Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Using Self-supervised Algorithms for Video Analysis and Scene Detection

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Juan Felipe Mora Arias

ADVISOR: Francesc Tarrés Ruiz

DATE: October, 23rd 2020

Title: Using Self-supervised Algorithms for Video Analysis and Scene Detection

Author: Juan Felipe Mora Arias

Advisor: Francesc Tarrés Ruiz

Date: October 23, 2020

Abstract

With the increasing available audiovisual content, well-ordered and effective management of video is desired, and therefore, automatic, and accurate solutions for video indexing and retrieval are needed.

Self-supervised learning algorithms with 3D convolutional neural networks are a promising solution for these tasks, thanks to its independence from human-annotations and its suitability to identify spatio-temporal features.

This work presents a self-supervised algorithm for the analysis of video shots, accomplished by a two-stage implementation: 1- An algorithm that generates pseudo-labels for 20-frame samples with different automatically generated shot transitions (Hardcuts/Cropcuts, Dissolves, Fades in/out, Wipes) and 2- A fully convolutional 3D trained network with an overall achieved accuracy greater than 97% in the testing set.

The model implemented is based in [5], improving the detection of large smooth transitions by implementing a larger temporal context. The transitions detected occur centered in the 10th and 11th frames of a 20-frame input window.

CONTENTS

FIGURES

TABLES

INTRODUCTION

In recent years, there has been an increasing amount of audiovisual content generated and streamed in the cloud, driven by technological advances such as smartphones, improved internet access bandwidths, and new multimedia content platforms (Netflix, Youtube, Tik Tok, Instagram, etc).

Well-ordered and effective management of video is desired to handle this massive available data, which depends on the availability of indexes. However, manual indexing and retrieval is not feasible for large video collections[1], and therefore, automatic solutions to process the video in a fast and accurate way are needed.

A common way to structure a video is by decomposing it into scenes which is an efficient solution for many browsing, indexing and summarization applications developed nowadays. A scene can be considered as a series of semantically correlated shots. The term scene usually refers to a group of shots taken in the same physical location describing objects or events.[2]

Proceeding towards the goal of video indexing, requires the grouping of shots into scenes, and therefore, needs a lower level segmentation of the video into shots.
Shot boundary detection is the first step for automatic cataloging. It detects the frame boundaries in the shot transitions and is the basis for advanced applications of video storage, management and retrieval. [3]

Once the video is splitted into shots, the keyframes[1] from each shot can be extracted and used to represent the respective shot. Different features of the keyframes and the shot can be used for video summarization, indexing and retrieval applications[4].

The purpose of this document is to present a self-supervised algorithm that generates pseudo-labels for different automatically generated shot transitions (Hardcuts/Cropcuts, Dissolves, Fades in/out, Wipes) and a fully convolutional 3D trained network topology based in [5].

Although the network topology proposed by Michael Gygli performs well detecting hard transitions, and temporarily short gradual transitions, its main drawback is when detecting smooth transitions with long durations, due to the limitation of its 10-frames window input size.

[1] The frame of a shot which conveys maximum information about the visual content in the entire shot.

To overcome the short context limitation, the topology proposed in this document, includes some modifications to the original model as listed below:

- Longer input window (20 frames).
- Detection of transitions centered in the 10th and 11th frames of the window.
- Adjustment in the temporal dimension of the convolutional kernel (from 3 to 5) to compensate for a larger input window.
- Gradual transitions (dissolves, fades in/out and wipes) with variable duration from 2 to 18 frames.

Although the input window is 20-frames, it is possible to detect special cases with longer gradual transitions, by downsampling the video. The subsampling can be performed, by a factor of 2 (or even greater), allowing to detect transition sizes of 40 frames or longer. This characteristic is especially useful for non-traditional shots as high-motion video, where a smooth transition is performed on several frames due to a higher frame rate.

After training the model with the stated improvements, the results threw an overall accuracy greater than 97% in the testing set.

The remainder of the document is organized as follows. The first chapter introduces the basic concepts of deep learning, self-supervised algorithms, Keras/Tensorflow and some aspects of the structure of a video to contextualize the project. Chapter two explains the proposed model implementation. Chapter three presents the self-supervised algorithm that produces the dataset with the pseudo labeled transitions and finally, chapter 4 and 5 are dedicated to results analysis and conclusions.

CHAPTER 1. BASIC CONCEPTS

1.1. Artificial Intelligence, Machine Learning and Deep Learning

Although these concepts can be confused with each other, it is important to understand the difference when talking about Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL). A simple way to understand them is as shown in (Fig. 1.1). DL is a subfield of ML which in turn is a subset of AI.

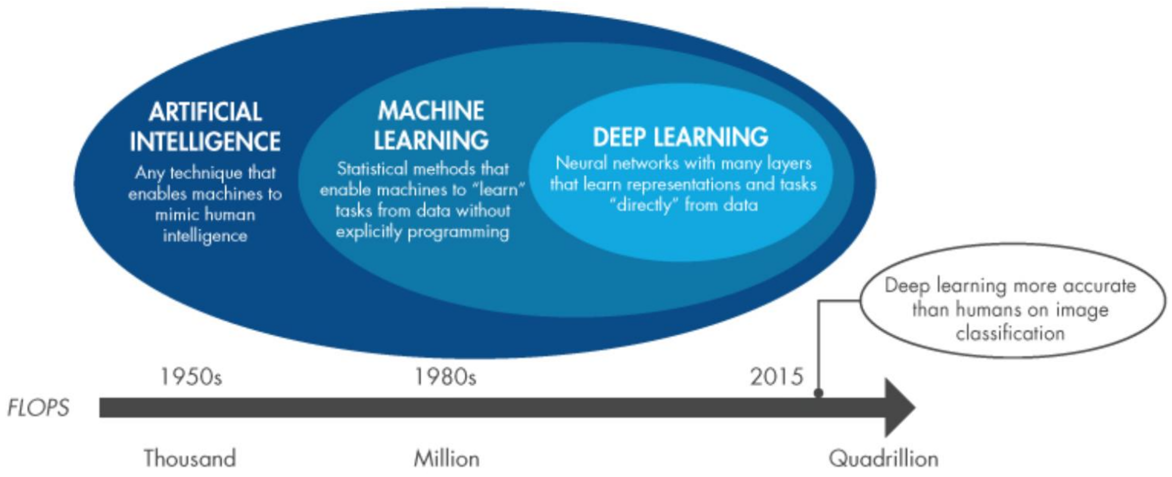

ARTIFICIAL INTELLIGENCE
Any technique that enables machines to mimic human intelligence
MACHINE LEARNING
Statistical methods that enable machines to "learn" tasks from data without explicitly programming
DEEP LEARNING
Neural networks with many layers that learn representations and tasks "directly" from data
Deep learning more accurate than humans on image classification
1950s
1980s
2015
FLOPS
Thousand
Million
Quadrillion

Figure 1.1 AI, ML and DL [6]

1.1.1. Artificial Intelligence (AI)

To define them into a little more detail and starting by the broader concept, Artificial Intelligence is a computer system, programmed to automate tasks that are normally associated with human thinking. This activities include decision-making, problem solving, learning, etc.[7]

In AI, the system receives some inputs (sensors information, data…), processes the information by performing operations with an algorithm, and throw certain output (take some action).

AI algorithms include machine learning and other approaches that does not involve any kind of learning. In fact, the first AI algorithms were all hardcoded and processed with low computational resources (a few thousand of FLOPS[2]).

This AI approach is known as symbolic AI or classical AI and pretends to explicitly represent human knowledge through a set of hardcoded explicit rules. Its main drawback is that, in order to mimic human intelligence, programmers need to translate implicit knowledge into explicit symbols and rules that the computer can understand.

[2] Floating Point Operations per Second.

1.1.2. Machine Learning (ML)

While in the classical paradigm, programmers set the rules, input the data and the computer throw out some output according to the rules, in ML the system is trained by introducing data with its associated outputs to a specific task, and the computer should come out with the rules for automating the task (Fig. 1.2).

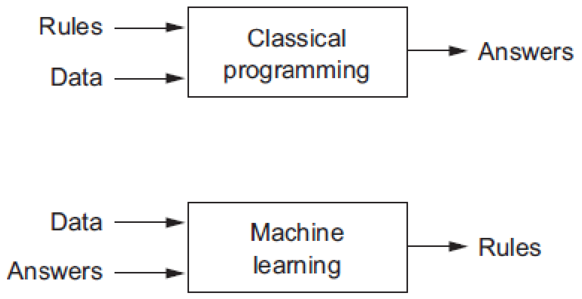


Figure 1.2 Classical AI vs ML[8]

The main aim of ML is to program the computers to learn from experience. Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases.[9]

The key problem of ML is to learn how to transform the data into representations that get closer to the expected output. To achieve this task, ML algorithms need four things:

- Input data.
- Labels/Examples of the expected output.
- A way to measure the distance between the current and expected output.
- A feedback signal used to adjust the algorithm and minimize the measure.

Traditional ML techniques transform the input data into few successive representation spaces via simple transformations and therefore, human intervention and feature engineering is a must.

There are several algorithms that may be used depending on the type of problem to solve. In general, the problems to solve with machine learning may fit in any of the three following classes:

- **Classification:** Classification models classify input data into categories. Predict discrete responses. For example, whether an email is genuine or spam, or whether a tumor is cancerous or benign.
- **Regression:** Predict continuous responses. For example, changes in temperature or fluctuations in power demand.
- **Clustering:** Is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns to group the data.

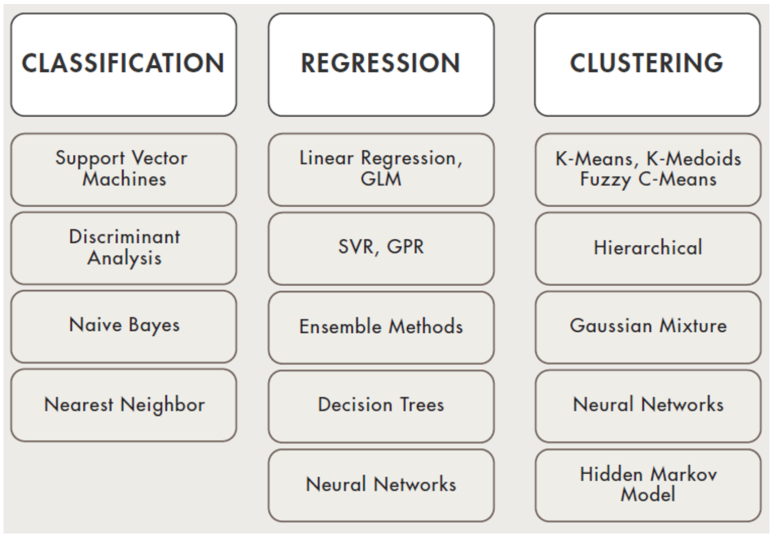


Figure 1.3 ML algorithms for each problem category[9]

In (Fig. 1.3), there is a summary guide of different algorithm alternatives to choose depending on the category the problem fits in. ML is a good alternative for solving complex tasks that involves large amount of data and lots of variables without a formula to solve the problem.

1.1.3. Deep Learning (DL)

Deep learning is a subfield of machine learning that puts an emphasis on learning successive layers of increasingly meaningful representations. The “deep” in deep learning stands for this idea of successive layers of representations. The number of layers in the model is known as the depth of the model.

With the arrival of DL, the feature engineering is automated and the model learns all layers of representation jointly, simplifying therefore the machine learning workflows, from a multistage approach to a simple end-to-end deep learning model.[8]

Most DL models implement Deep Neural Networks (DNN), which consists of an input layer, several hidden layers, and an output layer. Each layer is composed by different neurons and is interconnected using the output of the previous layer as its input.

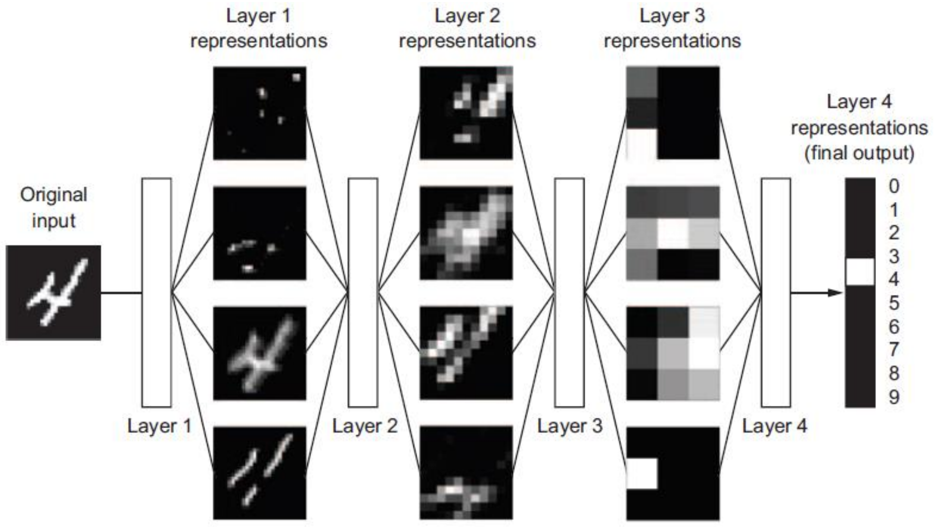


Figure 1.4 Deep representations learned by a digit-classification model[8]

DL allows to learn abstract complex representations by splitting them in several intermediate spaces, the layers. In (Fig. 1.4) there is an example of the representations learned by each layer in a digit-classification model. The first layers learn basic transformations of the input, while deeper layers show more complex representations that finally turn in the output prediction.

In the (Fig. 1.5) there is a general diagram of the deep learning flow. Initially, the weights of the network are randomly initialized and the model computes random transformations layer by layer, which will end up in predictions that are far from the true targets.

The loss function compares the predictions of the model with the true targets, and computes a loss score, which indicates how well has the model predicted the specific sample. Then, an optimizer uses this signal as feedback to adjust the weights in the network in a direction that decreases the loss score for the current sample (backpropagation).

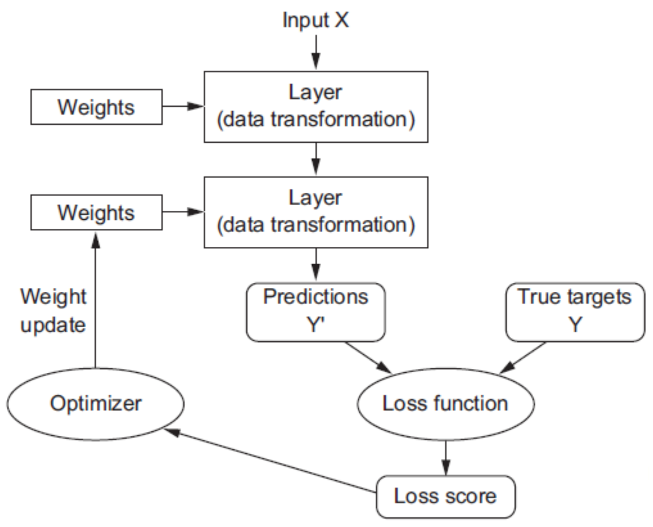


Figure 1.5 Deep learning diagram

By iteratively repeating the process mentioned above, the weights are adjusted, and the loss score decreases until the model finds the parameters that minimize the loss function.

Now, we see that what learning actually means, is tuning the correct set of parameters (weights) in the network, so that the network correctly matches the inputs into their associated output targets. The challenge is that the deeper the network is, the more parameters needed to be tuned (up to several million parameters), and therefore, more training samples and computational resources are needed to train the network.

1.2. Convolutional Neural Networks (CNN's)

A Convolutional Neural Network (CNN or ConvNet) is one of the most popular DL algorithms that is especially suitable for solving image and video tasks. What makes CNN special compared to traditional densely connected neural networks, is that dense layers learn global patterns involving all the pixels on the input, whereas convolution layers learn local patterns in small convolutional windows.

Furthermore, fully connected neural networks are not scalable, because the number of parameters increases exponentially with the depth (number of layers) and width (number of neurons on each level) of the network.

The key concept in CNN is the convolutional kernel, which may be thought as a filter specialized on finding a specific feature, for instance, edges on the image.

A convolution is applied by sliding the kernel window through every possible location in the image, extracting the spatial feature information in a filtered output image. (Fig. 1.6) shows an example of a 3 by 3 convolutional kernel (blue square) applied to an image.

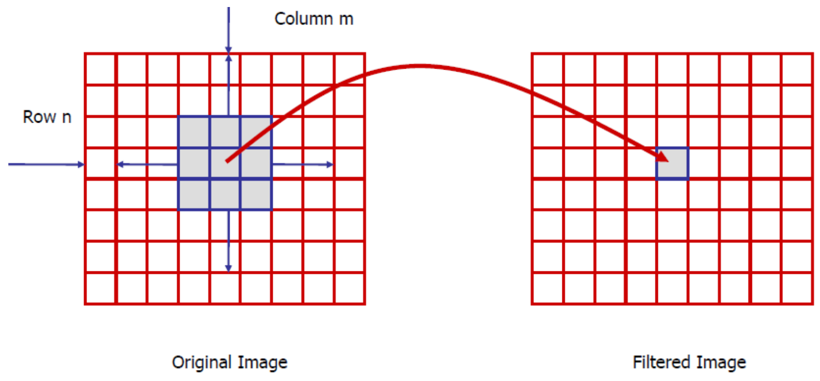


Figure 1.6 Kernel convolution over an image

Every spatial location in the output feature map corresponds to the same location in the input feature map. For instance, the upper-right corner of the output filtered image contains information about the upper-right corner of the original input image, and so on.

Usually, after applying the convolution to the input image, the output feature map will be spatially downsampled, due to border effects, where it is not possible to center-fit the convolutional kernel.

As showed in (Fig. 1.7), given an original image of size MxM and a convolutional kernel of size NxN, the resulting size of the feature map is (M-N+1) X (M-N+1).

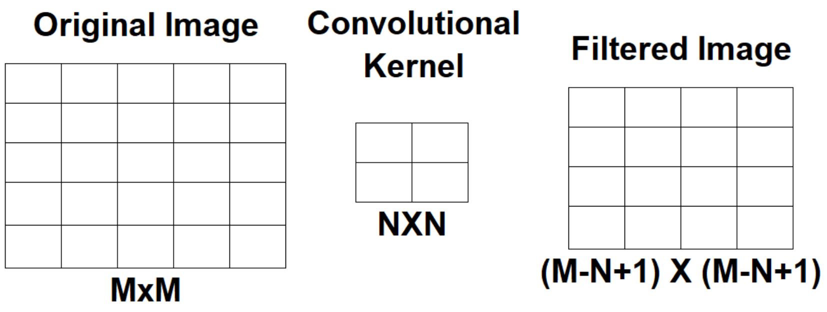


Figure 1.7 Image spatial shrinking after convolution

It is possible to keep the same input size in the filtered image, by applying padding, which consists of adding the necessary number of rows and columns on each side of the input feature map, so the convolution window may be centered at every position in the image. This strategy is useful in convnets, when it is desired to keep the same input size in the output feature map, which will be reduced in a controlled way using pooling layers.

As previously mentioned, convolutional layers look at local patterns by applying the same geometric transformation to the spatial locations in the input. This gives convnets two interesting properties[8].

- **Translation invariant representations:** A pattern learned in one corner of the image could be recognized anywhere else, making convolutional layers highly data efficient.
- **Learn spatial hierarchies of patterns:** Early convolution layers will learn small local patterns such as edges, while deeper layers will learn patterns made of the previous features, i.e. an eye composed by the previously identified edges. This allows that the network learns increasingly complex and abstract visual concepts.

Convnets are normally composed by two stages (Fig. 1.8). The first stage is the feature detection and is composed by stacks of convolution and max-pooling layers.

Each convolution layer contains several convolutional kernels (filters) that are applied to the input image. Each kernel produces a representation of the data (response map) with relevant information of certain feature that are stacked into the output feature map. Consequently, the depth of the output feature map indicates the number of kernels in the layer.

The feature map goes through the pooling layers, which down-sample the spatial dimensions, to keep a reasonable size, while maintaining the feature dimension space. This allows subsequent layers to see a greater spatial extent of the inputs.

It is common in convnet topologies to progressively increase the depth of the feature maps (number of features/filters), whereas the size of the map (width/height) decreases.

After the feature detection stage, it is a common practice to add a classification stage that includes a flatten operation, to turn the feature maps into vectors, followed by a fully connected network that ends up in the output classifier.

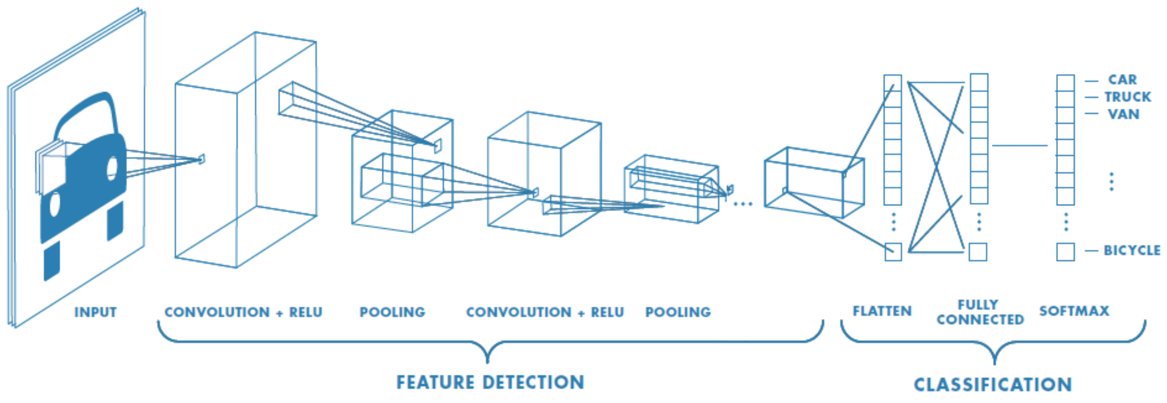


Figure 1.8 Typical convnet topology[10]

1.2.1. Conv3D

The previously explained convolutional layers were Conv2D (2D convolution layers), intended to perform spatial convolutions over images. Although Conv2D may be applied both, to image (Fig. 1.9a) and video inputs (Fig. 1.9b), the convolutional kernel operation would still be spatial convolution with a bidimensional feature map as output, losing important temporal information which is relevant for video classification tasks as shot boundary detection.

In contrast, Conv3D (3D convolution layers), extend the concept of convolution to volumes, where a 3D input, for instance a video, is processed by 3D convolutional filters that result in spatio-temporal feature maps as output (Fig. 1.9c). This allows to keep a temporal context information through the different layers, making conv3D more suitable for video processing.

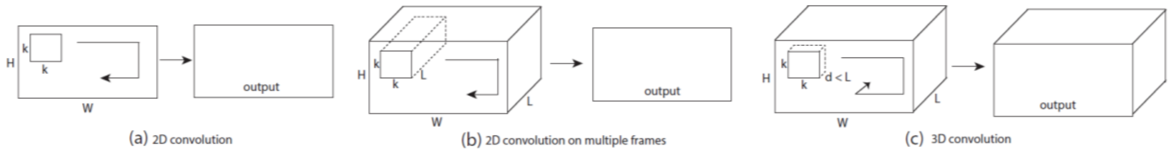


Figure 1.9 2D and 3D convolution operations[11]

1.3. Keras framework

As defined by its creator François Chollet, Keras is a Deep Learning framework for python that provides a convenient way to define and train almost any kind of DL model[8]. Keras follows best practices for reducing cognitive load, offering consistent and simple APIs that minimize the number of user actions required for common use cases, and providing clear error messages[12].

As shown in (Fig. 1.10), it provides high-level building blocks for developing deep-learning models, allowing it to run on top of different backend engines as TensorFlow[13], Theano[14] and Microsoft Cognitive Toolkit (CNTK)[15].

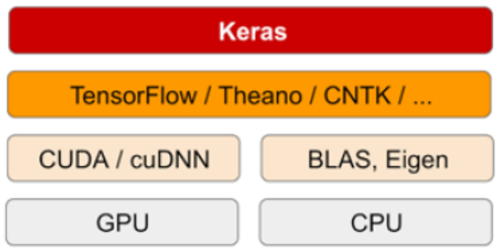


Figure 1.10 Keras software/hardware stack[8]

For the purpose of this project, Keras was chosen as the developing framework on top of TensorFlow, due to its simplicity, available documentation and developer guides online. It allows to define a network in few lines of code and supports training on Nvidia GPUs without extra coding, which saves a lot of time, especially when many parameters and samples need to be processed.

1.4. Learning schemas

1.4.1. Supervised learning

Consists of learning the weights of the network that allows mapping input data with known ground-truth targets, given a set of examples. The key point in supervised learning algorithms is the source of the reference labels, which are human-annotated. This learning schema is especially suitable for classification and regression tasks.

DL algorithms Trained with accurate human-annotated labels have obtained outstanding results on different hot topic DL applications as image classification, Optical Character Recognition (OCR), speech recognition, language translation, etc.

The main advantage of using supervised learning algorithms is that there is complete control over what the model is currently learning, since the labels are being manually provided, as opposed to unsupervised learning where the patterns are automatically learned. On the other hand, its main drawback is that the generation of large labeled datasets is expensive and time consuming, and therefore, finding enough useful data to train the algorithm may be a challenge.

1.4.2. Unsupervised learning

Unsupervised learning, as opposed to supervised learning, consists of discovering interesting transformations of the input data without predefined labels. As described in Section 1.1.2, it is mainly used for clustering tasks, by finding patterns to group the data.

It is commonly applied in the data analytics field, in tasks as data visualization, data compression, data denoising and better understanding of the data correlations at hand. Its main advantage is the less complexity compared to supervised learning algorithms because no previous knowledge or labels in the data are needed. Additionally, it can be implemented in real-time data processing since it does not require previously annotated labels.

The choice of a supervised or unsupervised algorithm mainly depends on the previous knowledge of the data and the available information of the data. Although unsupervised learning is less complex, it also provides less accurate results compared to supervised algorithms, because the machine needs to discover all the patterns on its own.

1.4.3. Self-supervised learning

Self-supervised learning is a form of supervised learning, without human-annotated labels, but still including labels, in order to supervise what the algorithm learns. The training datasets are autonomously (or automatically) pseudo-labelled[3] by finding and exploiting characteristics in the data, typically using heuristic algorithms.

Its main advantage lies on the independence of human annotations, which allows the usage of very large-scale datasets in the training of self-supervised learning models. Self-supervised methods have achieved promising results, making the performance gap in downstream tasks smaller compared with supervised methods.[16]

For the aim of this project, self-supervised learning schema was applied in a two staged scenario. First, the computer automatically generates the labeled samples, based on the instructions coded in the algorithm. In the second step, the model is trained with the samples and pseudo-labels of the generated database.

It should be noted that, with this approach, the algorithm that generates the data is a crucial step to correctly produce the desired labels and, in this case, correctly simulate the transitions of an edited film. The samples were created from a dataset of different TV series, producing two different sample categories: ‘Positive samples’ and ‘Negative samples’.

The positive samples are 20-frame snippets with transitions occurring centered in the 10th and 11th frames. Meanwhile, negative samples, are divided into samples without transition (20 frames from a single shot) and samples with non-centered transitions.

The samples are stored in an intuitive folder structure that helps to easily construct a samples/pseudo-labels database for training the network in the second stage:

Positive_Samples

- Cropcuts
- Dissolves
- Fades
- Hardcuts
- Wipes

Negative_Samples

- Dissolves
- Fades
- Hardcuts
- No transitions
- Wipes

Although the algorithm could be trained as a multiclass classifier that independently identifies the transitions, the approach for this project was to simplify the network making it easier to converge by training two output categories: ‘Transition’ or ‘No transition’.

[3] It is common to find the distinction between labels (human-annotated) and pseudo labels (automatically generated)

1.5. Video structure

Conceptually, a video is a stack of images (frames), that sequentially displayed in a screen at certain frame rate[4], are interpreted by the human eye as a continuous sequence, producing the illusion of image movement. (Fig. 1.11) illustrates a video with a frame rate equal to 1/T. Some basic properties of a video are listed below:

- **Frame rate:** Is the frequency at which consecutive images are displayed on the screen. Its units are the Frames Per Second (FPS). Most popular standards include Film (24 fps), PAL (25 fps), NTSC (30fps).
- **Resolution:** Is the number of pixels in each dimension of the image (width x height).
- **Length:** Total number of frames in the video.
- **Frame:** A single image in the video.

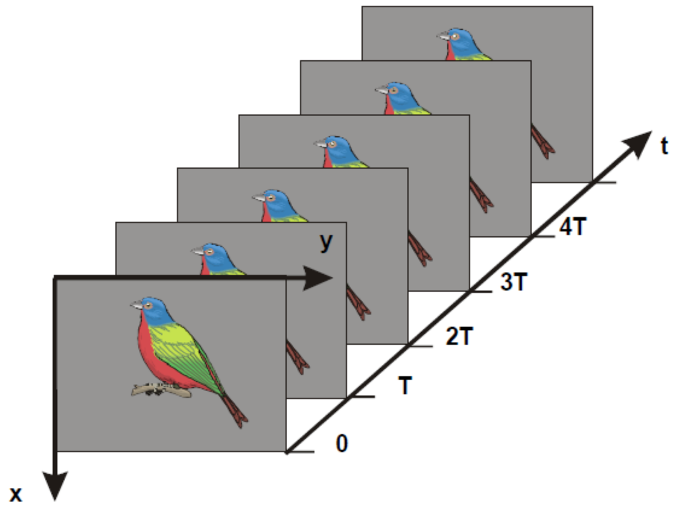


Figure 1.11 Temporal image sampling[17]

1.5.1. Scenes, shots and frames

A video may be understood as a 4 levels structure showed in (Fig. 1.12). The higher level is the video/movie/film, which is a visual form used to communicate ideas or stories through the usage of sequential images.

The video can be decomposed into scenes, which is a fundamental problem in semantic video processing, and its solutions have many applications in video summarization, indexing, and retrieval[18]. The concept of scene is more a semantic idea, that represents a spatio-temporal situation happening in the story. It may be composed by 1 or more shots that constitute a continuous narrative unit. A scene change happens when there is a temporal discontinuity or spatial (location) change.

[4] The minimum frame rate to achieve an illusion of fluid motion is between 16 and 18 frames per second.

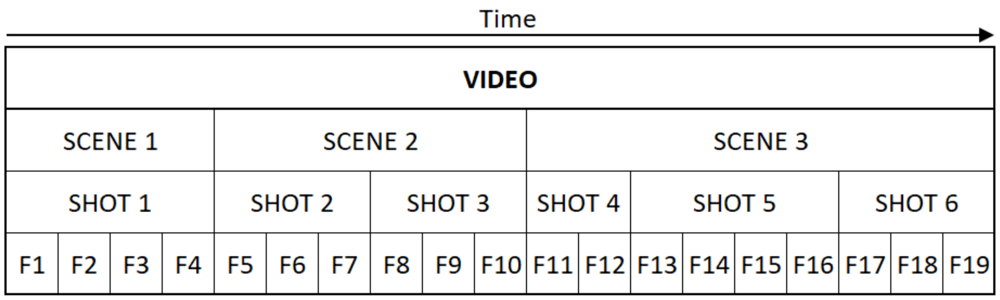

Time
VIDEO
SCENE 1
SCENE 2
SCENE 3
SHOT 1
SHOT 2
SHOT 3
SHOT 4
SHOT 5
SHOT 6
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16 F17 F18 F19

Figure 1.12 Video structure

The second level in the video structure is the shot, which is a small video segment capturing a continuous sequence of frames that were taken with a single camera without stopping or pausing the recording.

One key difference between shots and scenes is that inside one shot, the sequence should be continuous and without transitions, while a scene usually has transitions involved (even smooth cuts). The first and basic level of the video structure is the frame[5], which is a single image in the video.

1.5.2. Shot transitions

A film transition is a post-production process in which scenes or shots are combined to follow the storyline. There are two types of transitions according to its temporal duration, abrupt transitions (hardcuts and cropcuts), and gradual transitions (dissolves, fades in/out and wipes).

The most commonly used transitions are the hard cuts, but it is a common practice to carefully select similar joint shots in order to make the cut transitions smooth and sometimes hard to identify. A short explanation of each of the generated transitions is below.

- **Hard cuts:** Is the concatenation of two different shots.
- **Crop cuts:** A hard cut to a zoomed in version of the same scene.
- **Fade in:** The shot appears by a linear interpolation between a single-colored (black/white) sequence and the introduced shot sequence. This effect is commonly used to introduce a scene smoothly.
- **Fade out:** The shot disappears by a linear interpolation between the shot sequence and a single-colored (black/white) sequence. This effect is commonly used to end a scene smoothly.
- **Dissolves:** Linear interpolation between two different shots. The first shot fades out while the second shot fades in.
- **Wipe:** The frames of the first shot are horizontally replaced by the frames of the second shot. This transition occurs gradually by steps.

[5] A concept frequently used in the literature is “Key Frame”, which is the most representative frame of a shot.

CHAPTER 2. MODEL IMPLEMENTATION

2.1. Topology

The model implemented in this project is based on the network proposed by Michael Gygli in [5] to solve a shot boundary classification problem. The topology is a fully Convolutional neural network, which means, that all the layers involved are convolutional layers, removing the fully connected network classifier that is commonly implemented at the output stage of other topologies as mentioned in section 1.2.

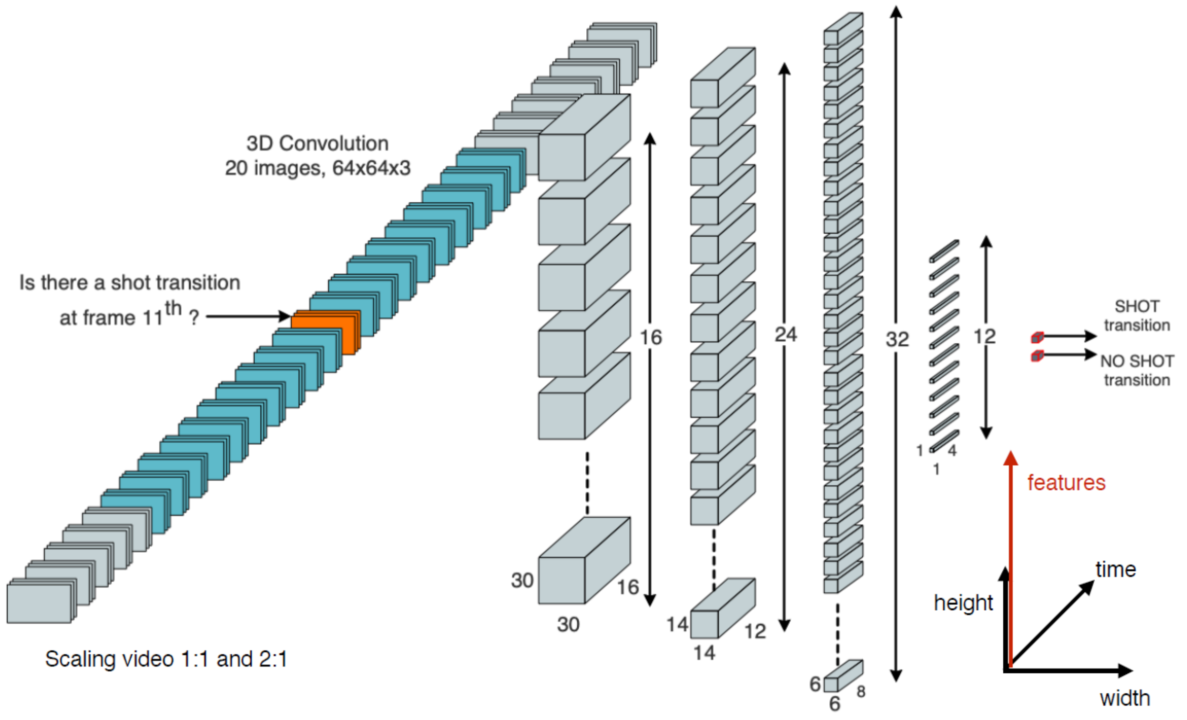


Figure 2.1 Network topology schema

Figure 2.1 shows a schema of the network topology implemented in this project. All the convolutional kernels are 3D filters (conv3D) that allow the network to learn spatio-temporal features, analyzing changing patterns of the input frames over time.

The input of the network is (20x64x64x3) which corresponds to a 20 frames video snippet with a resolution of 64 pixels, both in width and height dimensions, and 3 channels corresponding to its RGB color components.

This input goes through 5 convolutional layers from which the first 3 layers are Conv3D+Max-Pooling3D, while the last 2 layers only implement Conv3D. The spatial size reduces on each layer while the number of features is incremented. The last convolutional layer is followed by a flatten layer to end as a two-classes classifier.

The main changes implemented in the network, respect to the topology proposed in the paper are:

- Longer input window (20 frames).
- Detection of transitions centered in the 10th and 11th frames of the window.
- Adjustment in the temporal dimension of the first layer convolutional kernel (from 3 to 5) to compensate for a larger input window.
- Possible detection of larger gradual transitions, by reducing the input frame rate.

With these changes implemented, the network grows from 48,698 trainable parameters in the original network to 127,142 in our network. This increase in the number of parameters is the price to pay for involving a greater temporal context, but can be overcome by training a larger dataset, with the advantage of enhancing accuracy on the detection of smoot transitions. A more detailed description of the architecture is included in the table 2.1

Table 2.1 Network parameters

Layer (type)	**Kernel Size (t,w,h)**	**Output Shape**	**Param #**
input_1 (InputLayer)	-	[(None, 20, 64, 64, 3)]	0
conv3d_1 (Conv3D)	5x5x5	(None, 16, 60, 60, 16)	6016
max_pooling3d (MaxPooling3D)	Max pool (2,2,1)	(None, 16, 30, 30, 16)	0
conv3d_2 (Conv3D)	3x3x5	(None, 12, 28, 28, 24)	17304
max_pooling3d_1 (MaxPooling3D)	Max pool (2,2,1)	(None, 12, 14, 14, 24)	0
conv3d_3 (Conv3D)	3x3x5	(None, 8, 12, 12, 32)	34592
max_pooling3d_2 (MaxPooling3D)	Max pool (2,2,1)	(None, 8, 6, 6, 32)	0
conv3d_4 (Conv3D)	6x6x5	(None, 4, 1, 1, 12)	69132
conv3d_5 (Conv3D)	1x1x4	(None, 1, 1, 1, 2)	98
flatten (Flatten)		(None, 2)	0
Total params:	*127,142*		
Trainable params:	*127,142*		
Non-trainable params:	*0*		

Notice that the number of network parameters is relatively short compared to a densely connected network. For instance, if we take the first convolutional layer, we can calculate the number of parameters following (eq. 2.1).

$$\text{\# of layer parameters} = \text{\# of kernels} \times \text{kernel size} + \text{\# of bias} \tag{2.1}$$

There are 16 convolutional kernels in the layer, that generates the 16 output filters. Each kernel has associated one bias, so we have 16 bias values. The kernel dimensions (time, width, height, input channels) are 5x5x5x3, and as a result, we have 6,016 parameters in the first convolutional layer:

$$\textbf{\# of layer parameters} = 16 \times (5\times5\times5\times3) + 16 = \mathbf{6{,}016} \tag{2.2}$$

If we had an equivalent densely connected network layer, we would have:

$$\text{\# of layer parameters} = (\text{input}+1) \times \text{layer neurons} \tag{2.3}$$

The input is 20x64x64x3, the number of neurons in the layer would be the output shape 16x60x60x16 and as a result, the equivalent densely connected network layer would have around 226,500 million of parameters.

$$\textbf{\# of layer parameters} = ((20\times64\times64\times3)+1) \times 16\times60\times60\times16$$

$$\cong \mathbf{226{,}500\ M} \tag{2.4}$$

2.2. Activation functions

Fig. 2.2, shows a basic representation of a neuron with N inputs with its corresponding N weights to the neuron, and extra bias input. The represented neuron does not have any activation function.

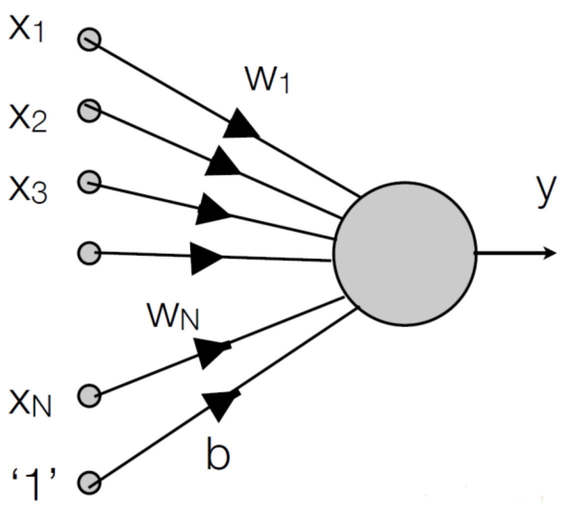


Figure 2.2 Neuron without activation function

The output of this neuron would be the weighted sum of its inputs and the bias as shown in (eq. 2.5).

$$y = \sum_{i=1}^{n+1} x_i \cdot w_i \; ; x_{n+1} = 1 \; ; w_{n+1} = \text{bias} \tag{2.5}$$

Without an activation function, the network would consist of only two linear operations (a dot product, and an addition), so the layer could only learn linear transformations (affine transformations) of the input data that would not take advantage of the depth in the network given its reduced hypothesis space.

To extend the possible transformations space in the intermediate layers, it is necessary to add a non-linearity or activation function. A non-extensive list of activation functions includes:

- Rectified Linear Unit (RELU)
- Sigmoid
- Softmax
- Softplus
- Hyperbolic Tangent Function (Tanh)…

2.2.1. Rectified Linear Unit (ReLU)

The rectified linear unit (ReLU) activation function performs a threshold operation to each input element where values less than zero are set to zero, and higher values are kept the same. The function that defines ReLU is (eq. 2.6) and its graphic representation is in (Fig. 2.3)

$$f(x) = \max(0,x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i \leq 0 \end{cases} \qquad \textbf{(2.6)}$$

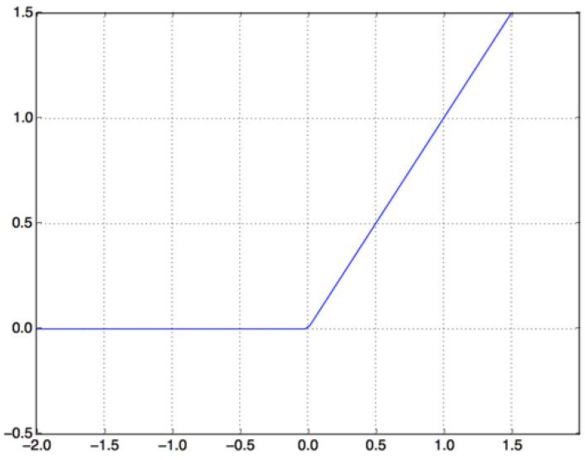


Figure 2.3 Rectified Linear Unit (RELU) activation function

The ReLU function is non-saturated, and its derivative function is a constant for positive values, which helps to avoid the vanishing gradient descent problem observed in other activation functions as sigmoid and Hyperbolic Tangent Function (Tanh)[19]. It also introduces sparsity in the hidden units, due to its zero forcing for negative values, but largely squishes the values between zero to maximum.

A big advantage of implementing ReLU as activation function is that it guarantees a faster computation compared to other functions since it does not compute exponentials and divisions. Additionally, as it represents a nearly linear function, it preserves properties of linear models that made them easy to optimize, with gradient-descent methods.[20]

ReLU has been the most widely used activation function for DL applications, due to its good results and simplicity. It has been implemented as activation function in several DL reference models (AlexNet[21], GoogleNet[22], ResNet[23]), making it a reliable choice when implementing a new network. It is usually applied to intermediate layers and combined in the output layers with other activation functions as Sigmoid and Softmax.

2.2.2. Sigmoid and Softmax

Sigmoid is one of the most widely used activation functions and is defined by the (eq. 2.7). Its graphical representation is in (Fig. 2.4)

$$f(x) = \frac{1}{1 + e^{-x}} \qquad \textbf{(2.7)}$$

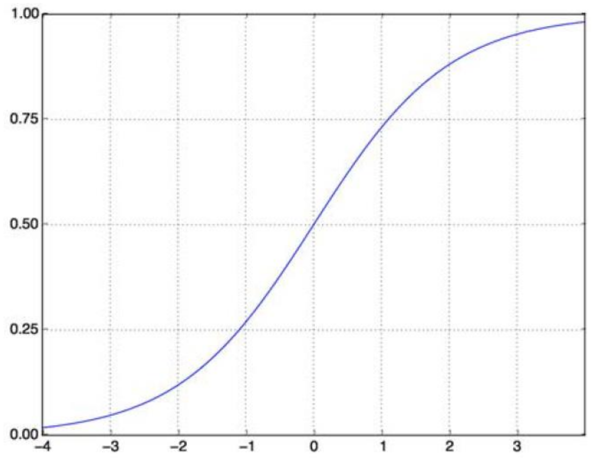


Figure 2.4 Sigmoid activation function

Its domain are the real numbers (-∞, ∞), and its range is (-0,1). It means that it behaves as a binary classifier with two saturation levels (0 and 1), which makes it suitable to implement as the output activation function for classification problems.

It is rarely adopted in intermediate layers due to its soft saturation. Specifically, in the process of optimizing the loss function, the derivatives of sigmoid function will reduce to zero in the saturation area, causing the vanishing gradient problem in more than five layers networks[19], in which the first layers receive less contributions of the learning.

The Softmax activation function is used to compute a probability vector distribution implemented in multi-class classifier models. The sum of the probabilities in the vector is equal to 1, and the target class predicted corresponds to the higher probability in the vector.

The main difference between the Sigmoid and Softmax AF is that the Sigmoid is used in binary classification while the Softmax is used for multivariate classification tasks. In our model, the last layer was defined as a two-neurons classifier and therefore, softmax was applied as the output activation function. The function is computed by using the following equation (eq. 2.8)

$$f(x_i) = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}} \qquad \textbf{(2.8)}$$

2.3. Loss function

The loss function, also called objective function, is in charge of comparing the predictions of the network with the targets provided and compute a distance score that indicates the quality of the network's output.

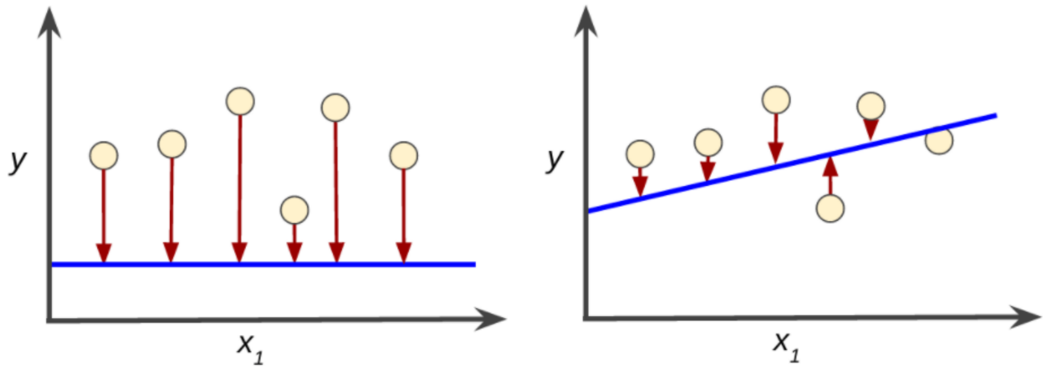


Figure 2.5 Representation of predictions with high (left) and low (right) loss[24]

In the (Fig. 2.5), there is a representation of a high and low loss. The circles represent the ground truth labels, the blue line represents the predictions of the model, and the red lines represent the loss of each prediction.

Although there are various loss metrics that can be applied to measure the performance of the network (and even personalized metrics can be defined), there are some guidelines, that may help to choose the correct loss function for common tasks such as classification, regression, and sequence prediction[8]. In the (Table 2.2) there is a summary of some of the common employed loss functions with its associated tasks.

Table 2.2 Common employed loss functions and its tasks

Loss Function	Task to solve
Binary crossentropy	Two-class classification
Categorical crossentropy	Many-class classification
Mean Squared Error (MSE)	Regression
Connectionist Temporal Classification (CTC)	Sequence learning

The shot detection problem of this project is a classification task, in which the network should classify the 20 frames input window as ‘Transition’ or ‘No transition’. This task can be solved even from the Two-class or Many-class classification approach, depending on how the output layer is defined. In our case, the proposed network topology includes a two neurons output, and therefore behaves as a categorical one hot encoded output (to be optimized with categorical crossentropy metric), in which **[1,0]** means **‘No Transition’** and **[0,1]** stands for **‘Transition’**.

Crossentropy is a quantity from the field of information theory that measures the distance between probability distributions. In this case, between the ground-truth distribution and the model’s predictions. It is usually the best choice when dealing with models that output probabilities, because due to the shape of the output activation function, other metrics as MSE may cause very small gradients when the initial state is far from the final solution.

2.4. Optimizer

If we could compute the loss function for all the possible weight values of a unidimensional problem, the resulting function would be convex as shown in (Fig 2.6). Conceptually, the lower point is the minimum and represents the point where the slope (derivative) of the loss function is 0 or in other words, the value of the weight by which the predicted value of the network matches (as close as possible) the desired output.

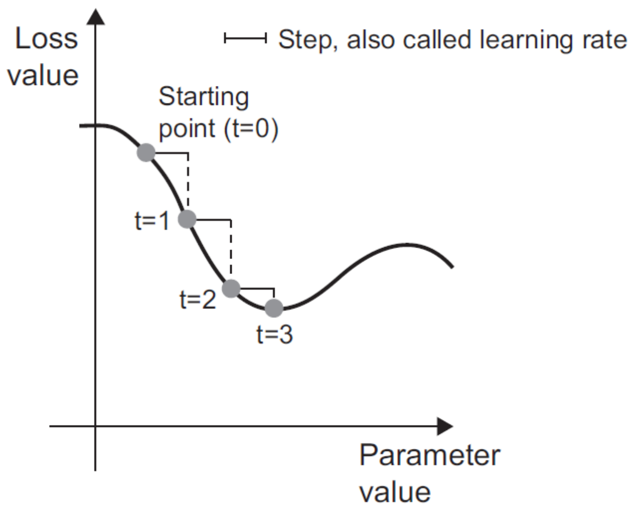


Figure 2.6 Loss vs weight function[8]

As it would be inefficient to calculate all the possible weight values and find the minimum, especially in a large network with thousands or even millions of parameters, an optimizer called gradient descent is implemented.

The idea is to pick a starting point (random weight value) and calculate the gradient of the loss curve at that point. This gradient is a vector that points in the direction of the steepest increase in the loss function, and its magnitude indicates how steep the slope is.

The weights of the network are then updated by taking a step in the opposite direction of the gradient multiplied by a constant (learning rate) in order to reduce the loss as quickly as possible as shown in (eq. 2.9).

$$w = w - (\text{learning rate} * \text{gradient}) \qquad \textbf{(2.9)}$$

This concept can be extended to an n-dimensional space with n-weights in the network, and the gradient would be a vector of partial derivatives with respect to the weights in the network.

It is important to pick a reasonable value for the step factor of the gradient updates. If it is too small, the algorithm will take many iterations to converge and it could get stuck in a local minimum. If, on the other hand, a too large value is chosen, the updates may end up in random locations on the curve, and the algorithm would never converge.

The optimizer chosen in DL specifies the exact way in which the gradient of the loss will be used to update parameters. There are some variations of basic SGD (RMSProp, Adam, Adagrad…) that pretend to accelerate the convergence of the algorithm and damp possible oscillations, but all of them are based on the above explained concept.

For the aim of this project, basic SGD optimizer was chosen to compile the model with its default configuration, which according to TensorFlow documentation, has a learning rate of 0.01 and disables momentum and nesterov, which are acceleration variants that update the weights of the network not only based on the current gradient but on the previous mini-batch.

CHAPTER 3. DATA GENERATION

Probably the main step for the successful implementation of a self-supervised learning algorithm concerns to the data generation. The algorithm should be able to produce samples that correctly simulate the transitions and associate the proper pseudo labels, so that the Convolutional Neural Network can learn and generalize[6] accurately the features it is intended to learn.

Creating a dataset with more than 1 million of 20-frame shot samples is not a trivial task, especially because it is time consuming and requires adequate processing and storage computational resources. It is important to organize and label the created samples in an easy-to-interpret way, thus facilitating the monitoring task and subsequently the fast assembling of the training dataset.

Although the 20-frame samples are automatically processed by the computer, it is necessary to perform a general supervision of the data outputted by the algorithm, to ensure its correct behavior.

There are two pseudo label categories in the data generated, **‘Positive samples**’ and **‘Negative samples’**.

3.1. Positive samples

The positive samples are any set of 20-frame samples that follow these three following characteristics:

- There is one of these transitions involved: Hardcut, Cropcut, Dissolve, Fade In/Out or Wipe.
- The transitions are occurring centered in the 10th and 11th frames of the sample.
- Smooth transitions (dissolves, fades or wipes) of different lengths[7].

The steps for the generation of the positive transitions are as follows:

1. Each episode is splitted in several 10-frames shot sequences.
2. Two shot samples are taken with a 400 frames separation (Fig 3.1).
3. The algorithm generates and store the 5 different transitions.
4. Next two shot samples with a 400 frames separation are taken…
5. Iteratively repeat steps 2 and 3 by taking the 10 consecutive frames.
6. Iteratively repeat on each episode of the database.

[6] Perform well on never-before-seen data.
[7] The length of the transition stands for the number of frames where the transition is occurring.

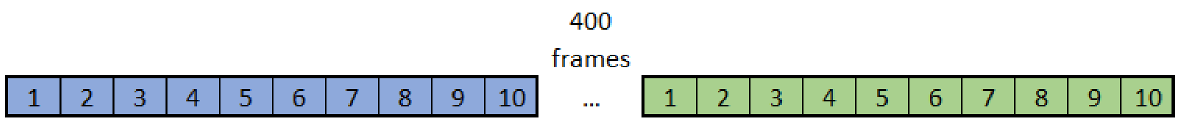


Figure 3.1 Source shots for transitions generation

Notice that considering a normal scenario with a 24 fps frame rate, the 400 frames separation of the shot samples correspond to a temporal difference of around 17 seconds in the video, which will match a different video context (even if the scene is the same), suitable to perform a transition.

3.1.1. Hardcuts

As mentioned in section 1.5.2, the hardcut is the concatenation of two different shots. The algorithm produces the two possible concatenations of the blue and green shots as showed in (Fig. 3.2)

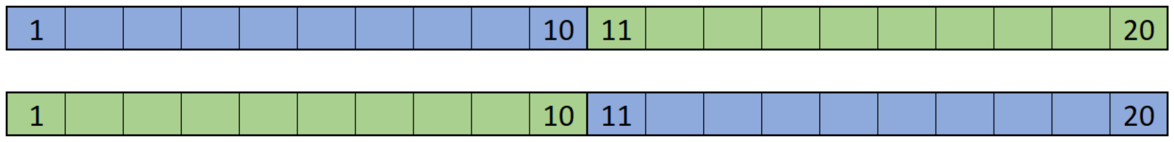


Figure 3.2 Hardcuts schema

The two 20-frame samples are stored with the following filename notation: ***hardcut_0_1_0001, hardcut_0_2_0001***

The number after the first underscore indicates the pair of frames from which the hardcut were taken.

> 0 means frame [0-9] concatenated with [400-409].
> 1 means frame [10-19] concatenated with [410-419] and so on...

The number after the second underscore indicates if the image corresponds to the first (top Fig 3.2) or second (bottom Fig 3.2) hardcut.

> 1 means [0-9] concatenated with [400-409]
> 2 means [400-409] concatenated with [0-9]

The last 4 digits correspond to the frame number in the output sequence. Between 0001 and 0020. Fig. 3.3 shows two different created hardcuts from the database.

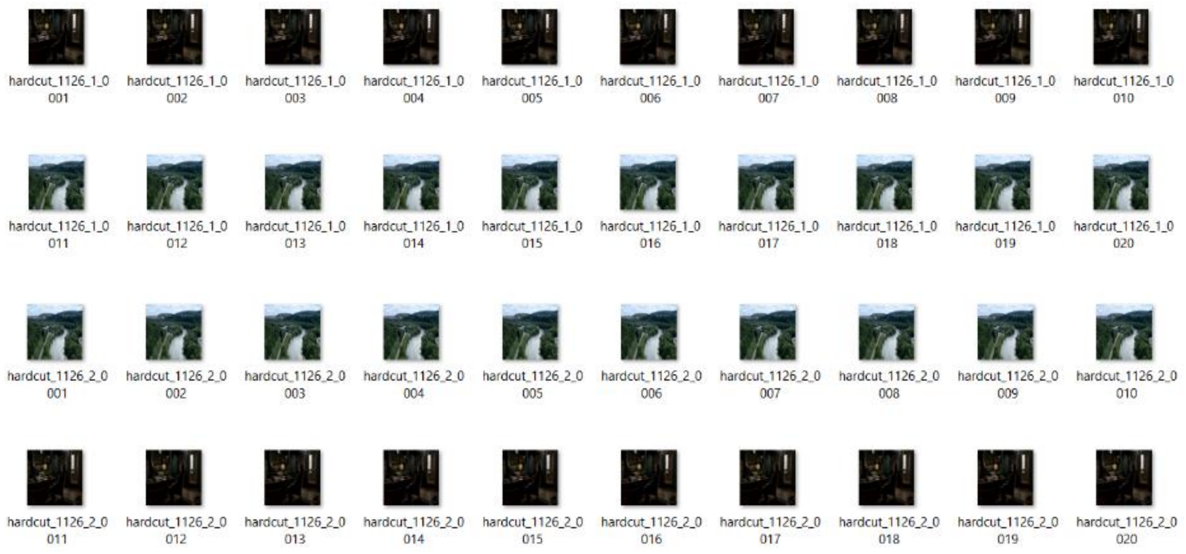


Figure 3.3 Hardcuts generated

3.1.2. Cropcuts

Figure 3.4 shows a schema of how cropcuts are generated. The steps for generating a cropcut are as follows:

1. Initially a 20-frame consecutive shot is taken (top of the figure).
2. A random window size and position in the frame are selected (Highlighted blue squares in frames 11 to 20).
3. Finally, a concatenation is performed with the 10 first original frames, and a zoomed in version in the window size of frames 11 to 20.

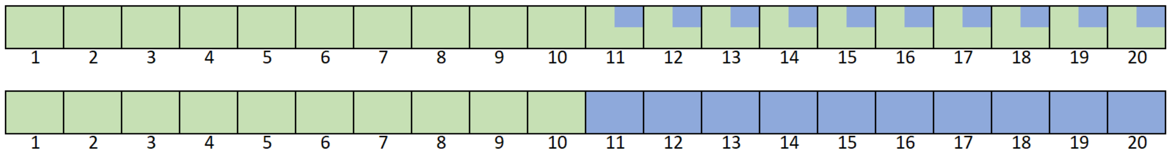


Figure 3.4 Cropcuts schema

The cropcut sample is stored with the following filename notation: ***cropcut_0_0001***

The number after the first underscore indicates a reference to the shot sample position in the original video.

0 means frame [0-19].
1 means frame [20-39] and so on...

The number after the second underscore correspond to the frame number in the output sequence indicated with 4 digits. Between 0001 and 0020. Fig. 3.5 shows a cropcut sample taken from the database.

Figure 3.5 Cropcut generated

3.1.3. Dissolves

A dissolve is a smooth transition in which one of the shots vanishes while the other one progressively appears. To achieve this, the algorithm performs the following steps:

1. Randomly chooses a transition size.
2. Separates the frames of both samples that are involved in the transition (transition frames) with the frames outside the transition window (non-transition frames).
3. Creates the transition by a progressive stepped interpolation between the transition frames of both shots.
4. Concatenates the non-transition frames with the progressive transition resulted from step 3.

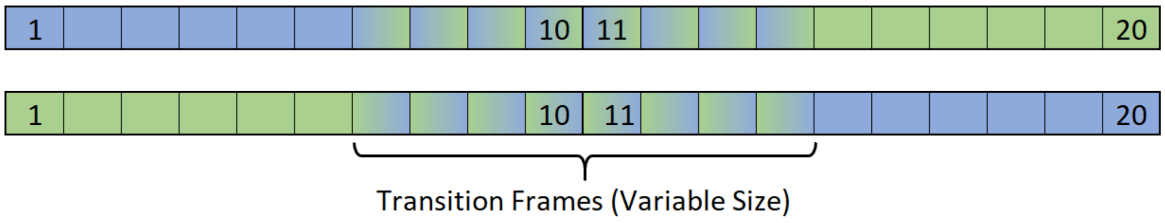


Figure 3.6 Dissolves schema

Two different dissolves are generated with the configurations shown in the (Fig. 3.6) and stored with the following filename notation:

dissolve_0_1_0001, dissolve_0_2_0001

The number after the first underscore indicates the pair of frames from which the dissolves were taken.

0 means frame [0-9] dissolved with [400-409].
1 means frame [10-19] dissolved with [410-419] and so on...

The number after the second underscore indicates if the image corresponds to a dissolve between green shot and blue shot or the opposite way.

1 means [0-9] dissolved with [400-409]
2 means [400-409] dissolved with [0-9]

The last 4 digits correspond to the frame number in the output sequence. Between 0001 and 0020. Fig. 3.7 shows an example of a dissolve taken from the database; were it is possible to identify a transition occurring from frame 7 to frame 14.

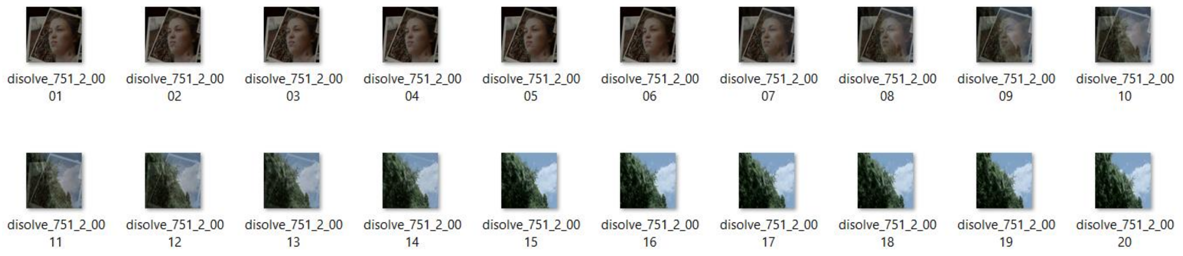


Figure 3.7 Dissolve generated

3.1.4. Fades In/Out

Fades are a specific case of dissolves, where the transition is performed between a 10-frames shot and a fixed black/white color. Fade in (Fig. 3.8 top) corresponds to the transition between the fixed color and the shot sequence, while fade out (Fig. 3.8 bottom) corresponds to the transition between the shot sequence and the fixed color.

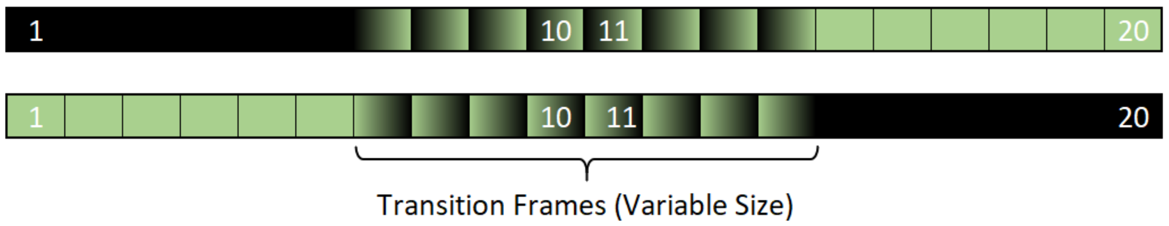


Figure 3.8 Fades In/Out schema

A fade in and a fade out are generated with each 10-frames shot sample and are stored with the following filename notation: ***fadingblack_0_1_0001, fadingblack_0_2_0001***

The number after the first underscore indicates a reference to the shot sample position in the original video.

0 means frame [0-19].
1 means frame [20-39] and so on...

The number after the second underscore indicates if the image corresponds to a fade in or a fade out.

1 means Fade In
2 means Fade Out

The last 4 digits correspond to the frame number in the output sequence. Between 0001 and 0020. Fig. 3.9 shows an example of a fade in and a fade out taken from the database.

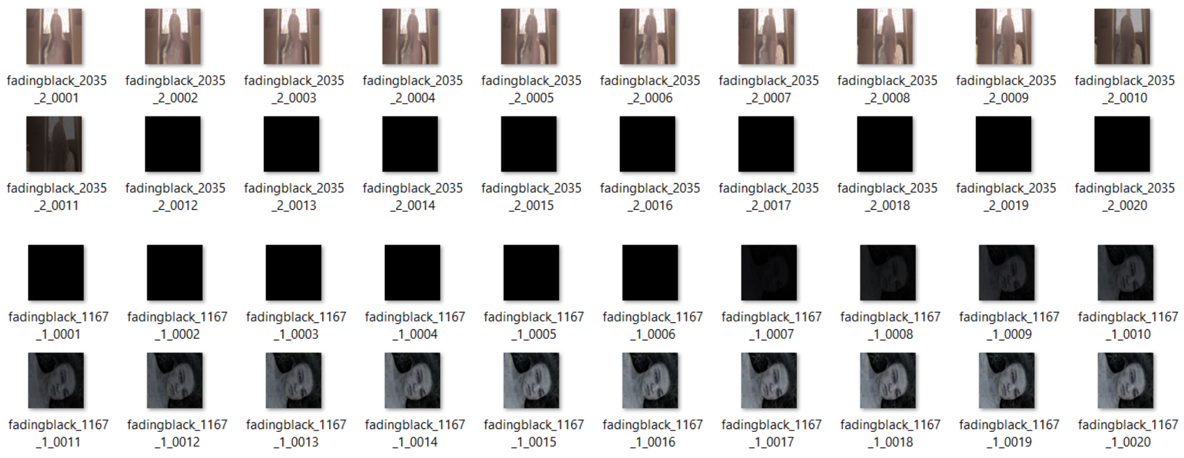


Figure 3.9 Fades In/Out generated

3.1.5. Wipes

A wipe is another type of smooth transition in which one of the shots is introduced horizontally in the scene by progressively replacing the pixels in the shot. The process is similar to the one described in dissolves, but instead of a linear interpolation of the pixels, there is a progressive increasing window in which the incoming shot pixels replace the previous shot pixels.

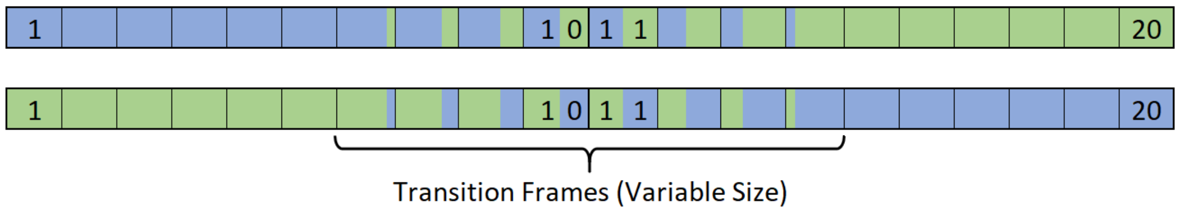


Figure 3.10 Wipes schema

Two different wipe configurations are generated as showed in schema (Fig. 3.10) and are stored with the following filename notation:

wiping_0_1_0001, wiping_0_2_0001

The number after the first underscore indicates the pair of frames from which the wipes were taken.

> 0 means frame [0-9] wiped with [400-409].
> 1 means frame [10-19] wiped with [410-419] and so on...

The number after the second underscore indicates which shot is being wiped into the other.

> 1 means [0-9] wiped with [400-409]
> 2 means [400-409] wiped with [0-9]

The last 4 digits correspond to the frame number in the output sequence. Between 0001 and 0020. Fig. 3.11 shows an example of a wipe sample taken from the database; were it is possible to identify a wipe transition occurring from frame 2 to frame 19.

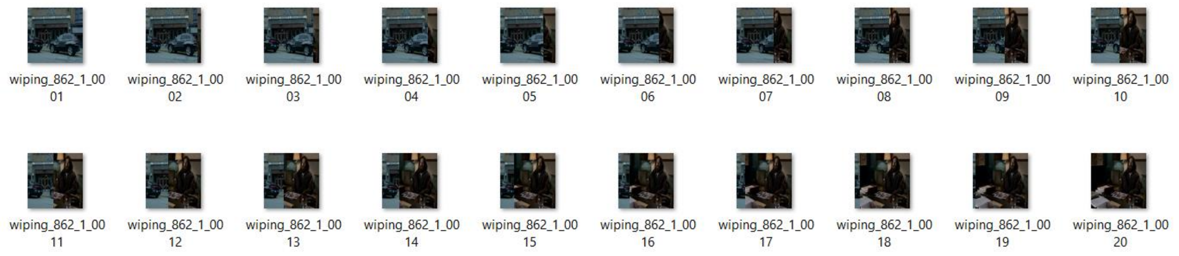


Figure 3.11 Wipe generated

3.2. Negative samples

The negative samples are the samples in which the desired classification result of the model is negative, which means there is not transition occurring centered in the 10th and 11th frames. It is important to identify two sample categories that fit in this label, **'No transitions'** and the **'Shifted transitions**.

3.2.1. No transitions

In these samples there is no transition occurring during the 20 frames window. This means, the 20-frame sample is a continuous sequence corresponding to only one shot. This samples are taken as the result of splitting each episode in several 10-frames shot sequences (as we did for generating the transitions) and then duplicating the resulting samples, to achieve a 20-frames size sample.

Additionally, as videos often contain flashes, which result in heavy frame-to-frame changes that can be falsely interpreted as a shot boundary[5]; some random artificial flashes were introduced to make the network less susceptible to these kind of changes. The process to create the flashes is as follows:

1. Randomly choose if a flash is introduced or not to the transition.
2. Randomly choose an offset (frame where the flash will be introduced) and a duration (# of consecutive flashed frames).
3. Convert the frames to be flashed in the LUV color space and increase the intensity by 50%.

The (figure 3.12) shows an example of a 20-frames 'No transition' sample in which frames 15, 16 and 17 were exposed to the flash process.

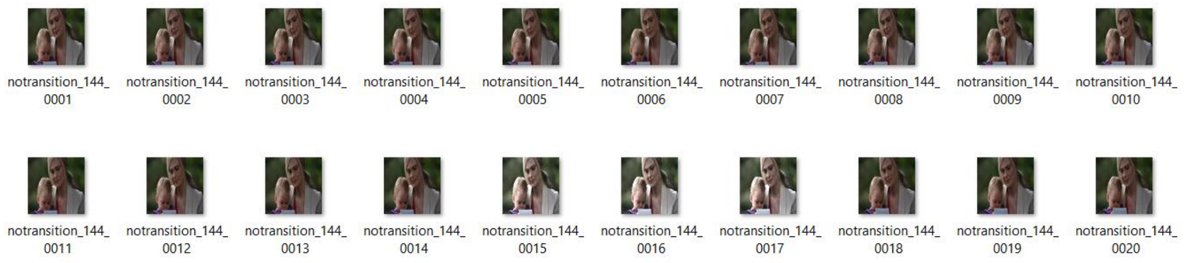


Figure 3.12 Sample labelled as 'No transition' with introduced flash noise

3.2.2. Shifted transitions

These samples are 20-frame snippets that contain a shot boundary inside. This means, there is a transition between two different shots occurring within its 20-frame window. The difference with a positive transition resides in the fact that this transition is shifted (not centered in the frames 10th and 11th), and therefore, should be interpreted as a 'Negative sample'.

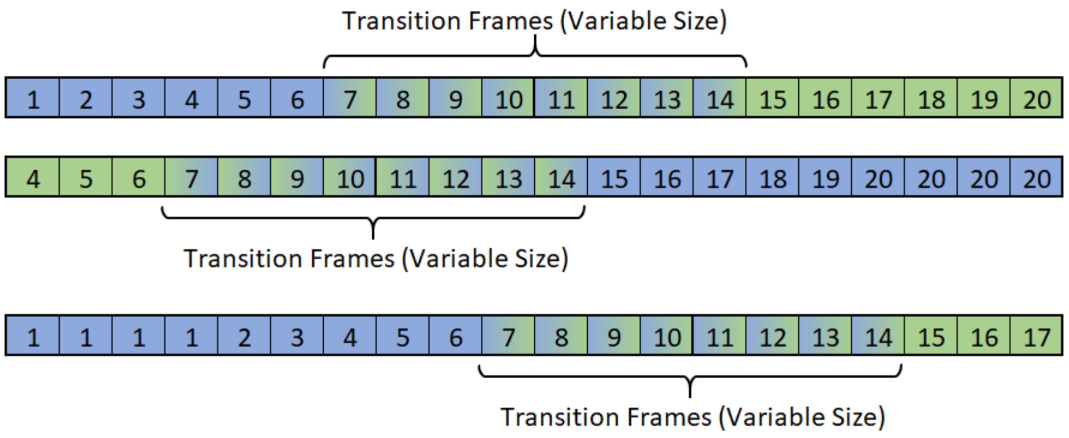


Figure 3.13 Creation schema of shifted transitions

Shifted transitions were obtained as showed in (Fig 3.13) by shifting the central frames of the positive transitions (top figure) to the left (center figure) or right (bottom figure) and padding the missing frames with a duplicate of the corner frame. Some snippets of shifted transitions are shown in figures 3.14 to 3.17 below.

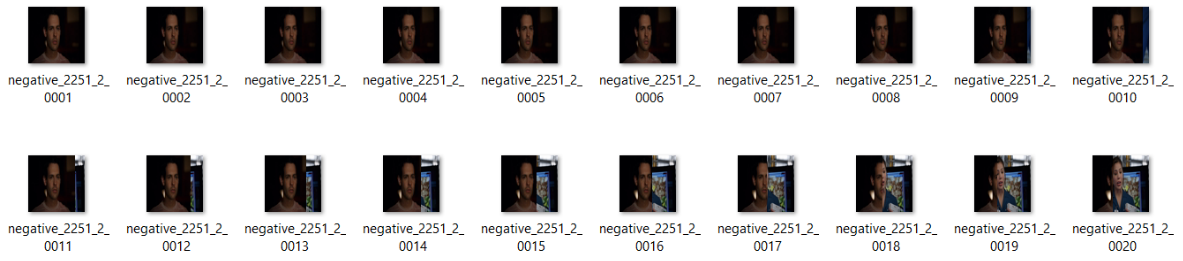


Figure 3.14 Shifted wipe sample

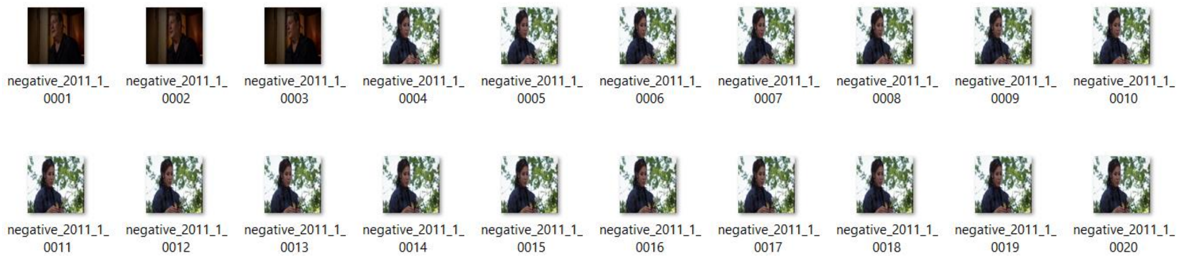


Figure 3.15 Shifted hardcut sample

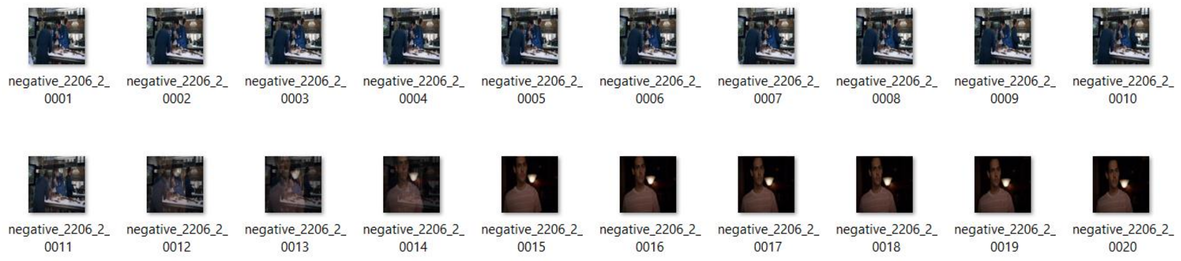


Figure 3.16 Shifted dissolve sample

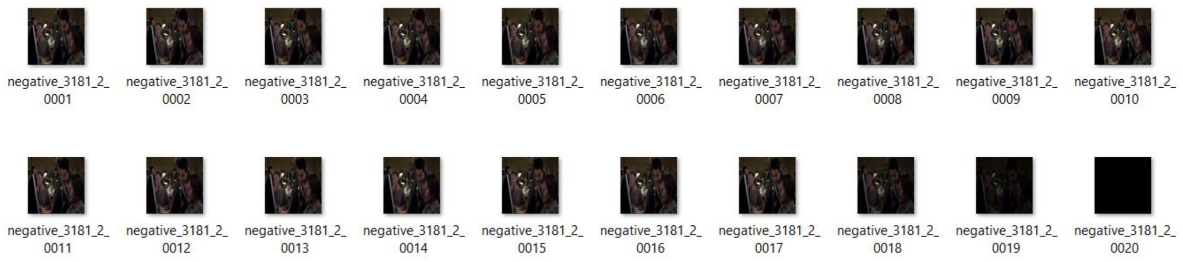


Figure 3.17 Shifted fade out sample

3.3. Preparing the data

The computational resources and time are big constraints to be considered when preparing the data and planning the training of the model. Depending on the number of samples generated, the computer may spend several days creating the transitions and training the model.

In general, for DL applications, good computational resources are required. In the specific case of this project, the model has been trained and tested with 1,331,792 samples and 20-frames window per sample.

Tables 3.1 and 3.2 show a summary with the number of positive and negative samples with which the network was trained. The classes were balanced to train the network with a similar number of representations of each transition and its negative counterparts. The total dataset was divided in training (~90%) and testing (~10%) samples.

Table 3.1 Positive samples

	Cuts	Dissolves	Fades	Wipes	Subtotal
Training	179,088	137,046	137,046	137,046	590,226
Testing	22,960	17,570	17,570	17,570	75,670
Total positive samples					**665,896**

Table 3.2 Negative samples

	No Transitions	Shifted Cuts	Shifted Dissolves	Shifted Fades	Shifted Wipes	Subtotal
Training	89.544	89.544	137.046	137.046	137.046	590.226
Testing	11.480	11.480	17.570	17.570	17.570	75.670
Total negative samples						**665.896**

It is generally a good practice to normalize the data before training the model, to make data more amenable for the network. This is easily achieved by subtracting the mean and dividing by the standard deviation, bringing data ranging from -1 and 1 and centered in 0.

$$\text{normalized sample} = \frac{\text{sample-127.5}}{127.5} = \frac{\text{sample}}{127.5}-1 \qquad \textbf{(3.1)}$$

The drawback of normalizing the data is that when represented in a [0, 255] range, it can be stored as a uint8, where each pixel/value occupy 8 bits of information, whereas normalized data needs to be represented (at least) by a float32 format, requiring therefore 4 times more space.

$$\textbf{\# Samples = 1,331,792 samples} \tag{3.2}$$

$$\textbf{Sample size} = 20\times64\times64\times3 \; \textbf{= 245,760 pixels} \tag{3.3}$$

$$\textbf{dtype} = 32 \text{ bits} = \textbf{4 Bytes} \tag{3.4}$$

$$\begin{aligned}\textbf{Database size} &= \#\text{ samples} \times \text{sample size} \times \text{dtype} \\ &= 1{,}331{,}792 \times 245{,}760 \text{ x } 4 \approx \textbf{1.3 TB}\end{aligned} \tag{3.5}$$

This means that 1.3 TB[8] needs to be processed by the model. Thus, data requires to be splitted in smaller size mini-batches that can be allocated by the memory of the system and processed by the GPU. The batch size is frequently selected as a power of 2 (typically between 8 and 128) to facilitate memory allocation on GPU.

For the aim of this project, a training and testing generator were programmed to provide 128 sample mini-batches to train the network. Each of the mini-batches is normalized, class balanced and shuffled before feeding the convolutional neural network.

[8] Tera Byte

CHAPTER 4. RESULTS

4.1. Measures of success

It is important to have a way to measure the performance of the network on the training and testing samples, but ¿What is a good performance? ¿What is a successful result?

In classification problems as it is our case, the confusion matrix is a useful statistical tool that may give us an insight of what the model is doing good and what is not. Each row of the matrix represents the predicted classes while each column represents the ground truth labels. The results can be divided in 4 classes:

• True Positives (TP):	Case was positive and predicted positive.
• True Negatives (TN):	Case was negative and predicted negative.
• False Positives (FP):	Case was negative but predicted positive.
• False Negatives (FN):	Case was positive but predicted negative.

Table 4.1 Confusion matrix

		Labels	
		Positive samples	Negative samples
Predictions	Transition	TP	FP
	No-Transition	FN	TN

Different measures of success can be analytically obtained from this matrix. The most widely used for classification tasks are the accuracy, precision, and recall.

4.1.1. Accuracy

For balanced-classification problems, where every class is equally likely, Accuracy is a commonly used metric. It measures the fraction of predictions in which our model got right and is defined by equations **(4.1)** and **(*4.2)***

$$\text{Accuracy} = \frac{\#\text{ of correct predictions}}{\text{Total }\#\text{ of predictions}} \qquad \textbf{(4.1)}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \qquad \textbf{(4.2)}$$

The data batches created for training and testing the model are formed by a balanced number of positive and negative samples, with the purpose of having a better generalization and convergence, and therefore, this metric is very suitable to evaluate the results of the model.

4.1.2. Precision and Recall

For class-imbalanced problems, precision and recall are better metrics. Precision is a measure of the proportion of positive identifications that were ground truth positive labels, while Recall is a measure of the proportion of ground truth positive labels identified correctly.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{4.3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{4.4}$$

4.2. Preliminary model training

Although the network is based on a previously tested topology that achieved good results, and therefore, convergence is almost guaranteed, generally it is a good idea to train the model on a small bunch of data and check if the model learns the features and converges after some epochs[9].

A preliminary training of the model was performed on a small set of 6144 snippets, corresponding to 48 mini batches of 128 samples each, which were trained on 20 epochs. (Fig. 4.1) shows the accuracy and loss during the training of the model.

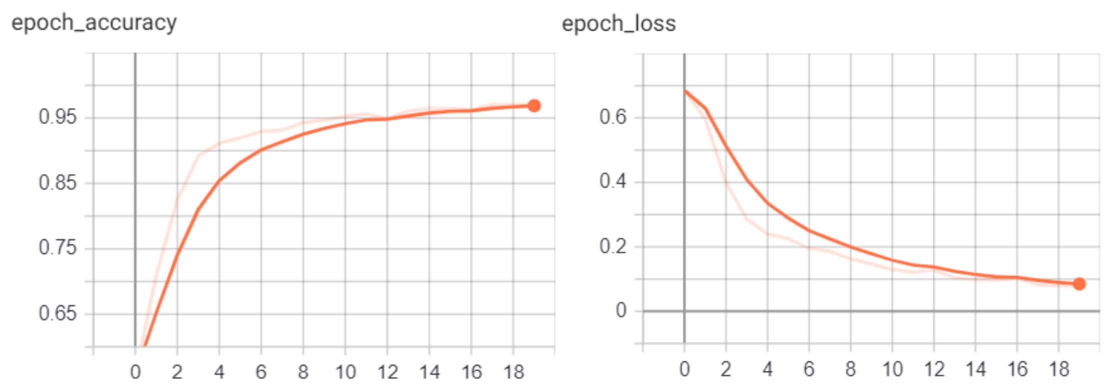


Figure 4.1 Preliminary training of the model

It is clearly seen that the accuracy improves on every epoch while the loss progressively decreases. This behavior indicates that the model is learning the features and the network is converging. In this case, the achieved training accuracy after the 20 epochs is around 96%.

After confirming that the model learns the features and converges, the next step is to train the network on the large dataset generated. A large dataset avoids overfitting[10] in the network, and allows a better generalization of the learned features.

[9] An epoch is a single iteration over all the training data.
[10] Overoptimize on the training data, ending up learning specific representations of the training data.

4.3. Overall training and testing of the model

The training set is composed by 1,180,452 samples with half of the snippets labeled as ‘Positive samples’ (Transition) and the other half as negative samples (No transition).

It is a waste of time and resources having constantly to access the disk looking for a small bunch of images, build a numpy array to train the network, and iterate the same operation thousands of times over millions of images. If the training process is already slow, reading the data is a bottleneck.

However, it is not feasible to read all the data at once and train the network, since all the samples can’t be allocated in memory. An efficient solution is to store numpy compressed files (.npz file extension) in a size that fits in memory once decompressed. These files are comparatively fast to read and generally occupy less than half of its original size in disk.

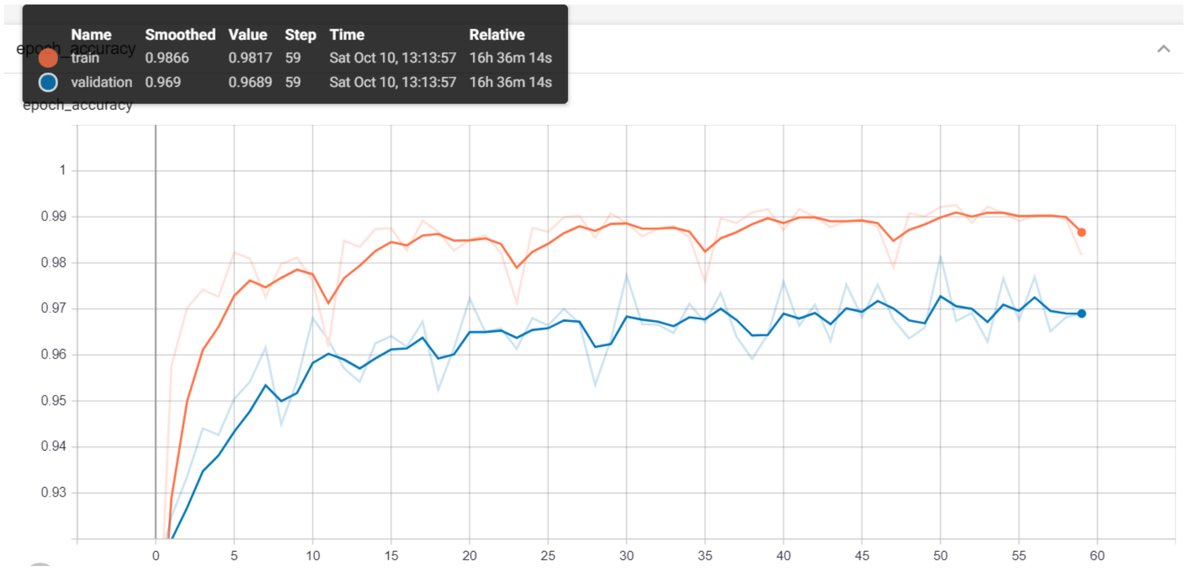
The training dataset was stored in 157 numpy compressed files occupying 230 GB space in disk, which during training were read by a generator that fed the model in mini-batches of 128-samples.

Because the training process is time consuming and it may take several hours or even days until the algorithm ends the iterations, it is desirable to have an updated feedback of the performance metrics and to periodically save the state of the model.

Two helpful keras callbacks were used on training time to monitor the progress and store the model information:

- TensorBoard: This callback logs events of the training metrics that can be displayed at the same execution time through the Tensorboard[25] monitoring tool.
- ModelCheckpoint: Allows to save the complete Keras model or the weights at a specified frequency.

The model was executed to train the network by 5 epochs, with the callbacks storing a total of 60 checkpoints. Each checkpoint contains the weights of the model and store the training and validation performance metrics. The execution time for the 5 epochs was 16 hours 40 minutes, running the model on a Nvidia GeForce GTX 1050 GPU.

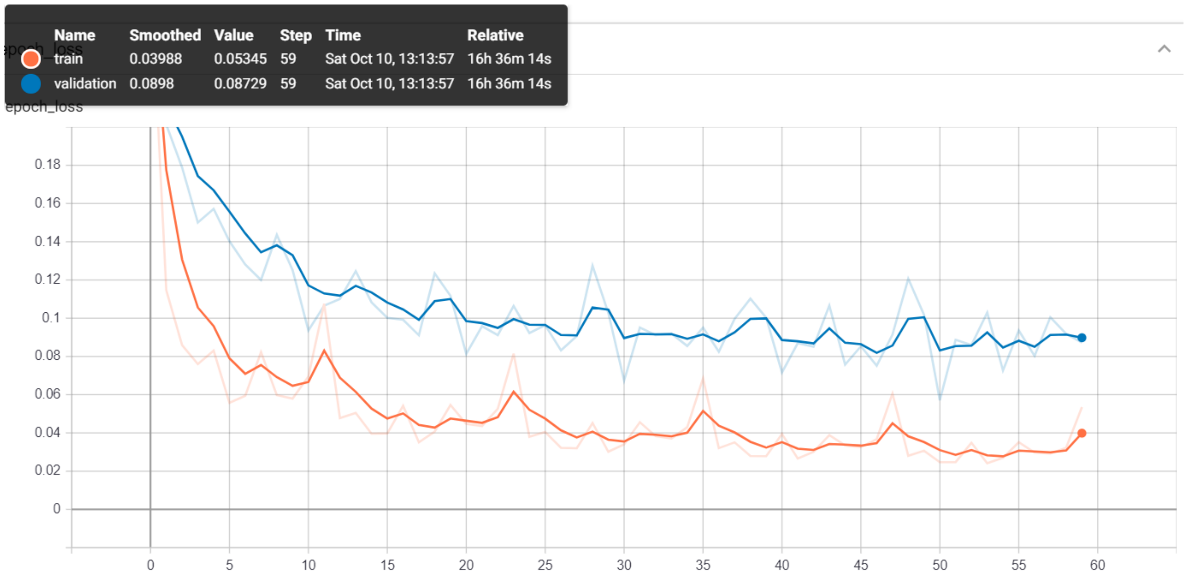


Name	Smoothed	Value	Step	Time	Relative
train	0.9866	0.9817	59	Sat Oct 10, 13:13:57	16h 36m 14s
validation	0.969	0.9689	59	Sat Oct 10, 13:13:57	16h 36m 14s

Figure 4.2 Training and testing (epoch vs accuracy) curves

Fig 4.2 and Fig 4.3 display the accuracy and loss information stored on the 60 checkpoints (5 epochs), during the training of the model. The Orange line corresponds to the training accuracy/loss, whereas the blue line represents the testing accuracy/loss.

It is clearly seen that on the first checkpoint, where the dataset has seen around 100.000 samples, the overall accuracy is below 92%, but keeps increasing after every iteration, until on the 5th epoch (60th checkpoint), the network accuracy converges around 99% and 97% for training and testing respectively.



Name	Smoothed	Value	Step	Time	Relative
train	0.03988	0.05345	59	Sat Oct 10, 13:13:57	16h 36m 14s
validation	0.0898	0.08729	59	Sat Oct 10, 13:13:57	16h 36m 14s

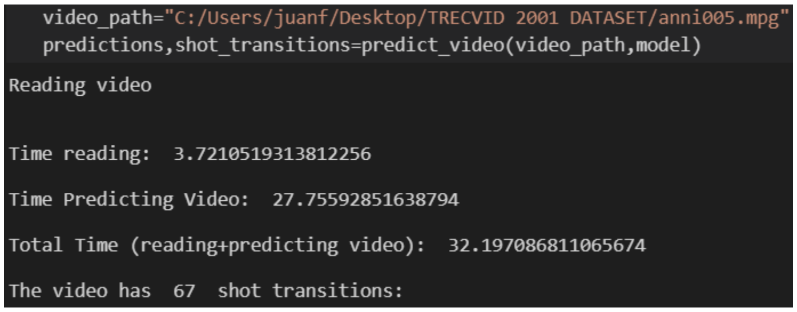
Figure 4.3 Training and testing (epoch vs loss) curves

4.4. Testing on TRECVID 2001

If we want to identify all the shots in a video, we need to evaluate the model identifying all the 20-frame windows in the video where the on-going transition is centered in the 10th and 11th frames. This means, we need to iteratively evaluate the model with a 20-frame window input that moves through the whole video. The number of iterations required are indicated in (eq 4.5).

$$\text{\# iterations for shot detection} = \text{\# of frames in video} - 19 \qquad \textbf{(4.5)}$$

The model was evaluated on the video “anni005.mpg” from the TRECVID 2001 database[26] to identify its shot composition.



```
video_path="C:/Users/juanf/Desktop/TRECVID 2001 DATASET/anni005.mpg"
predictions,shot_transitions=predict_video(video_path,model)
Reading video


Time reading:  3.7210519313812256

Time Predicting Video:  27.75592851638794

Total Time (reading+predicting video):  32.197086811065674

The video has  67  shot transitions:
```

Figure 4.4 Prediction information of video “ani005”

The video is composed by 11,363 frames with a duration of 06:18. As shown in (Fig. 4.4), the total time spend by computer to read the video into a numpy array and evaluate all the 20-frame windows was 32 seconds, finding a total of 67 windows with a centered transition.

Figures 4.5 and 4.6 show correctly detected transitions (True positives) for dissolves and hardcuts. Notice that the duration of the dissolve transition (Top Fig. 4.6) is greater than 10 frames and therefore, probably wouldn’t be detected by the 10-frames window implemented in [5].

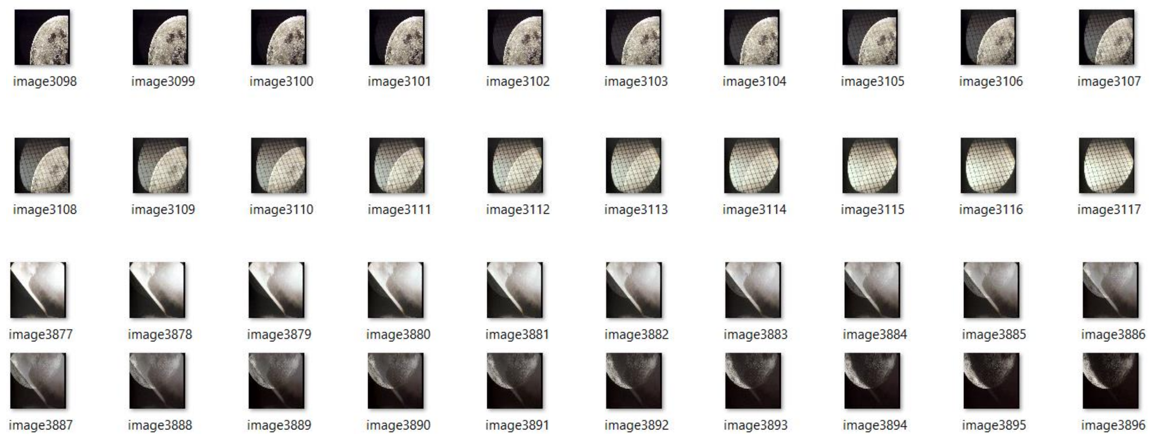


Figure 4.5 True positive dissolves

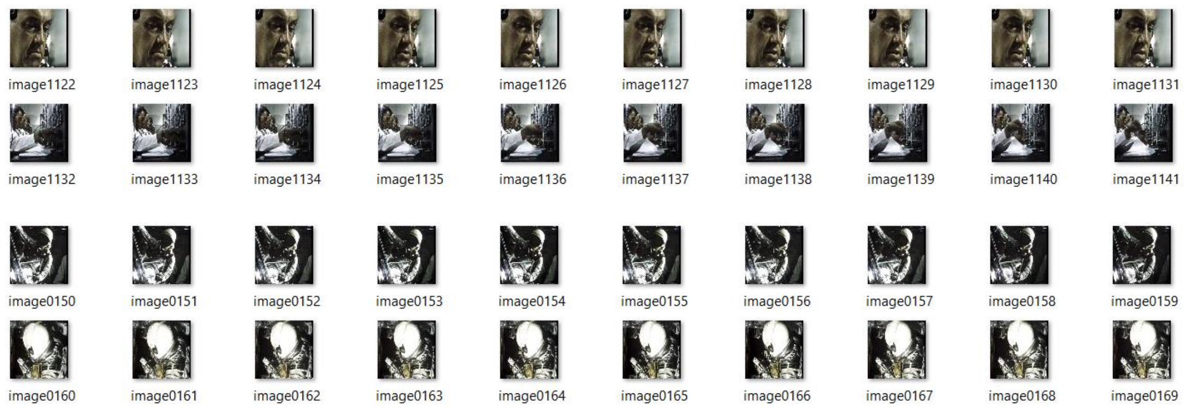


Figure 4.6 True positive hardcuts

After a manual inspection of all the detected frames, there were identified some cases where the algorithm falsely recognized a dissolve transition. An example of a false positive (the algorithm detected as "Transition" a sample that was not) is shown in (fig. 4.7).

By visually checking the 20 frames context of the image, we can realize that the algorithm falsely relates the bright changes caused by the white shirt into the scene, with the behavior of a soft transition.

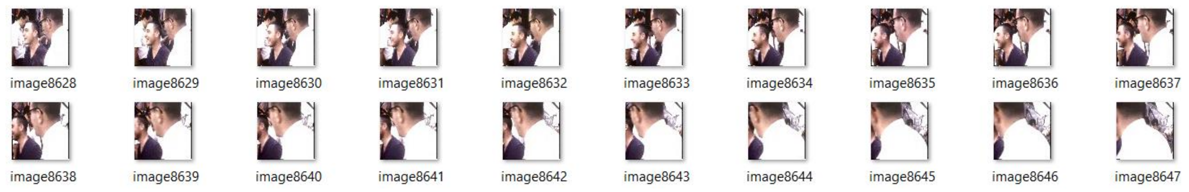


Figure 4.7 False positive dissolve

Another example of false detection of the algorithm is exposed in (fig. 4.8). In this case, the problem is a false negative (the algorithm detects as 'No transition' a sample that indeed was). Probably, the difficulty of this false classification is due to two things:

- Long smooth transition: The last frames in the sequence still contain information of the previous shot, and the transition has not concluded.
- The background of the two shots in the transition has a similar color information, so the algorithm understands a darker version of the same shot.

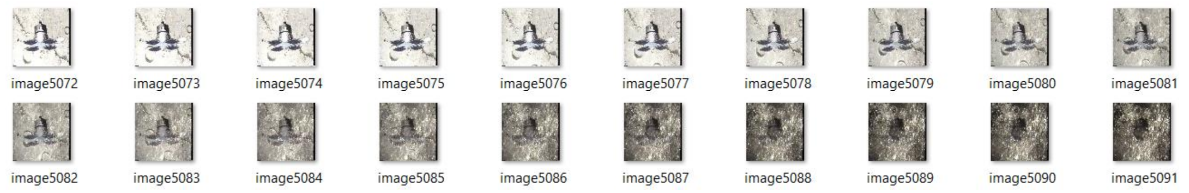


Figure 4.8 False negative dissolve

CHAPTER 5. CONCLUSIONS

Deep learning is a promising technology that has been driven by the growth of the information, where lots of data can be easily accessible, and better computational resources are available. With the change of computation paradigm, today, it is possible to buy relatively cheap GPU's in the market, or alternatively use cloud computing, where the resources are paid as needed (on-demand).

One of the main limitations of deep learning is that even with access to large amounts of data, raw data is not useful for implementing controlled tasks. Solving many special interest problems require to have some knowledge of the data (labels or ways to generate them).

Self-supervised algorithms provide an alternative to train a model with large-scale datasets that are independent of human-annotations and generate automatic pseudo-labels instead, bringing promising results on deep learning tasks.

Throughout this project, a self-supervised algorithm for the analysis of video shots was developed. The objective was accomplished by a two-stage implementation:

- Development of an algorithm that creates 20-frame samples from an input video with two types of pseudo-labels: Positive samples (transitions centered in the 10th and 11th frame) and negative samples (one shot transitions and shifted transitions).
- Implementation of a spatio-temporal fully convolutional network model based in [5], that improves the detection of large smooth transitions, by analyzing a larger temporal context.

The overall accuracy achieved in the model was greater than 97%, which is quite a good result. The model evaluation showed that most hardcuts are correctly detected, while main difficulties are presented in smooth transitions. Some of the problems detected are:

- Detection of long smooth transitions close to the window size and longer.
- Detection of fast motion shots i.e. animals' scenes, sports, natural events, etc.
- False transitions detected with large objects occlusion in several frames.

Although it was not addressed in detail, one characteristic of the model is that it is able to detect transitions larger than the 20-frame window size, by performing a lower input frame rate (subsampling). This feature can be combined with normal detection to enhance the accuracy performance on long smooth transitions.

Some possible future works may include:

- Reuse the shot detection model as a previous step of the dataset creation, to achieve more accurate transition samples.
- Implement a combined prediction algorithm that involves normal and subsampled input, to enhance the detection of long smooth transitions.
- Train the model with fast motion and occluded shots.
- Modify the topology to end in a multiclass classifier, that not only differentiates 'Transitions' with 'No Transitions', but also classifies the type of detected transition.

5.1. Sustainability considerations

Self-supervised algorithms allow to automate one of the main limitations in machine learning, which is human annotations to produce data labels. Instead, some features and knowledge of the data can be used to automatically generate the pseudo-labels, taking advantage of the fast processing capacity of computers.

Normally, the data preparation is one of the more time-consuming tasks, and a bottleneck in the process of training a model. The automation of the label creation has an important impact on a DL project, because it saves a lot of time and resources needed in the search of correctly labeled data, or even worst, the manual annotation process.

However, the implementation of DL models still has some impact associated. Even with automatized label generation, the creation of a dataset and the model training require high computational resources:

- Large storage is needed for the training and testing samples.
- Good processing capabilities for the generation of the training dataset.
- Preferably a GPU to reduce the training execution time which otherwise could last for several days.
- Long hours of uninterrupted power consumption by the computer with high performance configuration necessary to execute the required tasks.

5.2. Ethical considerations

Deep Learning (DL) was conceived with the purpose of automatize tasks that otherwise would be necessarily performed by the human action, allowing to optimize processes and improve productivity in many fields. However, it is important to keep in mind that models just act based on the limited knowledge of the data they were trained with, and even if they exceed human performance on some tasks, the results should always be supervised, and final decisions must be taken by people.

For example, A company going through financial difficulties may rely on an algorithm to decide how many people should be fired and even more, who should be fired. In such situation, it is not a good idea to drop responsibilities in an algorithm. People are responsible, while machines are just machines; and a model is just as good as the data it was trained with.

The use of technology should be responsibly employed, and as in the example before, probably a person could take a more conscious decision by reducing some unnecessary expenses to keep some job positions, and choose for example, not to fire a mother heading a family.

ACRONYMS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
Conv2D	2D Convolutional Layer
Conv3D	3D Convolutional Layer
ConvNet	Convolutional Neural Network
CTC	Connectionist Temporal Classification
DL	Deep Learning
DNN	Deep Neural Network
FN	False Negatives
Float32	Floating Point 32 bits format
FLOPS	Floating Point Operations per Second
FN	False Negatives
FP	False Positives
ML	Machine Learning
MSE	Mean Squared Error
RELU	Rectified Linear Unit
ROC AUC	Area Under the Receiver Operating Characteristic Curve
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
UInt8	Unsigned Integer 8 bits format

REFERENCES

[1] M. Ravinder, T. V. Gopal, and T. V. N. Rao, “Video indexing and retrieval - Applications and challenges,” vol. 3, p. 13, 2010.

[2] V. T. Chasanis, A. C. Likas, and N. P. Galatsanos, “Scene Detection in Videos Using Shot Clustering and Sequence Alignment,” *IEEE Trans. Multimedia*, vol. 11, no. 1, pp. 89–100, Jan. 2009, doi: 10.1109/TMM.2008.2008924.

[3] Y. Huo, Y. Wang, and H. Hu, “Effective algorithms for video shot and scene boundaries detection,” in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, Okayama, Japan, Jun. 2016, pp. 1–6, doi: 10.1109/ICIS.2016.7550913.

[4] Milind G. Padalkar, “Histogram Based Efficient Video Shot Detection Algorithms,” 2010, doi: 10.13140/RG.2.1.1590.3847.

[5] M. Gygli, “Ridiculously Fast Shot Boundary Detection with Fully Convolutional Neural Networks,” *arXiv:1705.08214 [cs]*, May 2017, Accessed: Oct. 13, 2020. [Online]. Available: http://arxiv.org/abs/1705.08214.

[6] “Deep Learning vs Machine Learning.” https://explore.mathworks.com/machine-learning-vs-deep-learning (accessed Feb. 11, 2020).

[7] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Englewood Cliffs, N.J: Prentice Hall, 1995.

[8] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.

[9] “Introducing Machine Learning.” Mathworks, [Online]. Available: https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/i/88174_92991v00_machine_learning_section1_ebook.pdf.

[10] “Introducing Deep Learning with MATLAB.” Mathworks, [Online]. Available: https://www.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf.

[11] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning Spatiotemporal Features with 3D Convolutional Networks,” *arXiv:1412.0767 [cs]*, Oct. 2015, Accessed: Oct. 18, 2020. [Online]. Available: http://arxiv.org/abs/1412.0767.

[12] “Keras: the Python deep learning API.” https://keras.io/ (accessed Oct. 16, 2020).

[13] “TensorFlow,” *TensorFlow*. https://www.tensorflow.org/?hl=es-419 (accessed Oct. 16, 2020).

[14] “Welcome — Theano 1.0.0 documentation.” http://www.deeplearning.net/software/theano/ (accessed Oct. 16, 2020).

[15] chrisbasoglu, “The Microsoft Cognitive Toolkit - Cognitive Toolkit - CNTK.” https://docs.microsoft.com/en-us/cognitive-toolkit/ (accessed Oct. 16, 2020).

[16] L. Jing and Y. Tian, “Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey,” *arXiv:1902.06162 [cs]*, Feb. 2019, Accessed: Oct. 13, 2020. [Online]. Available: http://arxiv.org/abs/1902.06162.

[17] F. Tarrés Ruiz, *Sistemas audiovisuales*. Barcelona: Edicions UPC, 2000.

[18] N. Kumar, P. Rai, C. Pulla, and C. V. Jawahar, “Video Scene Segmentation with a Semantic Similarity,” p. 12.

[19] B. Ding, H. Qian, and J. Zhou, “Activation functions and their characteristics in deep neural networks,” in *2018 Chinese Control And Decision Conference (CCDC)*, Jun. 2018, pp. 1836–1841, doi: 10.1109/CCDC.2018.8407425.

[20] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” *arXiv:1811.03378 [cs]*, Nov. 2018, Accessed: Oct. 15, 2020. [Online]. Available: http://arxiv.org/abs/1811.03378.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[22] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *arXiv:1409.4842 [cs]*, Sep. 2014, Accessed: Oct. 17, 2020. [Online]. Available: http://arxiv.org/abs/1409.4842.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015, Accessed: Oct. 17, 2020. [Online]. Available: http://arxiv.org/abs/1512.03385.

[24] “Descending into ML: Training and Loss | Machine Learning Crash Course,” *Google Developers*. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss (accessed Oct. 17, 2020).

[25] “TensorBoard,” *TensorFlow*. https://www.tensorflow.org/tensorboard (accessed Oct. 19, 2020).

[26] “Past TRECVID data and annotations by year.” https://www-nlpir.nist.gov/projects/trecvid/trecvid.data.html (accessed Oct. 15, 2020).