VIDEO SCENES CLASSIFICATION

(Violence & Non-Violence)

Project submitted in the context of the course
I3308 Senior Project under the supervision of ***Dr. Siba Haidar***

Prepared by
Mohammad Jaffal
Mohamad Bazzal – Mohamad Kassem

LEBANESE UNIVERSITY
FACULTY OF SCIENCES I
DEPARTMENT OF COMPUTER SCIENCES
2019 - 2020

Acknowledgements

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We are highly indebted to ***Lebanese University Faculty of Science, Department of Computer Sciences.*** And specially to our beloved ***Dr. Siba Haidar*** for her guidance and constant supervision through several weekly meetings and daily notes, as well as for providing the necessary information regarding the project & also for her support in completing the project.

Also, we would like to thanks ***Dr. Ihab Sbiety*** in particular for giving us some advices concerning the project due to his academic experience in the domain of Artificial intelligence.

We would also like to express our gratitude towards ***Mohamad Kurdi*** a Lebanese University graduate working currently in EverTeam company for his support and tips due to his working experience in the field of AI.

Abstract

The main goal of our project is to ***classify video scenes into 2 categories: Violence & Non-Violence.*** In order to achieve our goal, we have walked through several steps which are presented here in this document. We started with a dive deep in the field of ***Artificial intelligence*** and specifically in the domain of ***deep learning*** aiming to understand its deep concepts. Beside this we were studying ***python*** and its main libraries including ***Tensorflow*** which was useful as a main tool to build a ***CNN image classification model.***

After that, we went to more advanced topics like ***RNN & ConvLSTM*** and more interesting libraries like ***Keras.*** Which are the main tools used to build our final model. And like any other Deep Learning project, we have faced a lot of difficulties during the **Data Preparation** stage which we will discuss too in this document. And at the end we achieved our goal getting a **94%** accuracy using our model on our final **dataset**.

Table of Contents

Chapter I - Introduction

I.1. Importance for the society:

The main idea of our project is to distinguish between Violence and Non-Violence scenes. The importance is to detect any unusual behaviors or criminal attacks recorded by security cameras. So that it could be used by any company for its own security or by any governmental special forces to facilitate its searching for any violent scenes.

I.2. Artificial Intelligence (AI):

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving. It consists of several domains including “Deep Learning” that we are interested in.

At a very basic level, deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information. Observations can be in the form of images, text, or sound. The inspiration for deep learning is the way that the human brain filters information.

I.3. Deep Learning with Python:

There are many top deep learning platforms and libraries and we chose what we think is the best-of-breed platform for getting started and very quickly developing powerful and even state-of-the-art deep learning models in the Keras deep learning library for Python.

Unlike R, Python is a fully featured programming language allowing you to use the same libraries and code for model development as you can use in production. Unlike Java, Python has the SciPy stack for scientific computing and scikit-learn which is a professional grade machine library.

There are two top numerical platforms for developing deep learning models, they are Theano developed by the University of Montreal and TensorFlow developed at Google. Both were developed for use in Python and both can be leveraged by the super simple to use Keras library. Keras wraps the numerical computing complexity of Theano and TensorFlow providing a concise API that we will use to develop our own neural network and deep learning models.

I.4. Working Environment:

In our project we have used python with Jupyter Notebook (in Anaconda Distribution). **Jupyter Notebook** is an open-sourced web-based application which allows you to create and share documents containing live code, equations, visualizations, and narrative text. The **IDE** also includes data cleaning and transformation, numerical simulation, statistical modelling, data visualization, and many others.

Large deep learning models require a lot of computation time to run. You can run them on your CPU but it can take hours or days to get a result. If you have access to a GPU on your desktop, you can drastically speed up the training time of your deep learning models. GPUs are used to speed up the training of your deep learning models. So, we used **Google Colab Notebook.**

Colab is a Google research project created to help disseminate machine learning and deep learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud. It is free to use and it supports a FREE GPU that can be used to train and develop deep learning models using popular libraries such as Keras, TensorFlow and OpenCV.

In the next chapter we will introduce some necessary python libraries that are going to be very helpful to complete our project.

Chapter II - Background

After we discussed the main concept of AI and deep learning in the previous chapter, now we are going to talk about 3 important python libraries which are Theano, TensorFlow and Keras.
Actually, Theano and TensorFlow are similar libraries developed by different companies. And Keras is built on the top of TensorFlow by default (Can be changed to Theano).

II.1. Introduction to Theano

Theano is a Python library for fast numerical computation that can be run on the CPU or GPU. It is a key foundational library for deep learning in Python that you can use directly to create deep learning models.

Theano is an open source project released under the BSD license and was developed by the LISA (now MILA) group at the University of Montreal, Quebec, Canada (home of Yoshua Bengio). It is named after a Greek mathematician. At its heart Theano is a compiler for mathematical expressions in Python. It knows how to take your structures and turn them into very efficient code that uses NumPy, efficient native libraries like BLAS and native code to run as fast as possible on CPUs or GPUs.

It uses a host of clever code optimizations to squeeze as much performance as possible from your hardware. If you are into the nitty-gritty of mathematical optimizations in code, check out this interesting list. The actual syntax of Theano expressions is symbolic, which can be o↵ putting to beginners. Specifically, expression are defined in the abstract sense, compiled and later actually used to make calculations.

Theano was specifically designed to handle the types of computation required for large neural network algorithms used in deep learning. It was one of the first libraries of its kind (development started in 2007) and is considered an industry standard for deep learning research and development. [1]

II.2. Introduction to TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create deep learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

TensorFlow is an open source library for fast numerical computing. It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems, not least RankBrain in Google search [2] and the fun DeepDream project [3]. It can run on single CPU systems, GPUs as well as mobile devices and large-scale distributed systems of hundreds of machines.

II.3. Introduction to Keras

Two of the top numerical platforms in Python that provide the basis for deep learning research and development are Theano and TensorFlow. Both are very powerful libraries, but both can be difficult to use directly for creating deep learning models. In this lesson you will discover the Keras Python library that provides a clean and convenient way to create a range of deep learning models on top of Theano or TensorFlow.
Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make developing deep learning models as fast and easy as possible for research and development. It can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license. Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

- **Modularity:** A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.

- **Minimalism:** The library provides just enough to achieve an outcome, no frills and maximizing readability.

- **Extensibility:** New components are intentionally easy to add and use within the framework, intended for developers to trial and explore new ideas.

- **Python:** No separate model files with custom file formats. Everything is native Python.

After the discussion these main libraries. **In the next** chapter we are going to discuss CNN in details and build a CNN model using TensorFlow.

Chapter III - Convolutional Neural Network

Here we will define CNN then talk about its main concepts, such as Layers, activation functions… Then we will build a CNN image classification model.

III.1. Definition:

Convolutional Neural Networks are a powerful artificial neural network technique. These networks preserve the spatial structure of the problem and were developed for object recognition tasks such as handwritten digit recognition. They are popular because people are achieving state-of-the-art results on difficult computer vision and natural language processing tasks.

Convolutional Neural Networks expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Feature are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network. It is this reason why the network is so useful for object recognition in photographs, picking out digits, faces, objects and so on with varying orientation. In summary, below are some of the benefits of using convolutional neural networks:

- They use fewer parameters (weights) to learn than a fully connected network.
- They are designed to be invariant to object position and distortion in the scene.
- They automatically learn and generalize features from the input domain.

III.2. Building Blocks of Convolutional Neural Networks:

There are three types of layers in a Convolutional Neural Network:

2.1. Convolutional Layers

- **Filters:** The filters are essentially the neurons of the layer. They have both weighted inputs and generate an output value like a neuron.

- **Feature Maps:** It is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map.

2.2. Pooling Layers

The pooling layers down-sample the previous layers feature map. Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map. As such, pooling may be considering a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

They too have a receptive field, often much smaller than the convolutional layer. Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap. Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

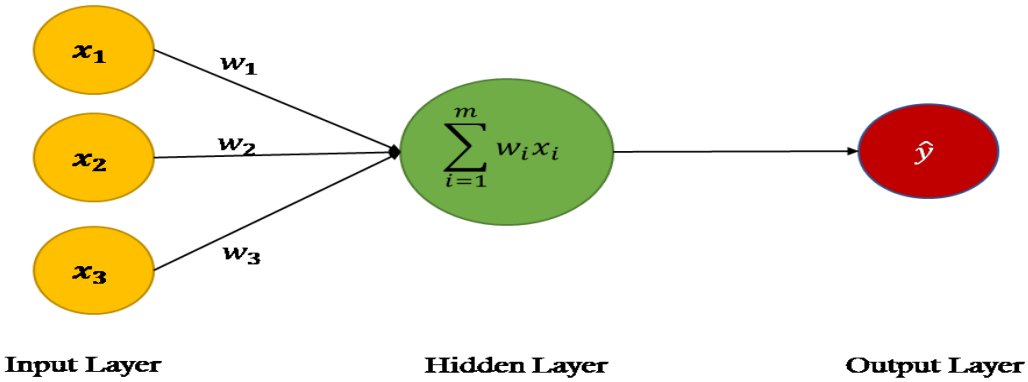
2.3. Fully Connected Layers:

Fully connected layers are the normal flat feedforward neural network layer. These layers may have a nonlinear activation function or a SoftMax activation in order to output probabilities of class predictions. Fully connected layers are used at the end of the network after feature extraction and consolidation have been performed by the convolutional and pooling layers. They are used to create final nonlinear combinations of features and for making predictions by the network.

III.3. Image Classification using CNN

3.1. Multilayer Perceptions:

- **Neuron:** The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.



- **Input Layer:** The bottom layer that takes input from your dataset is called the visible layer, because it is the exposed part of the network. Often a neural network is drawn with a visible layer with one neuron per input value or column in your dataset. These are not neurons as described above, but simply pass the input value to the next layer.

- **Hidden Layer:** Layers after the input layer are called hidden layers because they are not directly exposed to the input. The simplest network structure is to have a single neuron in the hidden layer that directly outputs the value. Given increases in computing power and efficient libraries, very deep neural networks can be constructed. Deep learning can refer to having many hidden layers in your neural network. They are deep because they would have been unimaginably slow to train historically, but may take seconds or minutes to train using modern techniques and hardware.

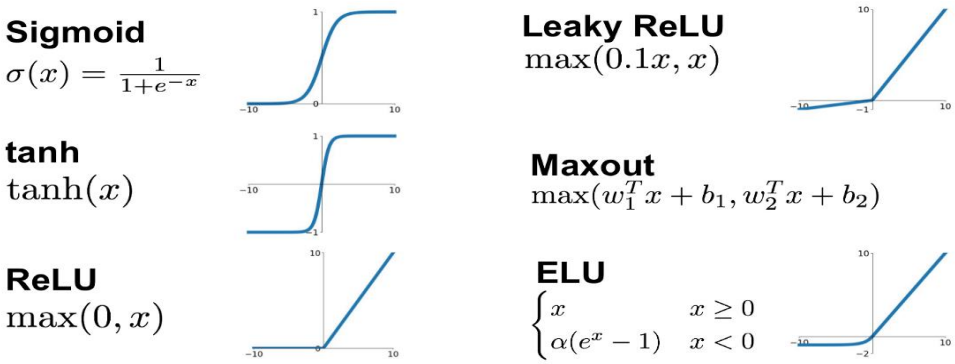
- **Output Layer:** The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem. The choice of activation function in the output layer is strongly constrained by the type of problem that you are modeling.

- **Neuron Weights:** You may be familiar with linear regression, in which case the weights on the inputs are very much like the coefficients used in a regression equation. Like linear regression, each neuron also has a bias which can be thought of as an input that always has the value 1.0 and it too must be weighted. For example, a neuron may have two inputs in which case it requires three weights. One for each input and one for the bias.
- Weights are often initialized to small random values, such as values in the range 0 to 0.3, although more complex initialization schemes can be used. Like linear regression, larger weights indicate increased complexity and fragility of the model. It is desirable to keep weights in the network small and regularization techniques can be used.

- **Activation Function:** The weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and the strength of the output signal. Historically simple step activation functions were used where if the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0.
  Traditionally nonlinear activation functions are used. This allows the network to combine the inputs in more complex ways and in turn provide a richer

capability in the functions they can model. Nonlinear functions like the logistic function also called the sigmoid function were used that output a value between 0 and 1 with an s-shaped distribution, and the hyperbolic tangent function also called Tanh that outputs the same distribution over the range -1 to +1. More recently the rectifier activation function has been shown to provide better results.

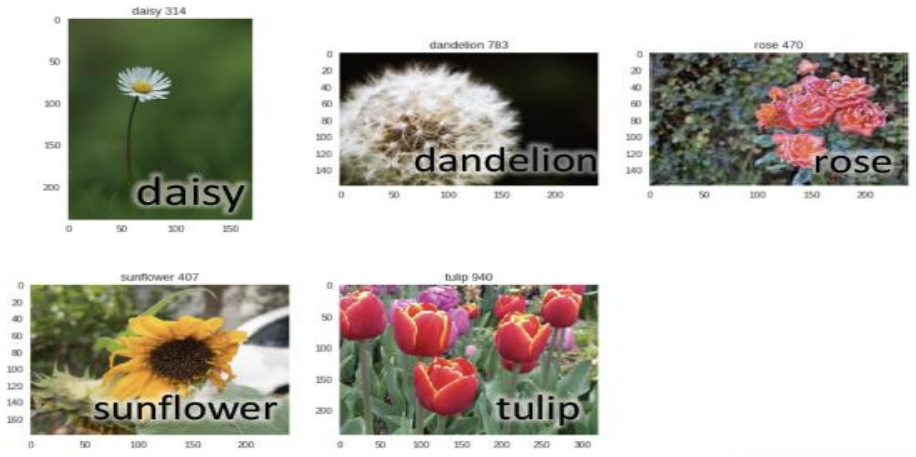
Some activation functions:



3.2. Flowers Classification Lab:

We have worked on Flowers Classification Lab using Tensorflow from Huawei. The main aim of the lab is to classify a given flower image into one among 5 main flower types:

- flower classification:
  - daisy | dandelion | rose | sunflower| tulip
- CNN based on TensorFlow



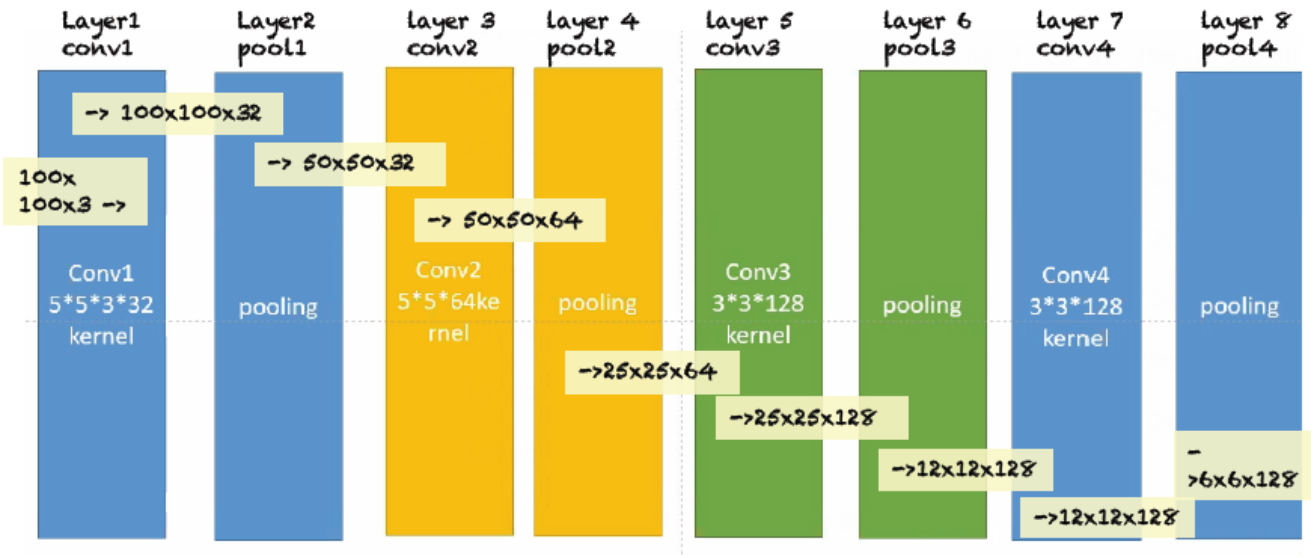
1 Data Preparation:

All images in this lab are provided to us by Huawei under the Creative Commons By-Attribution License, available at: [4]

Our dataset consists of 3670 flower images divided as follows:

- daisy 633 images
- dandelion 898 images
- roses 641 images
- sunflowers 699 images
- tulips 799 images

2 Layers Construction:



The entire model has 11 hidden layers:

- 4 convolutional layers
- 4 pooling layers
- 3 full connection layers

3 Results:

After running this lab several times, we have got the following results with an accuracy **60%.** Although in this practical lab the accuracy wasn't all what we were looking for. But the main purpose of this lab was to learn, understand the concepts, test the working environment and train a deep learning model for the first time as we didn't have any previous background or experiences in the domain.

```
-----------------------------------------------------
  batch acc: 1.000000      batch loss: 49.490753
  batch acc: 0.968750      batch loss: 50.738686
  batch acc: 0.937500      batch loss: 53.826225
  batch acc: 0.968750      batch loss: 52.837227
  batch acc: 0.906250      batch loss: 56.359886
  batch acc: 0.906250      batch loss: 55.238567
  batch acc: 0.968750      batch loss: 55.037094
  batch acc: 0.875000      batch loss: 54.057114
  batch acc: 0.937500      batch loss: 53.707466
  batch acc: 0.937500      batch loss: 57.846916
   train loss: 53.217087
   train acc: 0.938874
   validation loss: 99.387895
   validation acc: 0.603693
-----------------------------------------------------
```

The next challenge was about video classification and not images. Unlike image a video file has an additional dimension, which is time. So, **in the next chapter** we will talk about RNN, LSTM and ConvLSTM that are important concepts used to work with videos.

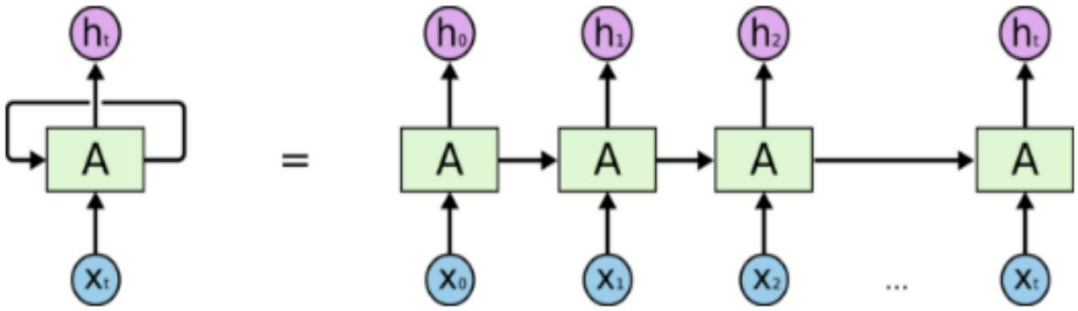
Chapter IV - Video Classification

In the following we will start defining RNN and how its back propagation works, then discuss the problem of gradients vanishing or exploding and how to solve it using LSTM and ConvLSTM.

IV.1. Recurrent Neural Network (RNN)

This is another type of neural network that is dominating difficult machine learning problems that involve sequences of inputs called recurrent neural networks. Recurrent neural networks have connections that have loops, adding feedback and memory to the networks over time. This memory allows this type of network to learn and generalize across sequences of inputs rather than individual patterns.

Recurrent Neural Networks or RNNs are a special type of neural network designed for sequence problems. Given a standard feedforward Multilayer Perceptron network, a recurrent neural network can be thought of as the addition of loops to the architecture. For example, in a given layer, each neuron may pass its signal latterly (sideways) in addition to forward to the next layer. The output of the network may feedback as an input to the network with the next input vector. And so on…



An unrolled recurrent neural network.

The recurrent connections add state or memory to the network and allow it to learn broader abstractions from the input sequences. The field of recurrent neural networks is well established with popular methods. For the techniques to be effective on real problems, two major issues needed to be resolved for the network to be useful:

1.1. How to train the network with Back propagation

The staple technique for training feedforward neural networks is to back propagate error and update the network weights. Back propagation breaks down in a recurrent neural network,

because of the recurrent or loop connections. This was addressed with a modification of the Back propagation technique called Back propagation Through Time or BPTT. Instead of performing back propagation on the recurrent network as stated, the structure of the network is unrolled, where copies of the neurons that have recurrent connections are created.

For example, a single neuron with a connection to itself (A --> A) could be represented as two neurons with the same weight values (A --> B). This allows the cyclic graph of a recurrent neural network to be turned into an acyclic graph like a classic feedforward neural network, and Back propagation can be applied.

1.2. How to stop gradients vanishing or exploding during training

When Back propagation is used in very deep neural networks and in unrolled recurrent neural networks, the gradients that are calculated in order to update the weights can become unstable. They can become very large numbers called exploding gradients or very small numbers called the vanishing gradient problem. These large numbers in turn are used to update the weights in the network, making training unstable and the network unreliable.

This problem is alleviated in deep Multilayer Perceptron networks through the use of the Rectifier transfer function, and even more exotic but now less popular approaches of using unsupervised pre-training of layers. In recurrent neural network architectures, this problem has been alleviated using a new type of architecture called the Long Short-Term Memory Networks that allows deep recurrent networks to be trained.

IV.2. Long Short-Term Memory (LSTM) Networks

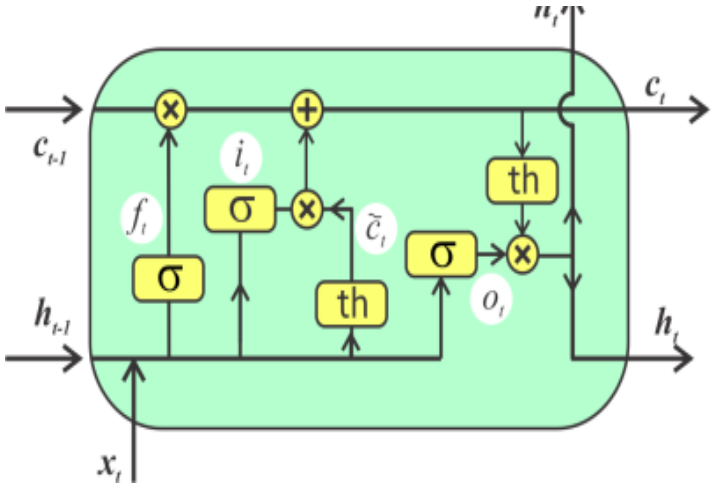
A powerful type of Recurrent Neural Network called the Long Short-Term Memory (LSTM) Network has been shown to be particularly effective when stacked into a deep configuration, achieving state-of-the-art results on a diverse array of problems from language translation to automatic captioning of images and videos.

The Long Short-Term Memory or LSTM network is a recurrent neural network that is trained using Back propagation Through Time and overcomes the vanishing gradient problem. As such it can be used to create large (stacked) recurrent networks, that in turn can be used to address difficult sequence problems in machine learning and achieves state-of-the-art results. Instead of neurons, LSTM networks have memory blocks that are connected into layers.[5]

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A unit operates upon an input sequence and each gate within a unit uses the sigmoid activation function to

control whether they are triggered or not, making the change of state and addition of information flowing through the unit conditional. There are three types of gates within a memory unit:

- **Forget Gate:** conditionally decides what information to discard from the unit.
- **Input Gate:** conditionally decides which values from the input to update the memory state.
- **Cell Update:** conditionally decides what information to keep.
- **Output Gate:** conditionally decides what to output based on input and the memory of the unit.



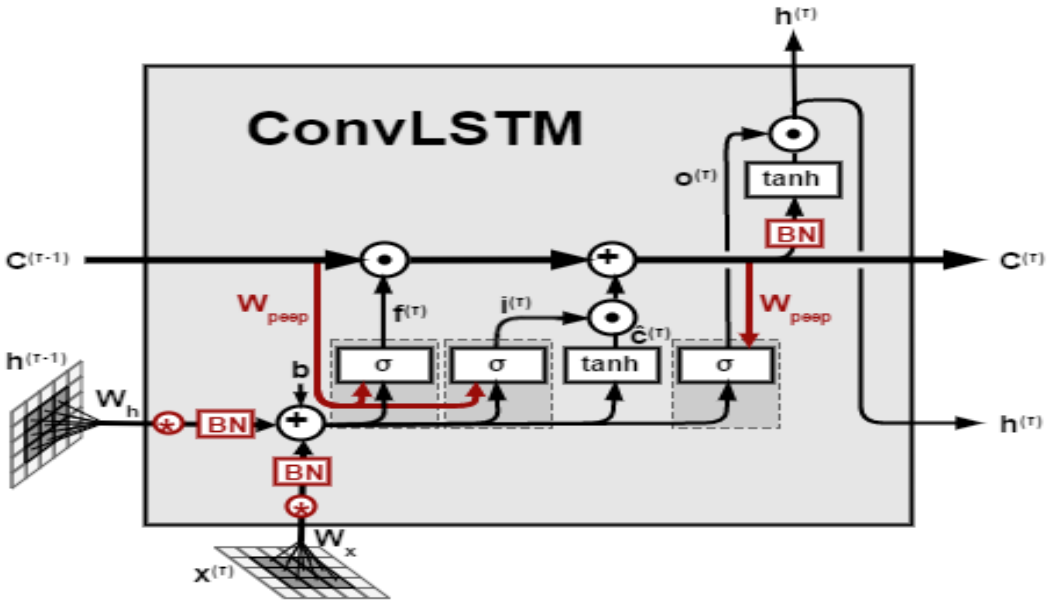
$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

IV.3. Convolutional LSTM (ConvLSTM)

ConvLSTM is a variant of LSTM (Long Short-Term Memory) containing a convolution operation inside the LSTM cell. Both the models are a special kind of RNN, capable of learning long-term dependencies.

ConvLSTM replaces matrix multiplication with convolution operation at each gate in the LSTM cell. By doing so, it captures underlying spatial features by convolution operations in multiple-dimensional data.

The main difference between ConvLSTM and LSTM is the number of input dimensions. As LSTM input data is one-dimensional, it is not suitable for spatial sequence data such as video, satellite, radar image data set. ConvLSTM is designed for 3-D data as its input. [6]



All the above information will be used **in the next chapter** in practice. Where we will start building our video classification model using RNN-ConvLSTM and Keras as a main library to write our python code. Also, we will see the used dataset.

Chapter V - Video Classification in Keras using ConvLSTM (Violence & NonViolence)

Our main project is about classifying video scenes into 2 main categories:

1. **Violence Scenes.**
2. **Non-Violence Scenes.**

In order to achieve our goal, we have passed through several steps.

V.1. Required Libraries:

We have used the following libraries in order to implement our code:

Required Libraries

```
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping

import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import keras_metrics as km

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import multilabel_confusion_matrix

from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping
```

V.2. Data Preparation:

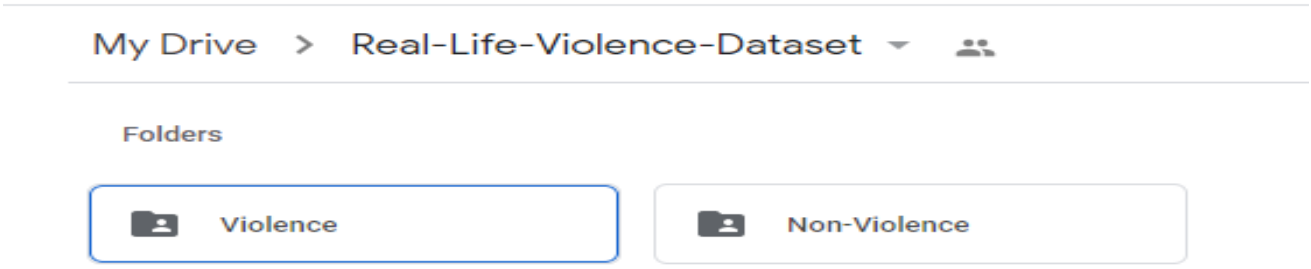
At the begging we have started working with a small dataset of 120 videos only, which is the **SDHA** dataset. But we did not reach the targeted accuracy with that dataset because of its small size. Then we kept searching and preparing a suitable dataset for many days before we have got the suitable one: **Real-Life-Violence-Dataset** from Kaggle: [7]

Our Dataset Contains a total of 2000 Videos classified into 1000 Violence and 1000 Non-Violence videos collected from YouTube videos, violence videos in our dataset contain many real street fights situations in several environments and conditions.
Also non-violence videos from our dataset are collected from many different human actions like sports, eating, walking …etc.

- Video length lasts from 2 to 7 seconds.
- Resolution: Frame Width 1920 * Frame Height 1080.
- Frame Rate: 25 frames/second.

Dataset link: [8]

Our dataset on the drive: [9]



The following piece of code was used to mount (connect) our Google Colab with Google Drive:

```
[ ]  from google.colab import drive
     drive.mount('/content/drive/')

     Go to this URL in a browser: https://accounts.google.com/

     Enter your authorization code:
     ··········
     Mounted at /content/drive/
```

Our dataset is saved in the drive under the name Real-Life-Violence-Dataset.
The dataset is divided into 2 classes Violence and Non-Violence.

We extracted 60 frames from each video where each frame has the shape of (64,64,3).

```
data_dir = "/content/drive/My Drive/Real-Life-Violence-Dataset"
img_height , img_width = 64, 64
seq_len = 60
classes = ["Non-Violence","Violence"]
```

The **frames_extraction** function takes a video path as input and outputs a list of frames.
The frames extraction is taken in a uniform period of time (equal time difference between each two screenshots taken) defined by the variable **milli** in the following code:

```
def frames_extraction(path):
    # Path to video file
    myframe=seq_len
    vidObj = cv2.VideoCapture(path)
    frames_list = []
    count = 1
    # checks whether frames were extracted
    success = 1
    fps = vidObj.get(cv2.CAP_PROP_FPS)
    frame_count = vidObj.get(cv2.CAP_PROP_FRAME_COUNT)
    duration = (frame_count) / fps
    second = 0
    vidObj.set(cv2.CAP_PROP_POS_MSEC, second * 1000) # optional
    success, image = vidObj.read()
    milli=(duration*1000)/myframe
    while success and second<=duration:
        second += milli/1000
        vidObj.set(cv2.CAP_PROP_POS_MSEC, second * 1000)
        image = cv2.resize(image, (img_height, img_width))
        frames_list.append(image)
       # cv2.imwrite("test/frame%d.jpg" % count,image)
        count += 1
        success, image = vidObj.read()
    if len(frames_list)==(seq_len-1) :
        frames_list.append(frames_list[seq_len-2])
        print("----------------------------------------")
    return frames_list
```

The **create_data** function takes the dataset path as an input and outputs a pair of (X, Y):

- X: is an array where each entry contains a list of frames of shape (nb_vid, nb_frames, img_height, img_width, channel).
- Y: is an array where each entry contains the corresponding Label of each entry in X of the shape (nb_vid, nb_classes).

For example Y[0] = [0 , 1] which mean Violence OR [1, 0] which means Non-Violence.

```
def create_data(input_dir):
    X = []
    Y = []
    i=0
    classes_list = os.listdir(input_dir)
    for c in classes_list:
        print(c)
        files_list = os.listdir(os.path.join(input_dir, c))
        for f in files_list:
           if i%100==0 :
                print(i)
           i=i+1
           frames = frames_extraction(os.path.join(os.path.join(input_dir, c), f))
           if len(frames)==seq_len:
                X.append(frames)
                y = [0]*len(classes)
                y[classes.index(c)] = 1
                Y.append(y)
    X = np.asarray(X)
    Y = np.asarray(Y)
    return X, Y
```

Now it's the time to create our data using the create_data function. And then split (divide) this data into Training and Testing data using the **train_test_split** function from **Sklearn.model_selection.** Of ratio 0.8 for training and 0.2 for testing with shuffling.

- Training Part is used to train our model.
- Testing Part is used to test and validate our model.

```
X, Y = create_data(data_dir)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, shuffle=True, random_state=0)
print(X.shape)
print(Y.shape)
```

```
(4226, 60, 64, 64, 3)
(4226, 2)
```

V.3. Model Creation:

In this section we are going to create our model using build-in functions from Keras.

Dropout is a regularization technique for neural network models proposed by Srivastava. in their 2014 paper Dropout: A Simple Way to Prevent Neural Networks from Overfitting[1]. Dropout is a technique where randomly selected neurons are ignored during training. They are dropped-out randomly. This means that their contribution to the activation of downstream neurons are temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

Models in Keras are defined as a sequence of layers. We create a Sequential model and add layers one at a time until we are happy with our network topology. The first thing to get right is to ensure the input layer has the right number of inputs.

After creating the model:

- We will add ConvLSTM to the input layer with weight (3,3) of 64 nodes.
- We make dropout of ratio 0.2
- We transform the nodes into vector.
- In the fully-Connected layer we reduce the vector size into 256 using the "**relu"** activation function.
- We make another dropout of ratio 0.3.
- We reduce the vector size again into 2 outputs using the "**softmax"** activation function.

```
model = Sequential()
model.add(ConvLSTM2D(filters = 64, kernel_size = (3, 3), return_sequences = False, data_format = "channels_last",\\
                     input_shape = (seq_len, img_height, img_width, 3)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(2, activation = "softmax"))

model.summary()
```

V.4. Model Training:

4.1. Compile Model:

Now that the model is defined, we can compile it. Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware. When compiling, we must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to make predictions for this problem.

Optimizers: We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training. In this case we will use logarithmic loss, which is **categorical_crossentropy**.
We will also use the efficient gradient descent algorithm "Stochastic Gradient Descent" **SGD** because it is efficient after several testing.

Learning Rate: Adapting the learning rate for our SGD optimization procedure can increase performance and reduce training time. Sometimes this is called learning rate annealing or adaptive learning rates. Here we will call this approach a learning rate schedule, were the default schedule is to use a constant learning rate to update network weights for each training epoch. Where we have taken it 0.001.

```
opt = keras.optimizers.SGD(lr=0.001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=["accuracy"])
```

4.2. Fit Model:

We have defined our model and compiled it ready for efficient computation. Now it is time to execute the model on some data. We can train or fit our model on our loaded data by calling the fit() function on the model.

The training process will run for a fixed number of iterations through the dataset called **epochs**, that we must specify using the nb epoch argument. The **number of epochs** is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

We can also set the number of instances that are evaluated before a weight update in the network is performed called the batch size and set using the batch size argument.

The **batch size** is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated. For example, 16 batch means that after every 16 videos one forward/backward propagation.

For this problem we will run for a number of **40 epochs** and we used **batch sizes of 8 & 16**. Again, these can be chosen experimentally by trial and error.

Keras callbacks are a group of functions that returns information from a training algorithm while training is taking place.
EarlyStopping Callback is used to stop the training of the model when the training is no longer beneficial for the validation accuracy of the model. [10]

```
earlystop = EarlyStopping(patience=7)
callbacks = [earlystop]
history = model.fit(x = X_train, y = y_train, epochs=40, batch_size = 16, shuffle=True, validation_split=0.2, callbacks=callbacks)
```

V.5. Evaluate Model:

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset. This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data. For this reason, we are going to test the accuracy of our model on the Testing dataset that contains new data that the model hasn't seen before. By calling the **predict()** function on the finalized model. The **predict()** function takes an array of one or more data instances and returns an array of predicted Label (0 or 1).

The **Classification_report** is used to observe the information of the model evolution on the testing dataset, and in particular to check the accuracy.

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis = 1)
Y_test = np.argmax(y_test, axis = 1)
print(classification_report(Y_test, y_pred))
```

V.6. Results:

In order to achieve our goal in this project we have done a lot of runs (more than 30 runs) using several variations in the dataset and the **hyperparameters** to reach the highest possible accuracy.

But before showing any running-samples let's first discuss some important terms:

Predicted / Observed	*Yes*	*No*	Total
Yes	*TP*	*FN*	*P*
No	*FP*	*TN*	*N*
Total	P'	N'	$P+N$

Confusion matrix

- **[P]** Positive: Are examples that involves the major interesting classes.
- **[N]** Negative: Other examples that doesn't involve major interesting classes.
- **[TP]** True Positive: positive examples that are correctly classified by the classifier.
- **[TN]** True Negative: negative examples that are correctly classified by the classifier.
- **[FP]** False Positive: negative examples that are incorrectly marked as positive examples.
- **[FN]** False Negative: positive examples that are incorrectly marked as negative examples.

And some formulas that we will need to understand and interpret the results:

- accuracy = (TP + TN) / (P + N)
- recall = TP/P
- precision = TP / (TP + FP)
- f-score = (2 * precision * recall) / (precision + recall)

And finally, it is time to show some running samples among lots of runs that we have worked on, including the final run.

The following are some running samples:

6.1. Run 1:

- **Dataset:** SDHA dataset (120 videos).
- **seq_len:** 50
- **validation_split:** 0.2
- **Optimizer:** SGD & **Learning Rate:** 0.001
- **Batch Size:** 8
- **Number of Epochs:** 15

We achieved an accuracy of **50%** as a result of this run:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	12
1	0.50	1.00	0.67	12
accuracy			0.50	24
macro avg	0.25	0.50	0.33	24
weighted avg	0.25	0.50	0.33	24

6.2. Run 2

- **Dataset**: "Real-Life-Violence-Dataset + SDHA" dataset (2120 videos).
- **seq_len:** 30
- **validation_split:** 0.2
- **Optimizer:** SGD & **Learning Rate:** 0.001
- **Batch Size:** 8
- **Number of Epochs:** 40

We achieved an accuracy of **81%** as a result of this run:

	precision	recall	f1-score	support
0	0.90	0.71	0.79	212
1	0.76	0.92	0.83	212
accuracy			0.81	424
macro avg	0.83	0.81	0.81	424
weighted avg	0.83	0.81	0.81	424

6.3. Run 3:

- **Dataset**: “Real-Life-Violence-Dataset + SDHA” dataset (2120 videos).
- **seq_len:** 60
- **validation_split:** 0.2
- **Optimizer:** SGD & **Learning Rate:** 0.0001
- **Batch Size:** 8
- **Number of Epochs:** 40

We achieved an accuracy of **85%** as a result of this run:

	precision	recall	f1-score	support
0	0.90	0.79	0.84	205
1	0.82	0.92	0.87	218
accuracy			0.85	423
macro avg	0.86	0.85	0.85	423
weighted avg	0.86	0.85	0.85	423

6.4. Run 4:

- **Dataset**: Real-Life-Violence-Dataset dataset (2000 videos).
- **seq_len:** 60
- **validation_split:** 0.2
- **Optimizer:** SGD & **Learning Rate:** 0.001
- **Batch Size:** 8
- **Number of Epochs:** 40
-

We achieved an accuracy of **90%** as a result of this run:

	precision	recall	f1-score	support
0	0.91	0.89	0.90	201
1	0.89	0.91	0.90	198
accuracy			0.90	399
macro avg	0.90	0.90	0.90	399
weighted avg	0.90	0.90	0.90	399

6.5. Final Run:

After achieving an accuracy of 90% we had to think of a new way to get a higher accuracy (other than changing hyperparameters). So, we got that idea to manipulate the dataset, by doubling its size through reversing the videos by the following piece of code:

First we deleted the 2 lines of code in the create_data() function:

```
X = np.asarray(X)
Y = np.asarray(Y)
```

Then we added the code below:

```
                y = [0]*len(classes)
                y[classes.index(c)] = 1
                Y.append(y)
    return X, Y
```

```
X, Y = create_data(data_dir)
f=[]
p=[]
for i in range(len(X)) :
    data=X[i]
#   data.reverse()
    data = data[::-1]
    f.append(data)
    f.append(X[i])
    p.append(Y[i])
    p.append(Y[i])

f=np.asarray(f)
print(f.shape)
p=np.asarray(p)
print(p.shape)
```

And we used the following hyperparameters:

- **Dataset**: “Real-Life-Violence-Dataset + SDHA” dataset (4240 videos).
- **seq_len:** 60
- **validation_split:** 0.2
- **Optimizer:** SGD & **Learning Rate:** 0.001
- **Batch Size:** 16
- **Number of Epochs:** 40

We achieved an accuracy of ***94%*** as a result of this run:

	precision	recall	f1-score	support
0	0.95	0.93	0.94	423
1	0.93	0.95	0.94	423
accuracy			0.94	846
macro avg	0.94	0.94	0.94	846
weighted avg	0.94	0.94	0.94	846

Chapter VI - Conclusion:

The accomplished work relies inside a very popular and trendy domain. Also, the application on violent video automatic detection is of high importance due to the increasing interest in terrorist defense and precaution measures. And classification into Violence & Non-Violence makes it more important as it could be used in several security domains for protection from terrorist attacks.

We are happy to be able to get an accuracy of 94% which makes our model very reliable by comparison to the other similar models with lower accuracy. And this result was the fruit of many trials and efforts of choosing the right dataset, preparing it and tuning the hyperparameters.

We are glad that we gained the previous experience which have strengthen our abilities in the domain of computer science and especially in the field of AI which was new to us. Also, we have learned more stuff through this journey, such as learning the python programing language, and acquiring new teamwork and online working skills.

VI.1. Future Considerations:

Actually, there are many features that could be added to our project in the future to make it more useful. For instance, we could build a GUI that could make the project more interactive and helpful especially for non-experienced users who would need this model. Also, we could update the model through several techniques and tools such as using image processing to help in easier detection for the violent moves of the interacting human bodies.

The senior project was a great experience that have opened a new horizon for us into a new important field which is AI. This domain seemed really interesting to us, and thus encouraged us to dive deeper within this field through learning more and continue our higher master degrees in this domain.

Chapter VII - References:

1- Article at: http://deeplearning.net/software/theano/optimizations.html#optimizations

2- https://en.wikipedia.org/wiki/RankBrain

3- https://en.wikipedia.org/wiki/DeepDream

4- https://creativecommons.org/licenses/by/2.0/

5- Article at: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

6- https://www.quora.com/What-is-the-difference-between-ConvLSTM-and-CNN-LSTM

7- https://www.kaggle.com/

8- https://www.kaggle.com/mohamedmustafa/real-life-violence-situations-dataset

9- https://drive.google.com/drive/folders/1cF0Wd3UsVLP6yislkWZbefNuR5D9U0sd

10- Article at: https://machinelearningmastery.com/