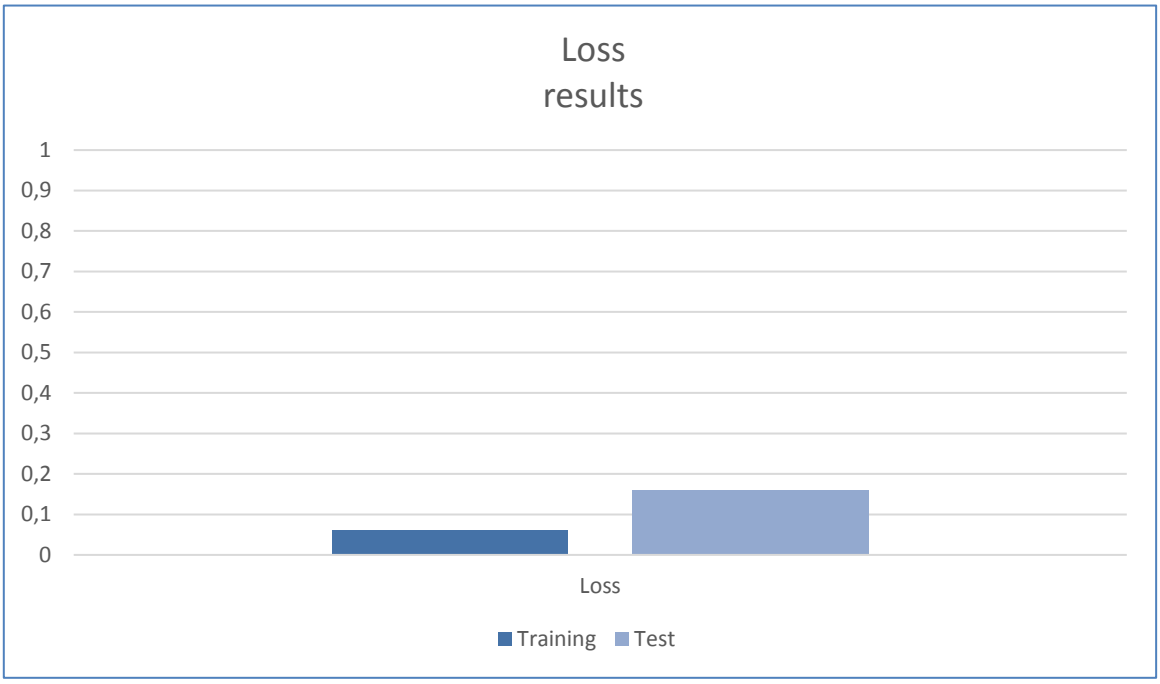


Figure 4.7 Loss results

From these results we can observe the excellent performance of the model in training and test data. This implies that overfitting has been avoided as the difference from metrics results between test and training is very small. In addition to that, the system obtains and accuracy over 90%. If we consider that each pixel of the images is analyzed, the relevancy of the results increases its significance.

4.1.4 Additional metrics

To provide a more precise perspective of the model's performance, we have employed additional metrics apart from accuracy and loss obtained from training and testing. These metrics are commonly used in deep learning projects to provide a complete report of the model.

These metrics are a combination of the following four concepts:

1. False Positives (FP): Defined as the pixels classified as changed which are labeled as background or unmodified values.
2. False Negatives (FN): Values classified as unmodified whose ground truth label is changed or foreground.
3. True Positives (TP): Pixels predicted as changed which are labeled with the same value.
4. True Negatives (TN): Values predicted as background or unmodified whose ground truth label contains that value.

With these four concepts, all the possible predictions are defined. The following list of metrics are a combination of these values:

1. Recall (Re): $\frac{TP}{TP+FN}$
2. Precision (Pr): $\frac{TP}{TP+FP}$
3. Specificity (Sp): $\frac{TN}{TN+FP}$
4. False Positive Rate (FPR): $\frac{FP}{FP+TN}$
5. False Negative Rate (FNR): $\frac{FN}{FN+TP}$
6. Percentage of Wrong Classifications (PWC): $100 * \frac{FP+FN}{FP+FN+TP+TN}$
7. F-measure (FM): $2 * \frac{Precision*Recall}{Precision+Recall}$

The results obtained from these metrics are represented in Table 3. It should be noted out that the result of precision and recall is covered with the F-measure, and therefore, not shown on the table.

Category	Sp	FPR	FNR	PWC	FM
Bad Weather (BW)	0.99991	0.00009	0.022035	0.017216	0.977963
Dynamic Background (DBG)	0.99810	0.00190	0.048662	0.038013	0.951339
Intermittent Object Motion (IOM)	0.99790	0.00210	0.005232	0.004808	0.994768

Table 3 Metrics results

From the previous table we can observe that the model obtains excellent results in known datasets such as in Bad Weather and Dynamic Background categories. Moreover, the system obtains an impressive result on a category not used at the training and test phase as the Intermittent Object Motion category. From these outcomes, we can remark that our model is accurate in different and unknown scenarios, which is a relevant factor for implementing it on different drone's flights.

4.2 STATE-OF -THE ART COMPARISON

As mentioned in section 2.3, we have analyzed multiple state-of-the-art systems for detecting changes between a background and a foreground image. We have selected these state-of-the-art implementations in order to compare the performance of our development.

All the compared systems have used the CD2014 dataset to train and test their performance. Due to this condition, the evaluation is performed in equal conditions. Therefore, the results obtained are an objective measure of the performance of the implementation with the same conditions.

We have selected systems based on traditional image processing techniques, such as SuBSENSE [18] and other mathematical implementations as Gaussian Mixture Models [19]. Moreover, systems

based on CNN are included, such as ConvNet-IUTIS and ConvNet-GT [5]. The objective of including implementations based on different technologies is to study the improvement of performance by the inclusion of CNNs. In addition to that, CNN-based implementations permit to obtain significant conclusions of the performance of our model versus a system based on the same deep learning structure.

The traditional mathematical implementation has been selected relying on two reasons. On the one hand, these systems have made a significant contribution to the field, due to its performance related to other implementation based on similar technologies. On the other hand, the systems have metrics results from the CD2014 dataset. Therefore, they are ideal to compare our system with.

Table 4 illustrates the result of each selected state-of-the-art implementation, alongside with the results obtained by our proposed system. The F-Measure explained in section 4.1.4 is used to measure the performance, as it provides a balanced perspective, because of the use of precision and recall values, which take into account both possible detection errors: false negatives and false positives.

From the mentioned table, it can be observed that our model is the best solution for each category of the dataset employed. Our system overcomes problems of moving cameras and has a better performance in static acquisition devices. Therefore, we have obtained a very accurate detection system combined with an alignment component in order to give a complete solution for detecting changes between two sequences using moving cameras.

Implementation	$FM_{OVERALL}$	FM_{BW}	FM_{DBG}	FM_{IOM}
Proposed	**0.9747**	**0.9779**	**0.9513**	**0.9947**
ConvNet-GT **[22]**	0.9054	0.9264	0.8845	-
ConvNet-IUTIS **[22]**	0.8386	0.8849	0.7923	-
CNN **[3]**	0.7718	0.8301	0.8761	0.6098
GMM **[4]**	0.6306	0.7380	0.6330	0.5207
SuBSENSE **[5]**	0.7788	0.8619	0.8177	0.6569
PBAS **[6]**	0.6749	0.7673	0.6829	0.5745
PAWCS **[7]**	0.8285	0.8152	0.8938	0.7764

Table 4 State-of-the-art compared results

5 CONCLUSIONS AND FUTURE WORK

In this section, the main conclusions extracted from the development process are explained. Along with that, the achieved objectives are listed. Finally, we have described the future lines of work for our implementation.

5.1 CONCLUSIONS

In this document we have presented a change detection system for static and moving cameras using image alignment based on the ORB algorithm and Convolutional Neural Networks. Due to the use of drone imagery, which implies the movement of the camera, the problem of having dynamic backgrounds has been addressed. As far as we know, this issue has not been considered in any of the state-of-the-art methods for change detection mentioned along the paper.

However, the system is able to adapt to these conditions using image alignment techniques and the idea of a reference video or image. Datasets with dynamic backgrounds have been selected to train the network in order to achieve meaningful outcomes for real world applications. Results from experiments indicate a precise detection in scenarios with adverse conditions such as a snowfall or a blizzard. The comparison with other state-of-the-art methods reflects that our system is the most accurate on the studied scenarios. The precision of the reference's acquisition is crucial for the system's performance.

5.2 ACHIEVED GOALS

Image alignment system

As an improvement of the state-of-the-art implementations for change detection, we have introduced an additional component to our deep learning model for image alignment. This method permits to compensate the modifications introduced by factors such as human error during drone's pilotage as well as the influence of wind or the position of the camera. As mentioned along the document, this approach improves significantly the use cases of the system.

Dataset generation

The process to generate the data for training model has supposed a considerable challenge. Due to the sliding window algorithm, we have been able to obtain a method to automatically label the patches of each foreground, background and ground truth images.

Multi-input model creation and training

In previous developments, simpler models with one image as an input a numerical label have been implemented. This system has supposed a challenge in terms of introducing the data to the model correctly. A specific data generator has been developed. The creation of a generator able to combine

images to create the concatenated image for our input and retrieve the corresponding image with labeled changes for output is an important advance for future developments.

Image generation with Deep Learning

A reconstruction system has been implemented as an initial option for solving our problem. Although the results were not successful, the dataset generation for training this type of system along with the special architecture of the model supposed an interesting point to develop. In addition, this development was the first were we have implemented the sliding window algorithm along with the custom generator for training described before.

Change detection

A method to detect changes between two images have been built. This method is based on combination of a deep learning model together with image post-processing techniques.

Publication on book's chapter

Part of this development has been included in a chapter of book Internet of Things: from Data to Insight [40].

Research paper

The complete project is going to conform a research publication which is currently on its final phase before being sent to review.

5.3 FUTURE WORK

Geo-referenced image implementation

As stated in previous sections, the absences of geolocational information have supposed a considerable problem along the development. Similar implementations such as [32] have employed this data, although the dataset used on it were not as complete as CD2014 for creating a change detection model. The combination of GPS (Global Positioning System) information with the image alignment component will reduce the restrictions on the sequence acquisition. In addition, it will improve the selection of frames for detection, permitting a more time-spaced detection. Consequently, the computational resources for the system will be reduced.

Deep Learning image alignment

The use of traditional image processing methods such as ORB descriptor and geometrical transformation for image alignment have achieved remarkable results. However, we have seen that significant variations from the reference image to the image with changes introduced considerable perspective differences and noise.

As the use of deep learning has proved to retrieve excellent results in image processing, we are going to explore the use of this technology to substitute the traditional algorithms for our alignment system.

6 BIBLIOGRAPHY

[1] S. Ribeiro, «Using SimpleCV for seed metadata extraction into XML document,» *Iberoamerican Journal of Applied Computing*, vol. 4, p. 29, 2014.

[2] M. Calonder, V. Lepetit, C. Strecha y P. Fua, «Brief: Binary robust independent elementary features,» de *European conference on computer vision*, 2010, pp. 778-792.

[3] E. Rublee, V. Rabaud, K. Konolige y G. Bradski, «ORB: An efficient alternative to SIFT or SURF,» de *International Conference on Computer Vision*, 2011, pp. 2564-2571.

[4] D. G. Lowe, «Distinctive Image Features from Scale-Invariant Keypoints,» *Int. J. Comput. Vision*, vol. 60, nº 2, pp. 91-110, 2004.

[5] E. Rosten y T. Drummond, «Machine learning for high-speed corner detection,» de *Springer*, 2006.

[6] C. Harris y M. Stephens, «A Combined Corner and Edge Detector,» de *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147-151.

[7] P. L. Rosin, «Measuring Corner Properties,» *Comput. Vis. Image Underst.*, vol. 73, nº 2, pp. 291--307, 1999.

[8] G. Bradski, «The OpenCV Library,» *Dr. Dobb's Journal of Software Tools*, 2000.

[9] A. Sukhadeve, «Understanding Neural Network: A beginner's guide,» 6 August 2017. [En línea]. Available: https://www.datasciencecentral.com/profiles/blogs/understanding-neural-network-a-beginner-s-guide. [Último acceso: 1 5 2019].

[10] L. Bottou, «Online Algorithms and Stochastic Approximations,» de *Online Learning and Neural Networks*, 1998.

[11] D. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» *International Conference on Learning Representations*, 2014.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *J. Mach. Learn. Res.*, vol. 15, nº 1, pp. 1929-1958, 2014.

[13] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» de *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

[14] L. Prechelt, «Early stopping-but when?,» de *Neural Networks: Tricks of the trade*, Springer, 1998, pp. 55-69.

[15] Y. LeCun, P. Haffner, L. Bottou y Y. Bengio, «Object Recognition with Gradient-Based Learning,» de *Shape, Contour and Grouping in Computer Vision*, 1999, pp. 319--.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo y others, «TensorFlow: Large-scale machine learning on heterogeneous systems,» 2015. [En línea]. Available: https://www.tensorflow.org/.

[17] F. Chollet, «Keras,» 2015. [En línea]. Available: htttps://keras.io.

[18] C. Stauffer y W. E. L. Grimson, «Adaptive background mixture models for real-time tracking,» de *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, 1999, pp. 246-252.

[19] A. Elgammal, R. Duraiswami, D. Harwood y L. S. Davis, «Background and foreground modeling using nonparametric kernel density estimation for visual surveillance,» *Proceedings of the IEEE,* vol. 90, nº 7, pp. 1151-1163, 2002.

[20] P. St-Charles, G. Bilodeau y R. Bergevin, «SuBSENSE: A Universal Change Detection Method With Local Adaptive Sensitivity,» *IEEE Transactions on Image Processing,* vol. 24, nº 1, pp. 359-373, 2015.

[21] M. Hofmann, P. Tiefenbacher y G. Rigoll, «Background segmentation with feedback: The pixel-based adaptive segmenter,» de *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012.

[22] M. Braham y M. Van Droogenbroeck, «Deep background subtraction with scene-specific convolutional neural networks,» de *International conference on systems, signals and image processing (IWSSIP)*, 2016, pp. 1-4.

[23] D. Zeng y M. Zhu, «Background Subtraction Using Multiscale Fully Convolutional Network,» *IEEE Access,* vol. 6, pp. 16010-16021, 2018.

[24] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition,» *arXiv preprint arXiv:1409.1556,* 2014.

[25] Y. Wang, P. Jodoin, F. Porikli, J. Konrad, Y. Benezeth y P. Ishwar, «CDnet 2014: An Expanded Change Detection Benchmark Dataset,» de *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014.

[26] M. Babaee, D. T. Dinh y G. Rigoll, «A Deep Convolutional Neural Network for Video Sequence Background Subtraction,» *Pattern Recogn.,* pp. 635-649, 2018.

[27] R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. Douglas y H. Sebastian Seung, «Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,» *Nature,* vol. 405, pp. 947-951, 2000.

[28] J. Schmidhuber, «Deep learning in neural networks: An overview,» *Neural Networks,* vol. 61, pp. 85-117, 2015.

[29] R. Caruana, S. Lawrence y C. L. Giles, «Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping,» de *Advances in neural information processing systems*, 2001, pp. 402-408.

[30] J. Wang y L. Perez, «The effectiveness of data augmentation in image classification using deep learning,» *Convolutional Neural Networks Vis. Recognit,* 2017.

[31] D. Radošević y I. Magdalenić, «Python implementation of source code generator based on dynamic frames,» de *Proceedings of the 34th International Convention MIPRO*, 2011.

[32] «itertools - Functions creating iterators for efficient looping,» Python Software Foundation, 2001. [En línea]. Available: https://docs.python.org/3/library/itertools.html.

[33] S. van der Walt, S. C. Colbert y G. Varoquaux, «Computing in Science Engineering,» *The NumPy Array: A Structure for Efficient Numerical Computation,* vol. 13, nº 2, pp. 22-30, 2011.

[34] T. H. Group, «Hierarchical data format version 5,» 2000-2010. [En línea]. Available: http://www.hdfgroup.org/HDF5.

[35] E. C. M. Association, «ECMAScript Language Specification 3rd Edition,» 1999. [En línea]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.html.

[36] P. Soille, Morphological image analysis: principles and applications, Springer Science & Business Media, 2013.

[37] T. Minematsu, A. Shimada, H. Uchiyama, V. Charvillat y R.-i. Taniguchi, «Reconstruction-Based Change Detection with Image Completion for a Free-Moving Camera,» *Sensors,* vol. 18, nº 4, 2018.

[38] D. Avola, L. Cinque, G. L. Foresti, N. Martinel, D. Pannone y C. Piciarelli, «A UAV Video Dataset for Mosaicking and Change Detection From Low-Altitude Flights,» *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. PP, nº 99, pp. 1-11, 2018.

[39] P. St-Charles, G. Bilodeau y R. Bergevin, «A Universal Background Subtraction Using Word Consensus Models,» *IEEE Transactions on Image Processing,* vol. 25, nº 10, pp. 4768-4781, 2016.

[40] J. N. Davies y C. Fortuna, Internet of Things: from Data to Insight, Wiley-Blackwell, 2019.

[41] M. Braham y M. Van Droogenbroeck, «Deep background subtraction with scene-specific convolutional neural networks,» de *International conference on systems, signals and image processing (IWSSIP)*, 2016.

[42] K. Chokmani, B. Khalil, T. Ouarda y R. Bourdages, «Estimation of River Ice Thickness Using Artificial Neural Networks,» 2019.

7 APPENDIX A: ETHICAL, ECONOMICAL, SOCIAL AND ENVIRONMENTAL ASPECTS

7.1 INTRODUCTION

As this project is oriented to the surveillance sector, the ethical and social aspects achieve a considerable importance. As we are acquiring images where people could appear, we have to manage them properly in order to comply with the corresponding laws regulating data privacy, such as the General Data Protection Regulation (GDPR) in Europe. Moreover, the economic influence of this development for security corporations as mentioned in introductory sections should be considered.

In following sections, the main aspects and impacts of this project will be described.

7.2 DESCRIPTION OF ASPECTS

Ethical

As mentioned on the introductory part of the section, the ethical aspect of our project has significant relevance, as a result of the acquisition of images and the implications of this in terms of data privacy. Moreover, the fact that this project is oriented to the surveillance sector causes that the ethical factor applied to the mentioned sector affects our development. Each deployment of the proposed algorithm should take care of these matters carefully and individually, given that the ecosystems where the images have to be acquired can be dramatically different.

Economical

The automation of security tasks by our system will reduce the human resources required to perform them. However, the development of our system implies a maintenance service not needed previously, which should be considered in terms of costs by the organization acquiring the implementation.

Social

In depth relation with the ethical and economical aspects described previously, the influence of our development in social terms is related to the variation of human resources from the security task to the development of the system and its maintenance and its labor implications. In addition to that, the management of data for complying with data privacy regulation laws is a relevant aspect to be considered. In addition to that, if our system is employed to prevent risks due to accidents critical structure, the social benefits of it increases.

Environmental

The environmental impact of this project is related to the substitution of the vehicles used by the human security supervisor for the security task by electrical drones. This modification would reduce

the contamination produced by surveillance tasks. Moreover, the increase of precision to detect the changes implies a more liable prevention system if our system is employed on natural ecosystems.

7.3 MAIN IMPACTS

From the previous aspects influenced by this project, the impact of our development in terms of economy has a significant relevancy. Due to the automation of tasks, the reduction of human resources along with the benefit produced for the developers and maintainer companies would be an importance factor for the economic viability of our project.

7.4 CONCLUSIONS

From the preceding sections, the next set of conclusions can be extracted:

- The ethical considerations of our project are related to data privacy and should be managed to comply with the regulatory laws of each country where the system is applied.
- In terms of the economical aspect, the impact of our development is related to the benefits of the development organization. In addition to that, the reduction of human resources needed for the surveillance task is a relevant aspect to consider.
- For the social aspect, we have to consider a combination of data privacy and the economic influence in terms of labor aspects.
- Finally, the main environmental impacts of our project are the prevention of environmental risks along with contamination reduction because of the substitution of vehicles of human use such as motorbikes, cars, etc. by an electrical device such as a drone.

8 APPENDIX B: ECONOMIC BUGDET

Table 5 illustrates the economic budget of our project.

		hours	Price/hour	TOTAL
Labour costs (direct costs)		300	15 €	**4.500 €**
Material resources costs (direct costs)	Purchase price	Use in moths	Depreciation in years	TOTAL
PC and software	1.500,00 €	6	5	150,00 €
TOTAL				**150,00 €**
General expendures (indirect costs)	15%	on DC		**697,50 €**
Industrial benefit	6%	on DC and IC		**320,85 €**
SUBTOTAL				**5.668,35 €**
VAT			**21%**	**1.190,35 €**
TOTAL				**6.858,70 €**

Table 5 Economic Budget