Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Deception Detection

August 2020

Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Deception Detection

By:

Eman Alaa Farag	Computer Science
Esraa Yasser	Computer Science
Menna Allah Mostafa	Computer Science
Menna Mohie El-deen	Computer Science
Mirna Muhammad	Computer Science

Under Supervision of:

Dr. Ahmed Salah,
Computer Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

T.A. Abdelrahman Shaker,
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Acknowledgement

We would like to use this opportunity to express our gratitude to everyone who supported us throughout this project. We are thankful for their indispensable guidance, invaluably constructive criticism, friendly advice, and the most obliged provision of their genuine and illuminating views.

We would like to offer our special thanks to our supervisor Dr. Ahmed Salah for his help and support.

We would also like to extend our thanks to T.A. Abdelrahman Shaker for guiding and helping us throughout the whole course of the project; his help was invaluable to us.

Finally, we must give some credit to professor Andrew Ng for providing a platform for students and passionate learners to expand their knowledge in machine learning and deep learning through his courses on Coursera.

Abstract

Deception affects many areas in our day-to-day life. It threatens our security on a global scale. And for some people, it's like second nature when they find themselves in tough situations.

In crucial areas, such as police investigations and law enforcement, detecting such acts falls solely on humans. But humans may exhibit biases and lack of judgment. And it takes a great-deal for a person to identify whether the information presented to them is the truth or not.

It is believed that when a person tries to deceive another, some involuntary expressions may reveal their doing. Such expressions are known as microexpressions as they occur in 1/25th of a second, and are very difficult for the human's eye to catch.

With the nature of the task of deception detection, automating it is of great concern due to its scale. Automatic deception detection is an important task that has gained momentum in computational linguistics due to its potential applications.

In our research we explored some of the benchmarks to try and enhance them. We worked from scratch to build similar models and test ourselves to see the effect of a person's visual features and their microexpressions combined. We achieved an accuracy of 78%.

Table of Contents

Acknowledgement 3
Abstract 4
List of Figures 8
List of Abbreviations 9
1. Introduction 10
  1.1 Motivation 10
  1.2 Problem Definition 10
  1.3 Objectives 11
    1.3.1 What was done for achieving these objectives 11
  1.4 Timeplan 12
  1.5 Document Organization 13
2. Background 14
  2.1 Neural Networks 14
    2.1.1 Neural Network Architecture 15
    2.1.2 Types of Artificial Neural Networks 17
      2.1.2.1 Feed Forward Neural Networks 17
        2.1.2.1.1 Single-Layer Perceptron 18
        2.1.2.1.2 Multi-Layer Perceptron (MLP) 19
      2.1.2.2 Recurrent Neural Networks 20
    2.1.3 Backpropagation 21
  2.2 Deep Neural Networks 22
  2.3 Convolutional Neural Networks (CNNs/ConvNets) 23
    2.3.1 Convolution Layer 24
    2.3.2 Strides 25
    2.3.3 Padding 25
    2.3.4 Pooling Layer 25
    2.3.5 Activation Functions 26
      2.3.5.1 Hyperbolic Tangent function (Tanh) 26
      2.3.5.2 Non-Linearity (ReLU) 26
    2.3.6 Fully Connected Layer 27
    2.3.7 Dropout Layer 27
    2.3.8 3D Convolutions 28
  2.4 Optimizer & Loss functions 28
    2.4.1 Gradient Descent 29
    2.4.2 Stochastic Gradient Descent (SGD) 29
    2.4.3 Mini-Batch Gradient Descent 29
    2.4.4 Momentum 30
    2.4.5 RMSprop Optimizer 30
    2.4.6 ADAM Optimizer 30
  2.5 Object Recognition 31
    2.5.1 Object Recognition tasks 31
    2.5.2 Image Classification (VGG) 32
      2.5.2.1 Architecture 33

2.5.3 Object Detection 34
2.5.3.1 Sliding Window Approach 34
2.5.3.2: R-CNN 34
2.5.3.3 Fast R-CNN 35
2.5.3.4 Faster R-CNN 36
2.5.3.5 You Only Look Once (YOLO) 36
2.6 Optical Flow 40
2.7 Micro-Expressions 41
2.8 Related Work 41
2.8.1 Deep Learning Approach for Multimodal Deception Detection 41
2.8.2 Deception Detection in Videos 41
2.8.3 Facial Action Units 42
2.8.4 Face-Focused Cross-Stream Network 42
2.8.5 Design & Development of a Lie Detection System 42
3. Analysis and Design 43
3.1 System Overview 43
3.1.1 System Architecture 43
3.1.2 System Users 43
3.2 System Analysis & Design 44
3.2.1 Use Case Diagram 44
3.2.1.1 Diagram 44
3.2.1.2 Description of Use Cases 44
3.2.2 Flow of Events 45
3.2.3 Sequence Diagram 45
4. Dataset 46
5. Implementation 47
5.1 Environment Setup & Tools 47
5.1.1 Environments 47
5.1.2 Packages and Libraries 47
5.2 Data Preprocessing 49
5.2.1 Subject-based Splitting 49
5.2.2 Train-Test Splitting 50
5.3 Region of Interest Extraction Module 50
5.4 Data Clipping Module 52
5.5 Visual Feature Extraction Module 52
5.5.1 3D-CNN 52
5.5.2 Visual Feature Extraction MLP 55
5.6 Data Augmentation Module 56
5.6.1 First trial: (Extending videos with noise) 56
5.6.2 Analysing dataset 58
5.6.3 Second trial: (Extend/shrink video to average) 59
5.7 Micro-Expressions Extraction Module 60
5.7.1 VGG 60
6. Experiments 62
6.1 Micro-expressions extraction with VGG 62

6.2 Visual Features Extraction 63
6.2.1 3D-CNN 63
6.2.2 MLP 64
6.3 Visual Features + Micro-expressions 65
7. User Manual 66
8. Conclusion & Future Work 68
8.1 Conclusion 68
8.2 Future work 68
References 70

List of Figures

Figure 1.1: Time Plan 12
Figure 2.1: Neural Network general architecture 14
Figure 2.2: Single Neuron Architecture 16
Figure 2.3: Feed Forward Neural Network 17
Figure 2.4: Linear Classifier 18
Figure 2.5: Single Perceptron 19
Figure 2.6: Multi-Layer Perceptron 20
Figure 2.7: Recurrent Neural Network 20
Figure 2.8: Non-linear separable data to linear separable data 22
Figure 2.9: Comparison between MLP and CNN 23
Figure 2.10: CNN Architecture 24
Figure 2.11: Max Pooling Layer 25
Figure 2.12: Hyperbolic Tangent function (Tanh) Graph 26
Figure 2.13: Rectified Linear Unit 26
Figure 2.14: Fully Connected Layer 27
Figure 2.15: Dropout Layer 27
Figure 2.16: 3D Convolution 28
Figure 2.17: Comparison between SGD and Mini-Batch GD 29
Figure 2.18: Effect of adding Momentum-term 30
Figure 2.19: Convergence of Different Optimizers 31
Figure 2.20: Overview of Object Recognition Computer Vision Tasks 32
Figure 2.21: VGG-19 Architecture 33
Figure 2.22: R-CNN: Regions with CNN features 34
Figure 2.23: Fast R-CNN 35
Figure 2.24: Faster R-CNN 36
Figure 2.25: The YOLO Detection System 37
Figure 2.26: Yolo's Model 39
Figure 2.27: Yolo's Architecture 39
Figure 2.28: Optical Flow 40
Figure 3.1: System Architecture 43
Figure 3.2: Use Case Diagram 44
Figure 3.3: Sequence Diagram 45
Figure 4.1: Dataset Screenshots 48
Figure 4.2: Microexpressions Annotations 48
Figure 5.1: Subject-based splitting 49
Figure 5.2: 3D CNN Architecture 53
Figure 5.3: Frame without noise 57
Figure 5.4: Frame with noise 58
Figure 7.1: Welcome Page 66
Figure 7.2: Video Selection 66
Figure 7.3: Uploading Video 67
Figure 7.4: Classifier's Verdict 67

List of Abbreviations

Abbreviation	Stands for
ADAM	Adaptive Moment Optimization
ADD	Automated Deception Detection
AU	Action Units
CNN	Convolutional Neural Network
Conv	Convolutional
Convnet	Convolutional network
EVS	Embedded Vision System
FC	Fully-Connected
FFCSN	Face Focused Cross Stream Network
fps	frames per second
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
ML	Machine Learning
MLP	Multi Layer Perceptron
NMS	Non Maximum Suppression
NN	Neural Network
OOM	Out Of Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RMSprop	Root Mean Square Propagation
RoI	Region of Interest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
YOLO	You Only Look Once

1. Introduction

Deception is an act or a statement which misleads, hides the truth, or promotes a belief, concept, or idea that is not true. People lie all the time, the lies do not always have major consequences, but detecting lies is crucial in many areas that may affect the security of their surroundings, like police investigations and airport security.

1.1 Motivation

So far, fields such as the police and the airport security rely on human resources when conducting their investigations with people. Reports suggest that the ability of humans to detect deception without special aids is only 54% [1], and that's a very low percentage, it would be fruitful to have a Deception Detection system with better accuracy.

Deception detection is a difficult process for humans, it can take years to be able to tell if someone is deceiving you, and you probably won't always be able to tell. There also exist other factors that may affect the judgment of a person when given such a task. Humans may be subject to certain biases according to their beliefs or ideologies. Other factors from people trying to mislead the truth can be present in some situations such as bribing.

With the help of AI we can better detect deception with a well trained model and an accuracy exceeding the human level performance using facial micro expressions detected from pre-recorded videos.

1.2 Problem Definition

The consequences of falsely accusing the innocents and freeing the guilty can be severe. Hence, the need arises for a reliable and efficient system to detect deceptive behavior and discriminate between liars and truth tellers.

In criminal settings, the polygraph test has been used as a standard method to identify deceptive behavior. This becomes impractical in some cases, as this method requires the use of skin-contact devices and human expertise. In addition, the final decisions are subject to error and bias [2, 3]. Furthermore, using proper countermeasures, offenders can deceive these devices as well as the human

experts. Given the difficulties associated with the use of polygraph-like methods, learning-based approaches have been proposed to address the deception detection task using a number of modalities, including text [4] and speech [5, 6].

Facial expressions also play a critical role in the identification of deception. Ekman [7] defined micro-expressions as relatively short involuntary expressions, which can be indicative of deceptive behavior. Visual information shows how a person feels through their expressions, but other information "leaks" out of a person's face between or during these intentional expressions.

Microexpressions can be as brief as about 1/25 of a second. They occur so fast that they're often not perceived by the conscious mind of either the expresser or the person observing the expression. As few as 10 percent of people are even aware of seeing microexpressions when tested [8].

1.3 Objectives

- Train a machine learning model to correctly and accurately identify whether a person is lying or not.
- Minimize false accusations of the innocent.
- Replace existing lie detectors that have lower accuracy.
- Detecting micro expressions from videos.

1.3.1 What was done for achieving these objectives

- Trained a 3D CNN model on detecting deception.
- Trained a VGG-19 model on extracting micro-expressions.
- Constructed a pipeline combining both models for the final video verdict.

1.4 Timeplan

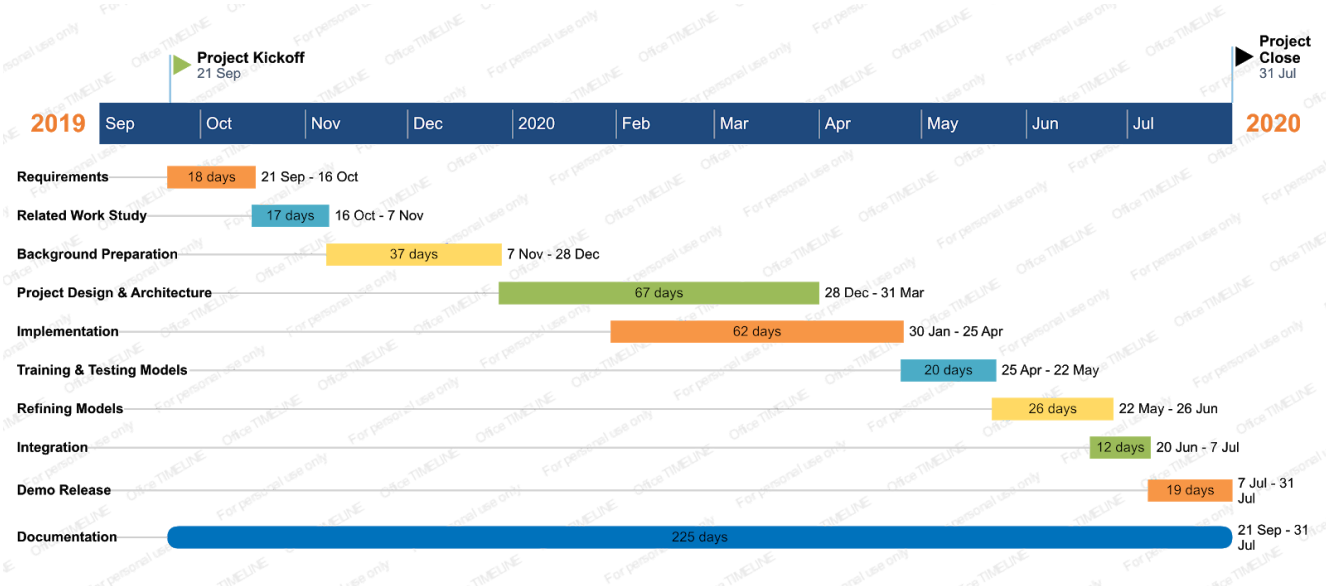

Figure 1.1: Time Plan

1. 18 days for understanding the project requirements.
2. 17 days for studying the literature published and related work.
3. 37 days for learning and understanding the project background.
4. 67 days for developing the models and project architecture.
5. 62 days for the implementation in parallel with the architecture development.
6. 20 days for the models' training and testing.
7. 26 days for refining and improving the models.
8. 12 days for integrating the models into a single pipeline.
9. 19 days for creating a demo release.
10. Working on the documentation in parallel with the whole work.

1.5 Document Organization

Chapter 2: Background
This chapter includes background about the project, basic concepts, and the related work according to our research.

Chapter 3: Analysis & Design
This chapter includes an overview of the whole system along with the intended user, system architecture, analysis and design.

Chapter 4: Dataset
This chapter includes an overview of the dataset used in the system.

Chapter 5: Implementation
This chapter describes the system functions with details about the implementation of the project's modules.

Chapter 6: Experiments
This chapter includes some of the experiments that we did through our work to improve the results.

Chapter 7: User Manual
This chapter talks about the needed packages and libraries that must be installed before using the application, and also shows the user how to use it.

Chapter 8: Conclusion & Future Work
This chapter includes the conclusion and results of our work and the future work that may be done based on this project.

References
This chapter includes all the papers and the sources that we used and studied from through our work.

2. Background

2.1 Neural Networks

Neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks and deep learning are big topics in Computer Science and in the technology industry, they currently provide the best solutions to many problems in image recognition, speech recognition and natural language processing.

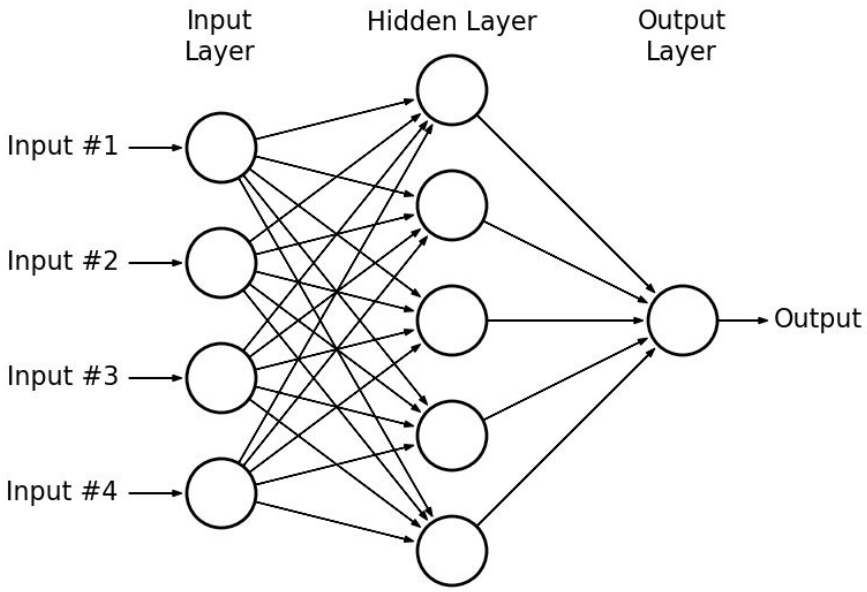

Figure 2.1: Neural Network general architecture

The basic unit of the human brain is the neuron, the essential building block of an artificial neural network is a perceptron which accomplishes simple signal processors and these are then connected into a large mesh network.

The computer with the neural network is taught to do a task by having it analyze training examples, which have been previously labelled in advance. A common example of a task for a neural network using deep learning is an object

recognition task, where the neural network is presented with a large number of objects of a certain type, such as a cat, or a street sign, and the computer, by analyzing the recurring patterns in the presented images, learns to categorize new images.

Unlike other algorithms, neural networks with their deep learning cannot be programmed directly for the task. Rather, they have the requirements, just like a child's developing brain, that they need to learn the information. The learning strategies go by three methods:

- **Supervised Learning:** This learning strategy is the simplest, as there is a labelled dataset, which the computer goes through, and the algorithm gets modified until it can process the dataset to get the desired result.
- **Unsupervised Learning:** This strategy is used in cases where there is no labelled dataset available to learn from. The neural network analyzes the dataset, and a cost function then tells the neural network how far off of the target it was. The neural network then adjusts to increase the accuracy of the algorithm.
- **Reinforced Learning:** in this algorithm, the neural network is reinforced for positive results, and punished for a negative result, forcing the neural network to learn over time.

2.1.1 Neural Network Architecture

The basic unit of computation in a neural network is the neuron, often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function to the weighted sum of its inputs.

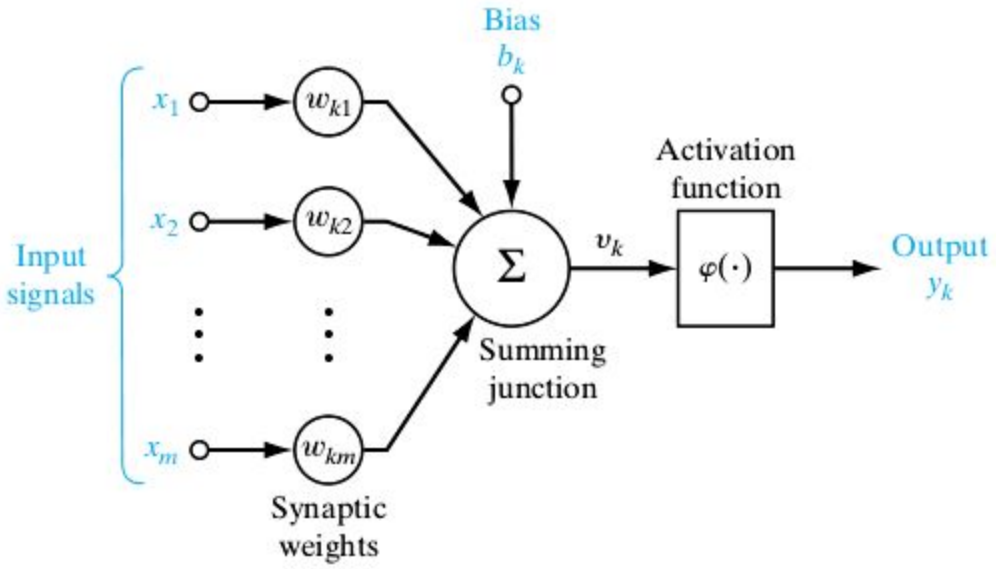

Figure 2.2: Single Neuron Architecture

Input Nodes (input layer): no computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called a layer.

Connections and weights: the network consists of connections, each connection transferring the output of a neuron i to the input of a neuron k . In this notation, i is the predecessor of k and k is the successor of i. Each connection is assigned a weight W_{ki}.

Output Nodes (output layer): here we finally use an activation function that maps to the desired output format (e.g. softmax for classification).

Activation function: it determines the output a node will generate, based upon its input. The activation function is set at the layer level and applies to all neurons in that layer. It may be linear or nonlinear. However, it is the nonlinear activation function that allows such networks to compute non-trivial problems using only a small number of nodes. It is also called the transfer function.

Hidden nodes (hidden layer): hidden layers are where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another

hidden layer or to the output layer). It is possible to have a neural network without a hidden layer.

Learning rule: the learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce the desired output. This learning process typically amounts to modifying the weights and thresholds.

2.1.2 Types of Artificial Neural Networks

2.1.2.1 Feed Forward Neural Networks

Feedforward neural networks are artificial neural networks where the connections between units do not form a cycle. They were the first type of artificial neural network invented and are simpler than their counterpart, recurrent neural networks. They are called *feedforward* because information only travels forward in the network, and there are no feedback (loops); *i.e.*, the output of any layer does not affect that same layer. The propagation traverses first through the input nodes, then through the hidden nodes (if present), and finally through the output nodes.

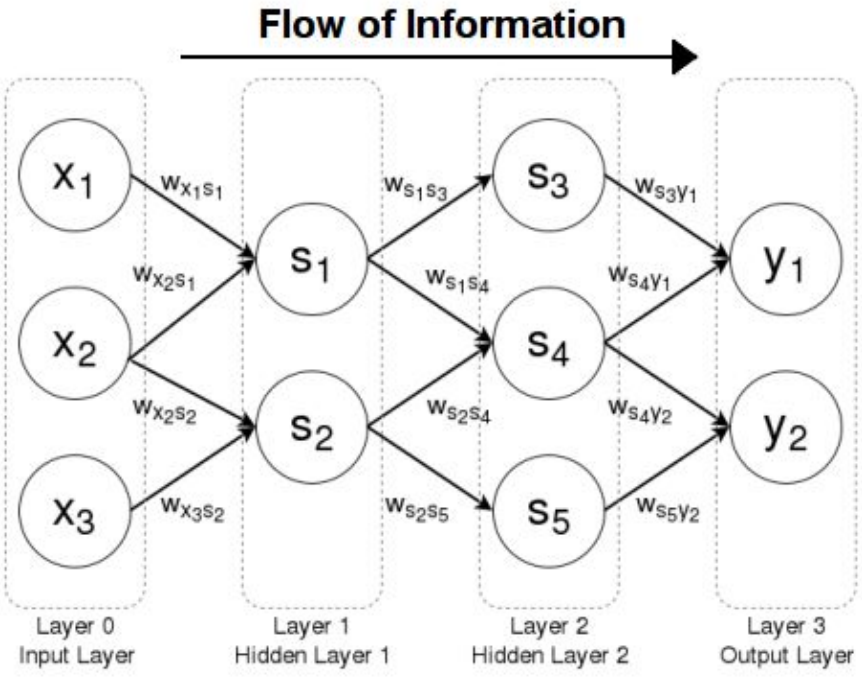

Figure 2.3: Feed Forward Neural Network

Feedforward neural networks are primarily used for supervised learning in cases where the data to be learned is neither sequential nor time-dependent. They are extensively used in pattern recognition.

2.1.2.1.1 Single-Layer Perceptron

The simplest type of feedforward neural network is the perceptron, a feedforward neural network with no hidden units. Thus, a perceptron has only an input layer and an output layer. The output units are computed directly from the sum of the product of their weights with the corresponding input units, plus some bias.

Historically, the perceptron's output has been binary, meaning it outputs a value of 0 or 1. This is achieved by passing the aforementioned product sum into the step function ϕ (x). This is defined as:

$$\phi(x) = \begin{cases} 1 & ,if\ x \geq 0 \\ 0 & ,if\ x\ < 0 \end{cases}$$

Since the perceptron divides the input space into two classes, 0 and 1 depending on the values of weight vector and bias, it is known as a Linear classifier. The line separating the two classes is known as classification boundary. In the case of two-dimensional input (as in the previously referred figure) it is a line, while in higher dimensions this boundary is a hyperplane. The weight vector defines the slope of the classification boundary while the bias defines the intercept.

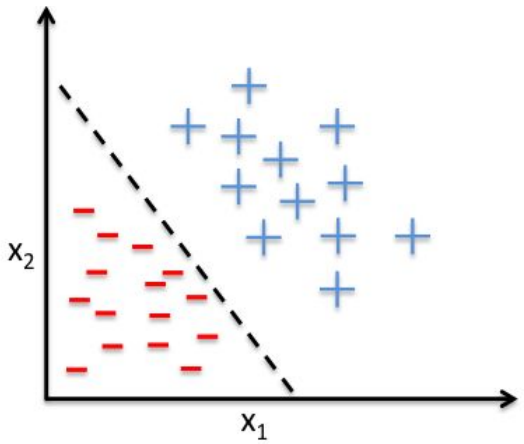

Figure 2.4: Linear Classifier

It was mentioned earlier that single-layer perceptrons are linear classifiers. That is, they can only learn linearly separable patterns. Since many functions aren't linearly separable they found an alternative which is multi-layer perceptrons.

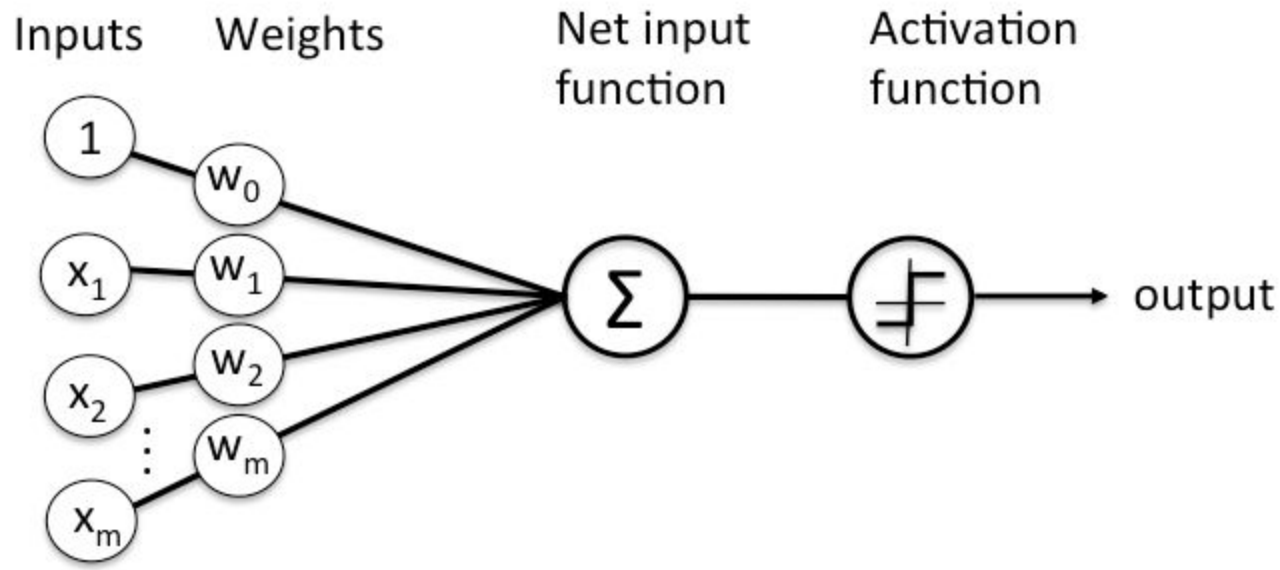

Figure 2.5: Single Perceptron

2.1.2.1.2 Multi-Layer Perceptron (MLP)

It is an artificial neural network composed of many perceptrons. Unlike single-layer perceptrons, MLPs are capable of learning to compute non-linearly separable functions. Because they can learn non-linear functions, they are one of the primary machine learning techniques for both regression and classification in supervised learning (most of the cases the data presented to us is not linearly separable).

These neural networks consist of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications, the units of these networks apply a sigmoid function as an activation function.

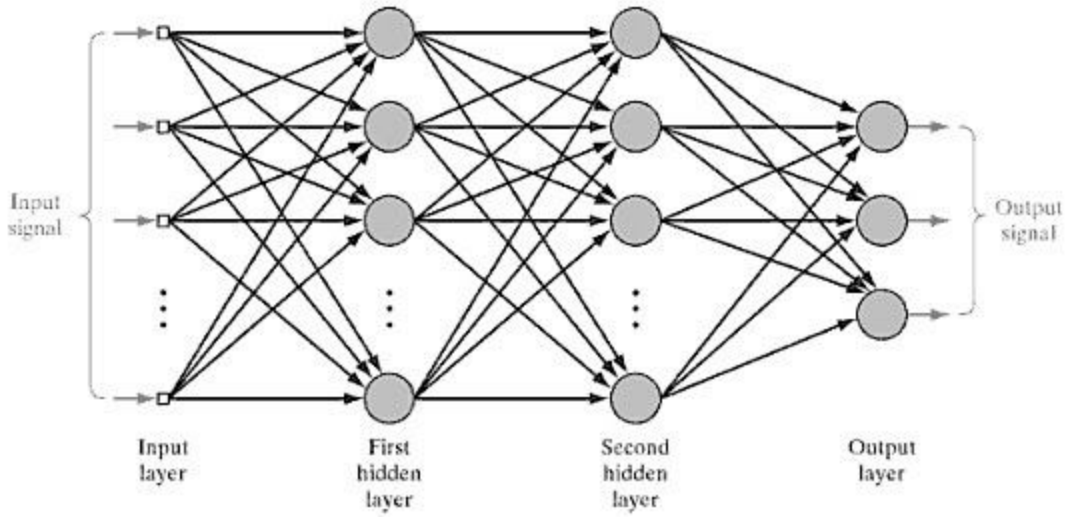

Figure 2.6: Multi-Layer Perceptron

2.1.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A Recurrent Neural Network remembers the past and its decisions are influenced by what it has learnt from the past. Note: Basic feedforward networks "remember" things too, but they remember things they learnt during training. For example, an image classifier learns what a "1" looks like during training and then uses that knowledge to classify things in production. While RNNs learn similarly while training, in addition, they remember things learned from prior input(s) while generating output(s).

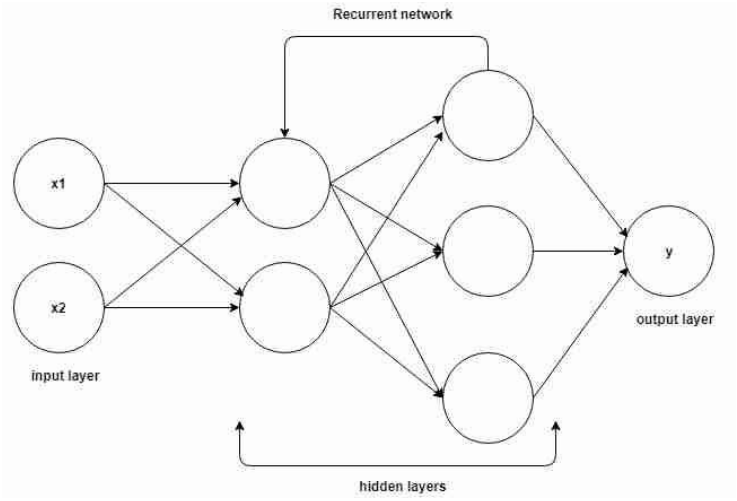

Figure 2.7: Recurrent Neural Network

2.1.3 Backpropagation

It is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization. Backpropagation has 3 main steps:

- **Forward-propagation**
  1- Getting the weighted sum of inputs of a particular unit
  2- Plugging the value we get from step 1 into the activation function and using the activation value we get (i.e. the output of the activation function) as the input feature for the connected nodes in the next layer.
  For hidden units:

$$net_j(t) = \sum_{i=0}^{n} w_{ji}(t)\, x_i(t) \equiv w_j^T . X \qquad (2.1)$$

n: the number of output units
w(j)(i): the weight of the connection from neuron i to neuron j
x(i): neuron i of the input layer

  For output units:

$$net_k(t) = \sum_{j=0}^{d} w_{kj}(t)\, z_i(t) \equiv w_k^T . Z \qquad (2.2)$$

w(k)(j): the weight of the connection from neuron j to neuron k
z(j): neuron j of the hidden layer

- **Backpropagation**
  We compute 'error signal', propagating the error backwards through the network starting at output units (where the error is the difference between actual and desired output values).

- **Update weights**
  1- The weights are updated using the following rules

Hidden-to-Output units:

$$\Delta w_{kj}(t) = -\eta \frac{\partial E}{\partial w_{kj}} = \eta(d_k - y_k) f'(net_k) z_j$$

η: learning rate
d(k): desired output
y(k): actual output
f'(net): derivative of the activation function
z(j): neuron j of the hidden layer

Input/Hidden-to-Hidden units:

$$\Delta w_{ji}(t) = \eta \delta_j(t) x_i(t) = \eta f'(net_j) \left[\sum_{k=0}^{m} \delta_k w_{kj} \right] x_i$$

η: learning rate
δ: local gradient of neuron
f'(net): derivative of the activation function
x(i): neuron i of the input layer

2.2 Deep Neural Networks

A deep neural network is an artificial neural network with multiple hidden layers between the input and output layers, in which each hidden layer output is the input for the next hidden layer except for the last hidden layer its output flows to the output layer.

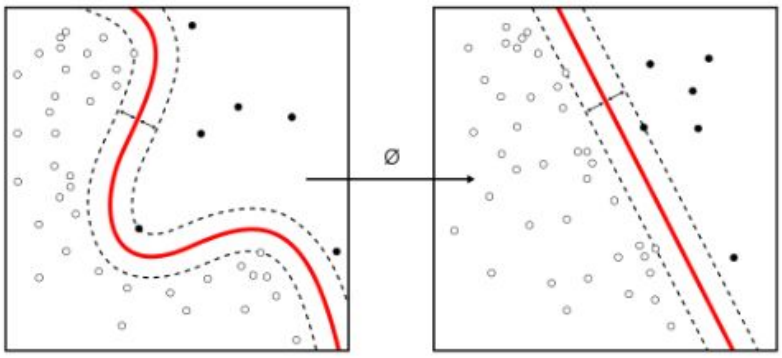

Figure 2.8: Non-linear separable data to linear separable data

This architecture makes data more sperable-like, so non-linear separable data is going to be linearly separable due to applying certain nonlinear functions.

Architecture of Deep Neural Networks helps also at extracting features of images and data points such as Object and Face Recognition tasks, it adapts its weights depending on how effective this pixel (or value) is on desired output.

We use Deep Neural Networks in our work in both modules: Visual Features extraction and Micro-expressions extraction.

2.3 Convolutional Neural Networks (CNNs/ConvNets)

Convolutional Neural Networks (CNNs) are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks developed for learning regular Neural Networks still apply.

The only difference is that ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

CNNs take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. Unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. Note that the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.

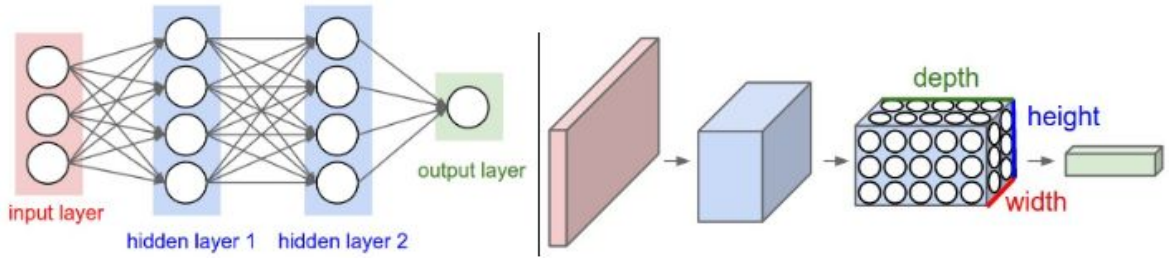

Figure 2.9: Comparison between MLP and CNN

There are three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer**. These layers stacked form a ConvNet **architecture**.

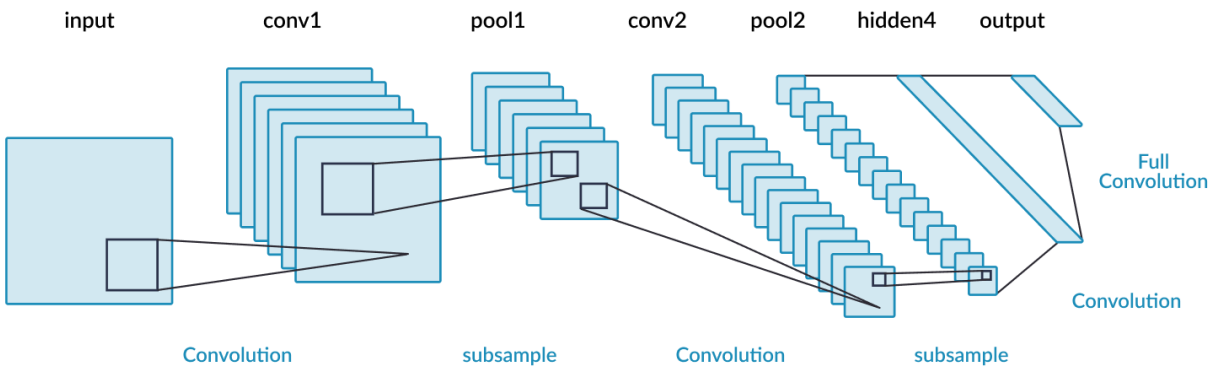

Figure 2.10: CNN Architecture

2.3.1 Convolution Layer

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The first layer in a CNN is always a **Convolutional Layer**. The input is a 3-dimensional volume (width, height, depth).

It contains a **filter**(kernel) which is an array of numbers (weights) covering a small area of input volume (called respective field), and it **convolves** the input volume. As the filter is convolving, it is multiplying the values in the filter with the original values of the respective field. So every unique location on the input volume produces a number.

Each filter convolution results in an **activation map** or **feature map**. The depth of this filter has to be the same as the depth of the input volume. By using more filters we get more feature maps, hence the depth of the output volume will be greater than one.

The aim of conv layers is to detect features in images from low level features (like straight edges, simple colors, and curves) to higher level features (such as hands, paws or ears) by going deeper through the network.

There are 2 main parameters that we can change to modify the behavior of each conv layer. After we choose the filter size, we also have to choose the **stride** and the **padding** values**.**

2.3.2 Strides

Controls how the filter convolves around the input volume. The amount by which the filter shifts is the stride (shifts in the width and height dimensions).

2.3.3 Padding

Used to get the width and height of the output volume to be the same as the input volume, by padding the input volume with zeros. As we keep applying conv layers, the size of the volume will decrease faster than we would like, padding is used to solve that issue. Also for preserving as much information about the original input volume in the early layers of our network as possible, so that we can extract those low level features.

2.3.4 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computations in the network, and hence to also control overfitting.

Spatial pooling can be of different types such as Max Pooling, Average Pooling or Sum Pooling.

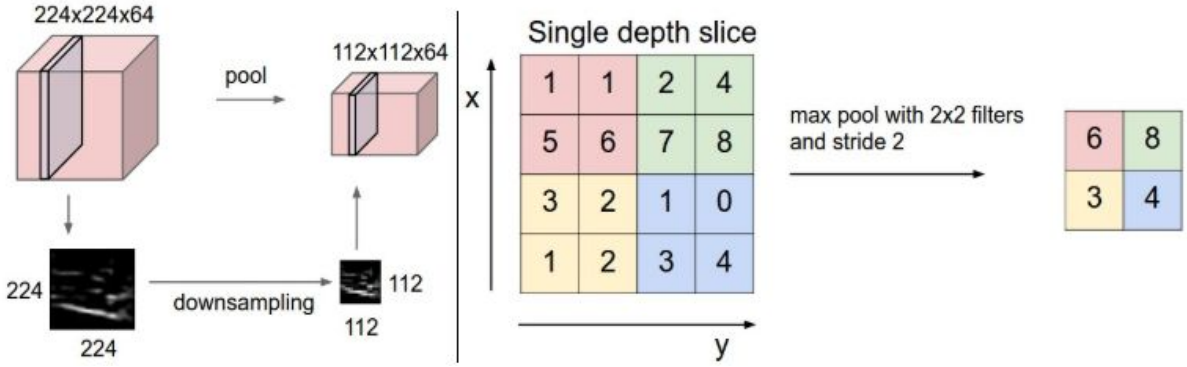

Figure 2.11: Max Pooling Layer

2.3.5 Activation Functions

The activation function is a node that is put at the end of or in between network layers, to help decide if the neuron would fire or not. The activation function adds a non-linear transformation over the output signal of the layer. This transformed output is then sent to the next layer of neurons as input.

2.3.5.1 Hyperbolic Tangent function (Tanh)

A better version of the logistic sigmoid with range from -1 to 1, used in feedforward networks. The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

The function is differentiable and monotonic while its derivative is not monotonic. And is mainly used for classification between two classes.

Figure 2.12: Hyperbolic Tangent function (Tanh) Graph

2.3.5.2 Non-Linearity (ReLU)

Rectified Linear Unit (ReLU) function is the most widely used activation function in neural networks today. One of the greatest advantages it has over other activation functions is that it does not activate all neurons at the same time.

From the image for ReLU function below, we'll notice that it converts all negative inputs to zero and the neuron does not get activated. This makes it very computationally efficient as fewer neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

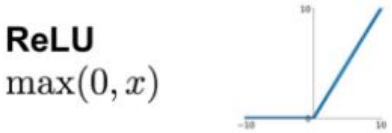

Figure 2.13: Rectified Linear Unit

2.3.6 Fully Connected Layer

The layer we call as FC layer, we flatten our matrix into a vector and feed it into a fully connected layer like a neural network.

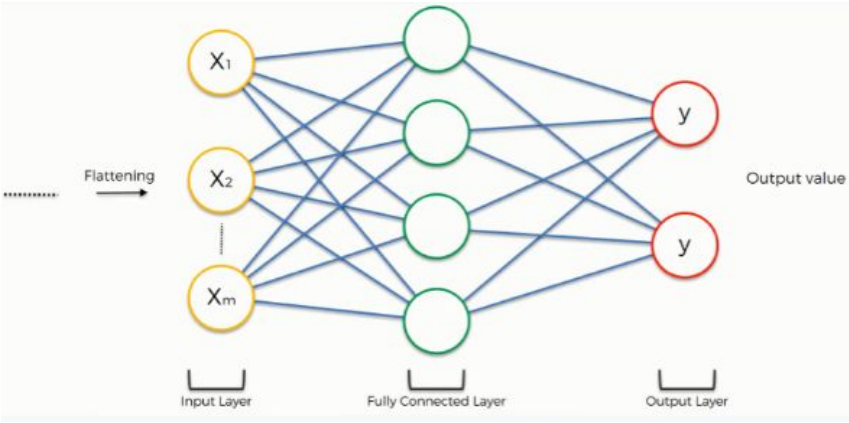

Figure 2.14: Fully Connected Layer

2.3.7 Dropout Layer

Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units to 0 at each update of the training phase. The term "dropout" refers to dropping out units in a NN.

It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently.

By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connection. The choice of which units to drop is random.It is subject to a predetermined dropout probability.

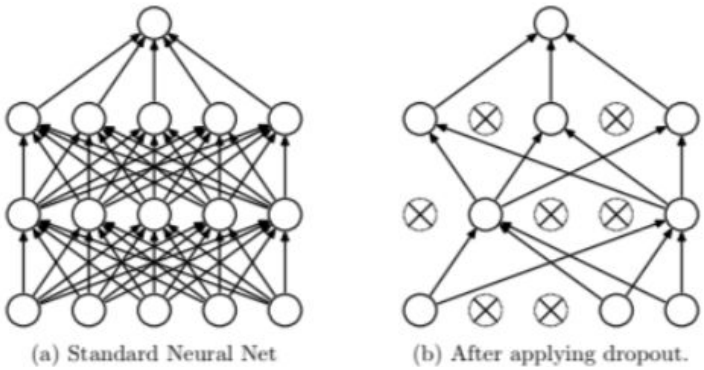

Figure 2.15: Dropout Layer

2.3.8 3D Convolutions

3D convolutions apply a 3 dimensional filter to the dataset and the filter moves in the 3-directions (x, y, z) to calculate the low level feature representations. Their output shape is a 3 dimensional volume space such as a cube or a cuboid. They are helpful in event detection in videos, 3D image data such as Magnetic Resonance Imaging (MRI) data etc. They are not limited to 3D space inputs but can also be applied to 2D space inputs such as images.

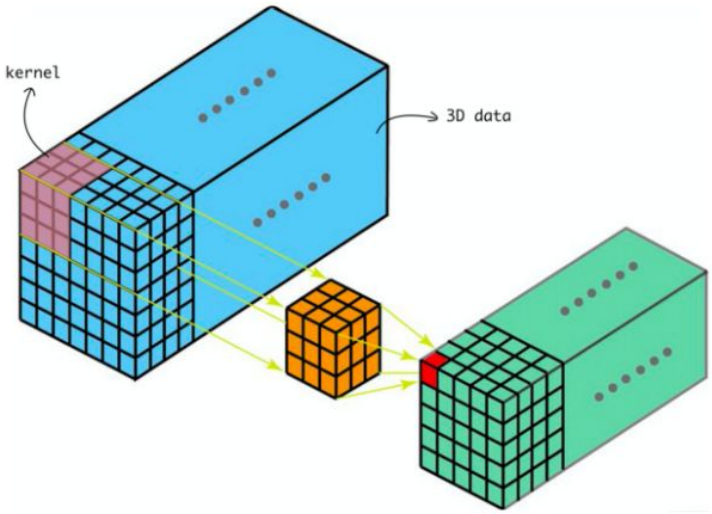

Figure 2.16: 3D Convolution

2.4 Optimizer & Loss functions

A loss function measures the quality of a particular set of parameters based on how well the induced scores agree with the ground truth labels in the training data. There are many ways and versions of this (e.g. Softmax/SVM).

Neural network models learn a mapping from inputs to outputs from examples & the choice of loss function must match the framing of the specific predictive modeling problem, such as classification or regression. Further, the configuration of the output layer must also be appropriate for the chosen loss function.

Optimization is the process of finding the set of parameters W that minimizes the loss function.

Loss functions and Optimizers are used to help better train a module on the training data by fine tuning the training parameters of weights used.

2.4.1 Gradient Descent

Gradient descent is an optimization algorithm which is dependent on the first order derivative of a loss function. It calculates which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred through layers and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

The weights are changed after calculating the gradient on the whole dataset. So, if the dataset is too large then this may take years to converge to the minima.

2.4.2 Stochastic Gradient Descent (SGD)

It's an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization. Samples are selected randomly (or shuffled) instead of as a single batch or in the order of appearance.

$$W^{(k+1)} = W^{(k)} - \eta * (\Delta J(W))$$

SGD reduces the computational expense of performing the typical Gradient Descent on datasets with a large number of samples.

2.4.3 Mini-Batch Gradient Descent

It is an improvement on both SGD and standard gradient descent. It updates the model parameters after every mini-batch. So, the dataset is divided into various batches and after every batch, the parameters are updated.

Please note the difference in the convergence between the **Stochastic** and the **Batch Gradient Descent** for better explanation of the smoother convergence of the **Batch Gradient Descent**.

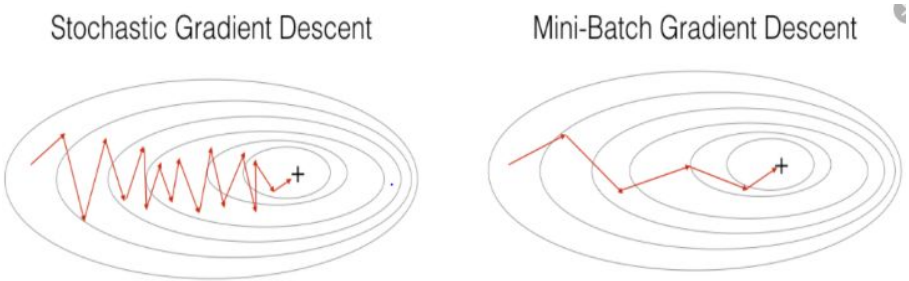

Figure 2.17: Comparison between SGD and Mini-Batch GD

2.4.4 Momentum

Momentum was invented for reducing high variance in SGD and softens the convergence. It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction. One more hyperparameter is used in this method known as momentum term symbolized by 'γ'.

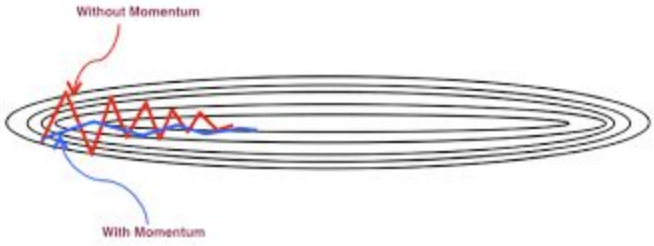

Figure 2.18: Effect of adding Momentum-term

Gradient Descent with momentum considers the past gradients to smooth out the update. It computes an exponentially weighted average of your gradients, and then uses that gradient to update your weights instead. It works faster than the standard gradient descent algorithm.

2.4.5 RMSprop Optimizer

Gradients of very complex functions like neural networks have a tendency to either vanish or explode as the data propagates through the function. **RMSprop** or Root Mean Square Propagation, was developed as a stochastic technique for mini-batch learning. It deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.

Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. This means that the learning rate changes over time.

2.4.6 ADAM Optimizer

Adaptive Moment Optimization (ADAM) algorithm combines the heuristics of both Momentum and RMSProp. We compute the exponential average of the

gradient as well as the squares of the gradient for each parameter. To decide our learning step, we multiply our learning rate by average of the gradient (as was the case with momentum) and divide it by the root mean square of the exponential average of the square of gradients (as was the case with momentum). Then, we add the update.

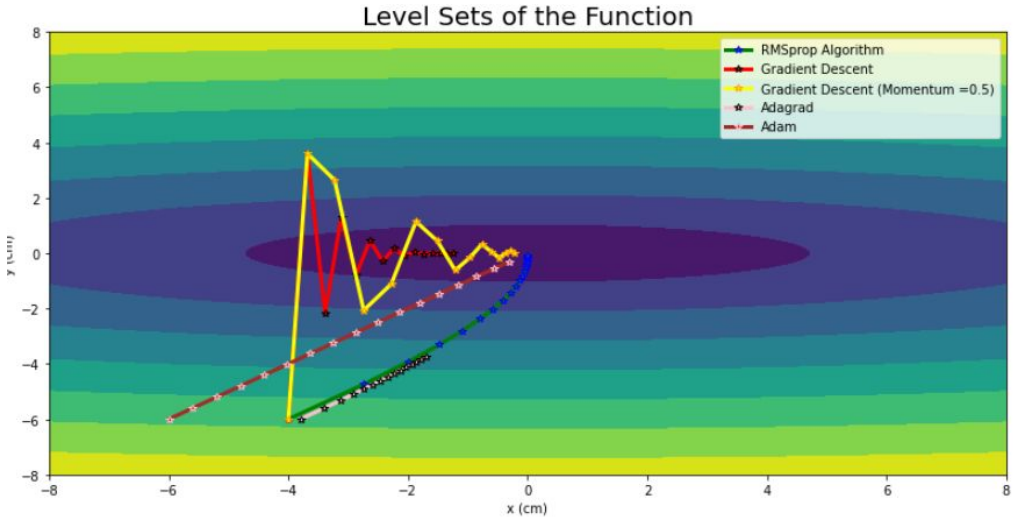

Figure 2.19: Convergence of Different Optimizers

2.5 Object Recognition

Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs.

2.5.1 Object Recognition tasks

- **Image Classification:** Predict the type or class of an object in an image.
    - *Input:* An image with a single object, such as a photograph.
    - *Output:* A class label (e.g. one or more integers that are mapped to class labels).
- **Object Localization:** Locate the presence of objects in an image and indicate their location with a bounding box.
    - *Input:* An image with one or more objects, such as a photograph.
    - *Output:* One or more bounding boxes (e.g. defined by a point, width, and height).

- **Object Detection:** Locate the presence of objects with a bounding box and types or classes of the located objects in an image.
    - *Input:* An image with one or more objects, such as a photograph.
    - *Output*: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.
- **Object Segmentation:** Instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a bounding box.

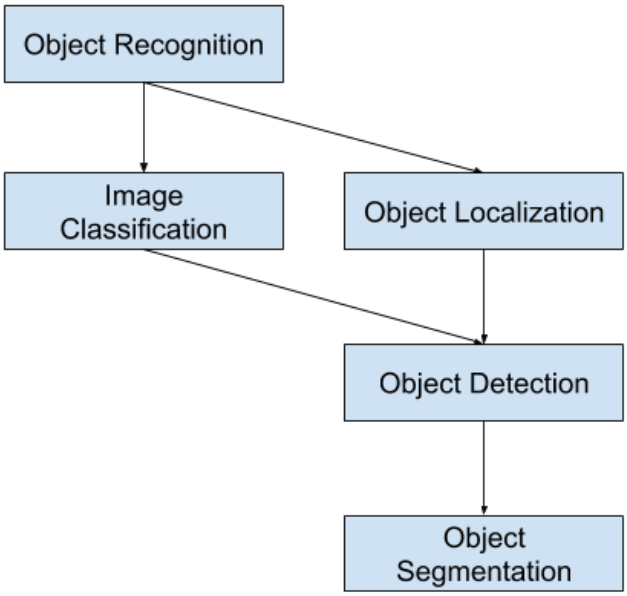

Figure 2.20: Overview of Object Recognition Computer Vision Tasks

2.5.2 Image Classification (VGG)

VGG is a convolutional neural network model [9]. This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier.

The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

2.5.2.1 Architecture

During training, the input to the ConvNets was a fixed-size 224 × 224 RGB image. The only preprocessing was subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where 3 × 3 filters were used. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2. A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity.

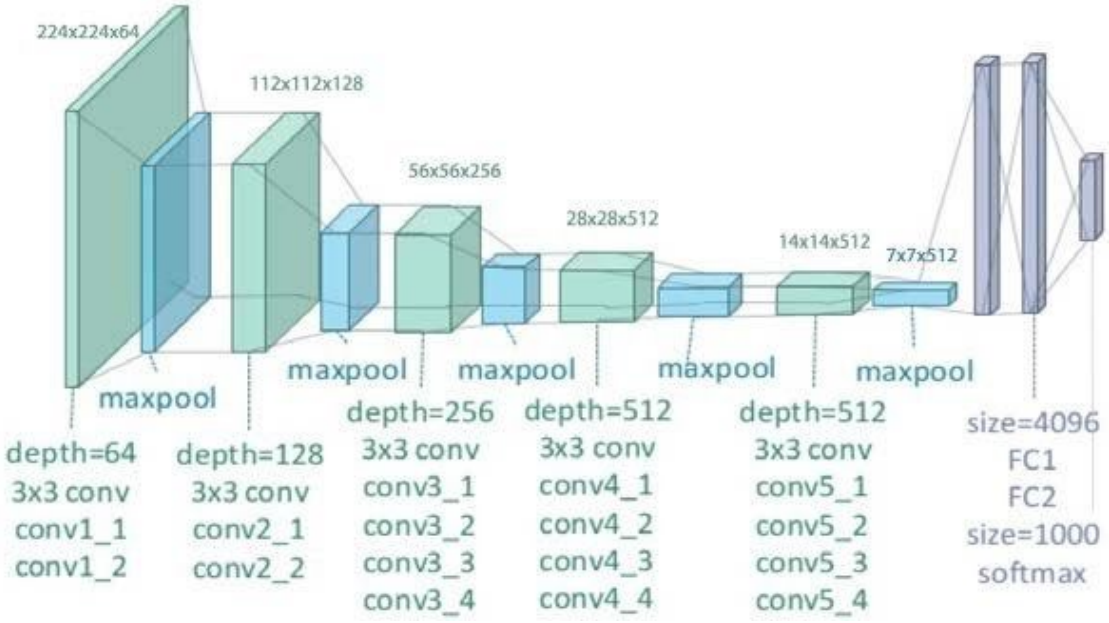

Figure 2.21: VGG-19 Architecture

2.5.3 Object Detection

2.5.3.1 Sliding Window Approach

A naive approach would be to take different regions of interest from the image, and use a **CNN** to classify the presence of an object within that region. In order for this to be correct, we would need to take a large number of regions by sliding a window across the image, we might also need to use more than one window-size because not all objects have the same dimensions nor aspect-ratios. Of course, this is computationally expensive and it's an impossible approach for real-time detection. That's why more optimized approaches like R-CNN and YOLO have appeared.

2.5.3.2: R-CNN [10]

To bypass the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use selective search to extract just 2000 regions from the image and he called them region proposals. Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions.

The R-CNN was described in the 2014 paper by Ross Girshick, et al. from UC Berkeley titled "Rich feature hierarchies for accurate object detection and semantic segmentation."

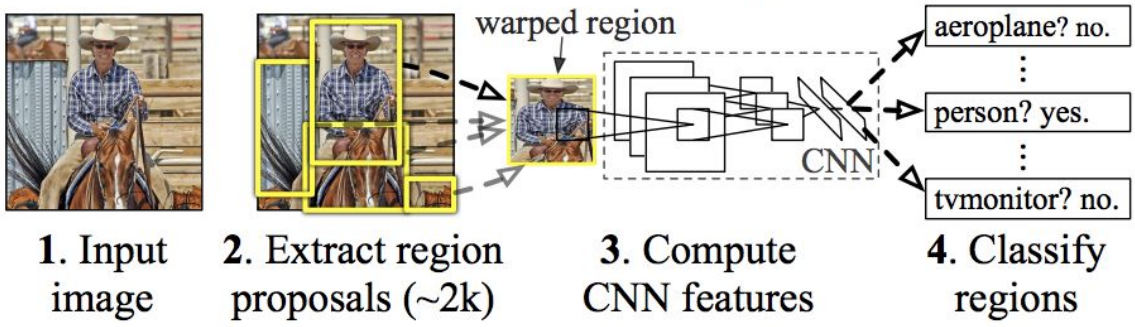

Figure 2.22: R-CNN: Regions with CNN features

2.5.3.3 Fast R-CNN [11]

The same author of the previous paper (R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm.

A Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map. Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

The reason "Fast R-CNN" is faster than R-CNN is because you don't have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.

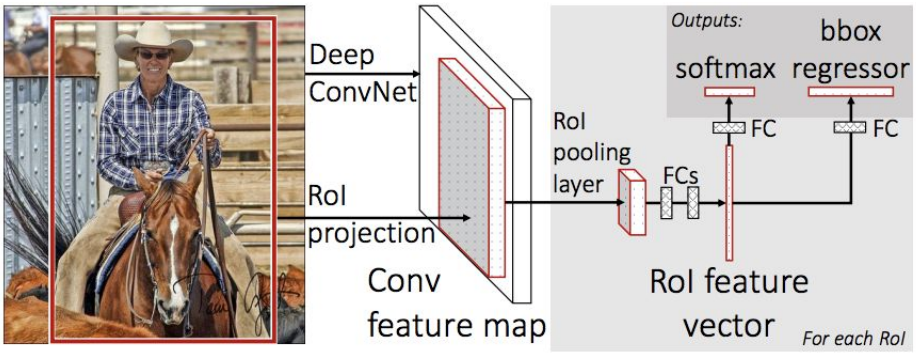

Figure 2.23: Fast R-CNN

2.5.3.4 Faster R-CNN [12]

Both of the above algorithms (R-CNN & Fast R-CNN) use selective search to find the region proposals. Selective search is a slow and time-consuming process which affects the performance of the network.

Therefore, Shaoqing Ren et al. came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals. Faster R-CNN is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions.

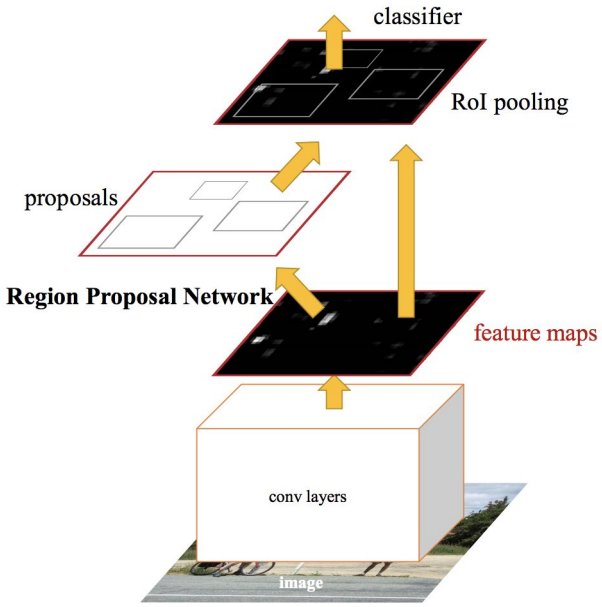

Figure 2.24: Faster R-CNN

2.5.3.5 You Only Look Once (YOLO) [13]

All the above algorithms are based on the concept of finding regions that have a high probability of containing an object and then classifying those regions, they don't look at the image as a whole unit. YOLO, on the other hand, does just that.

YOLO, which is an open-source project, reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.

Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are. YOLO is refreshingly simple: see **Figure 2.25**. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

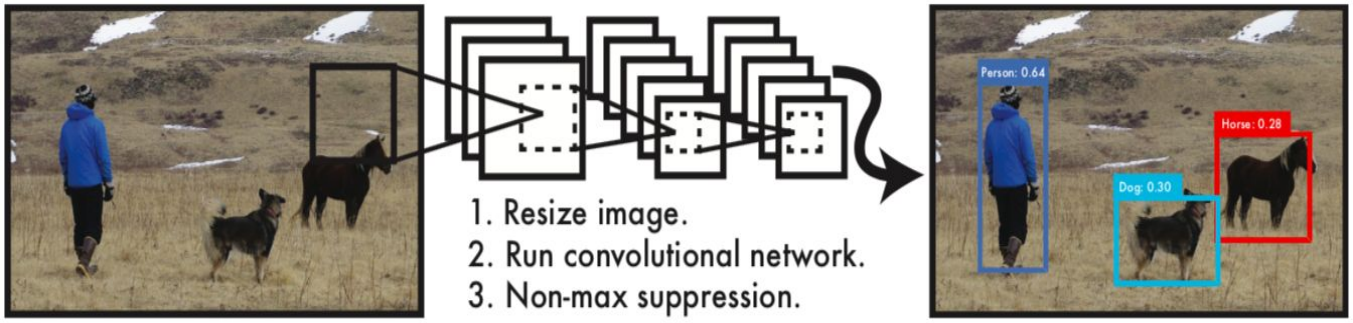

Figure 2.25. The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence

This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, mistakes background patches in an image for objects because it can't see the larger context.

YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

YOLO's Unified Detection:

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. Our system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $Pr(Object) * IOU_{pred}^{truth}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

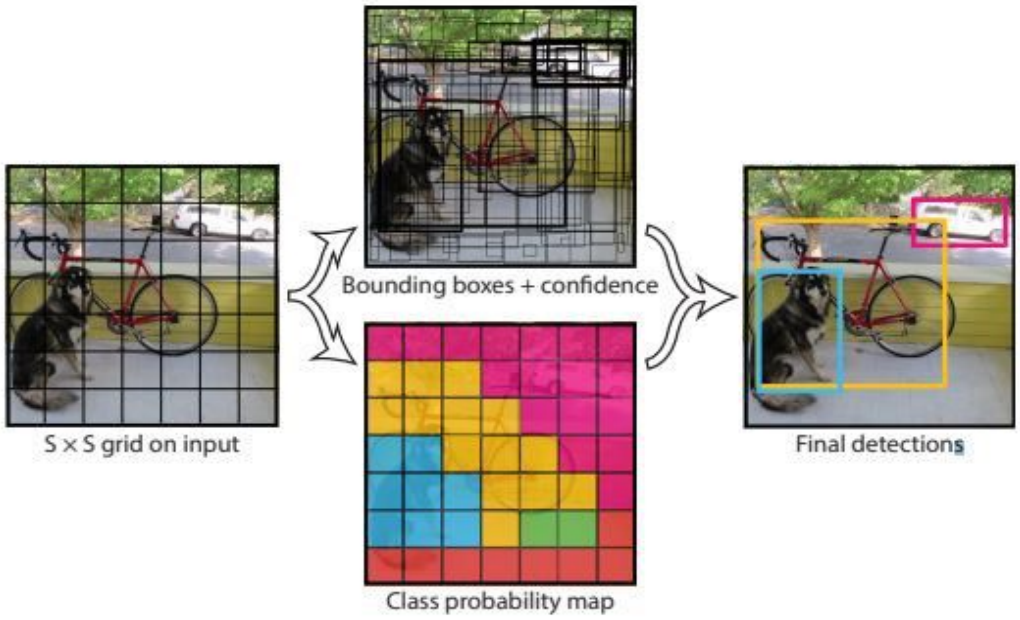

Figure 2.26: Yolo's Model. Our system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B ∗ 5 + C) tensor.

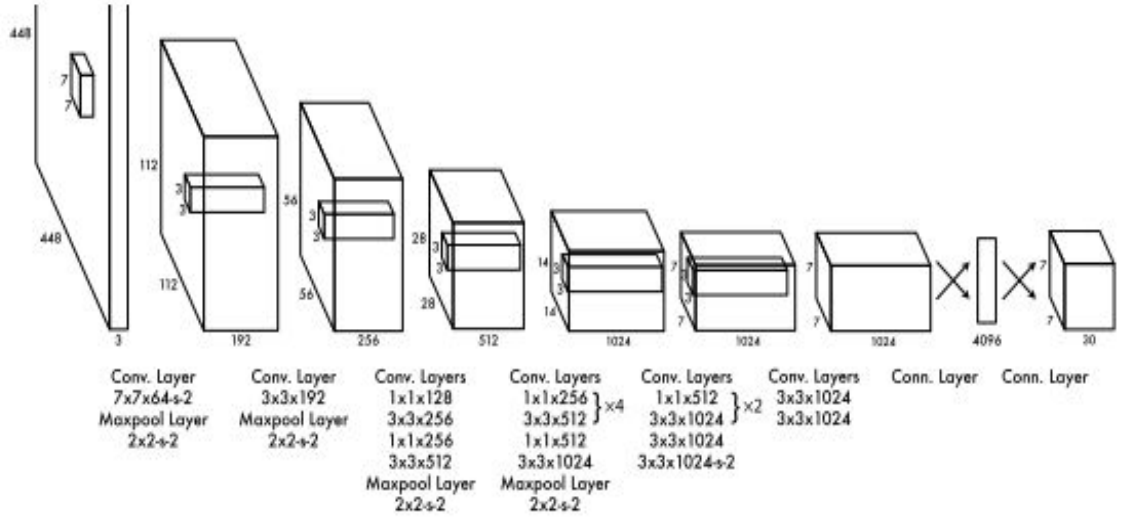

Figure 2.27: Yolo's Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection.

YOLO's Non-Maximum Suppression:
How does YOLO deal with redundant and/or inaccurate bounding boxes? It uses Non-Maximum Suppression (NMS).

First, it sets all class scores that are less than a certain threshold (0.6) to zero.

Then, for each class, it sorts the scores descendingly. Each bounding box is then compared with the first bounding box (the one with the highest class score), if the IOU (intersection over union) of the two bounding boxes is bigger than, say, 0.5, that bounding box is discarded by setting its confidence to zero.

That way, YOLO selects the strongest bounding box per class.

2.6 Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.

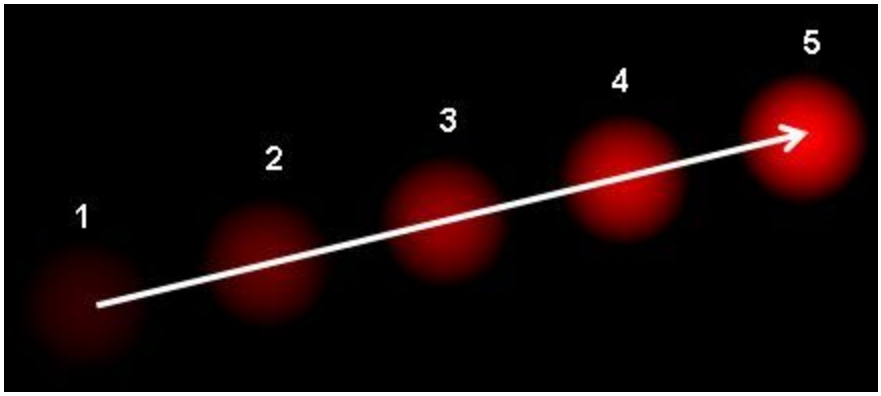

Figure 2.28: Optical Flow

Figure 2.28 shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector. Optical flow has many applications in areas like :

- Structure from Motion.
- Video Compression.
- Video Stabilization.

Optical flow works on several assumptions:
1- The pixel intensities of an object do not change between consecutive frames.
2- Neighbouring pixels have similar motion.

2.7 Micro-Expressions

Micro-expressions are facial expressions that occur within a fraction of a second. This involuntary emotional leakage exposes a person's true emotions. Ekman [14] dened micro-expressions as short involuntary expressions, which could potentially indicate deceptive behavior. At 1/25th of a second, micro-expressions can be difficult to recognize. Yet with training you may learn to spot them as they occur in real-time.

2.8 Related Work

2.8.1 Deep Learning Approach for Multimodal Deception Detection [15]

Automatic deception detection is an important task that has gained momentum in computational linguistics due to its potential applications. In this paper, they proposed a simple yet tough to beat multi-modal neural model for deception detection. By combining features from different modalities such as video, audio, and text along with Micro-Expression features, they showed that detecting deception in real life videos can be more accurate. Their models were trained on the real-life courtroom dataset.

2.8.2 Deception Detection in Videos [16]

They presented a system for covert automated deception detection using information available in a video. They studied the importance of different modalities like vision, audio and text for this task. On the vision side, their system used classifiers trained on low level video features which predict human microexpressions. They showed that predictions of high-level microexpressions can be used as features for deception prediction. Even though state-of-the-art methods use human annotations of micro-expressions for deception detection, their fully automated approach outperforms them. They also presented results of a user-study to analyze how well do average humans perform on this task, what modalities they use for deception detection and how they perform if only one modality is accessible.

2.8.3 Facial Action Units [17]

This paper a method for detecting deception in RGB videos is presented. The method automatically extracts facial Action Units (AU) from video frames containing the interviewed subject, and classifies them through an SVM as truthful or deception. Experiments on real trial court data and comparisons with the current state of the art show the effectiveness of the proposed method.

2.8.4 Face-Focused Cross-Stream Network [18]

Automated deception detection (ADD) from real-life videos is a challenging task. For face-body multimodal learning, a novel face-focused cross-stream network (FFCSN) is proposed. It differs significantly from the popular two-stream networks in that: (a) face detection is added into the spatial stream to capture the facial expressions explicitly, and (b) correlation learning is performed across the spatial and temporal streams for joint deep feature learning across both face and body. To address the training data scarcity problem, their FFCSN model is trained with both meta learning and adversarial learning. Extensive experiments show that our FFCSN model achieves state-of-the-art results.

2.8.5 Design & Development of a Lie Detection System [19]

Their technique to detect lies is through the identification of facial micro-expressions, which are brief, involuntary expressions. It is an automated vision system designed and implemented using LabVIEW. An Embedded Vision System (EVS) is used to capture the subject's interview. Then, a LabVIEW program converts the video into a series of frames and processes the frames, each at a time, in four consecutive stages. The first two stages deal with color conversion and filtering. The third stage applies geometric-based dynamic templates on each frame to specify key features of the facial structure. The fourth stage extracts the needed measurements in order to detect facial micro-expressions to determine whether the subject is lying or not.

3. Analysis and Design

3.1 System Overview

We introduce the entire architecture of our system. The main module on which our system is based is called “3D-CNN” which extracts visual features from the region specific videos.

Another module is “Micro-expression Detectors” which extracts the micro-expressions directly from the videos.

The output of these 2 modules are concatenated and entered to a softmax layer which performs the classification to **Deceptive** or **Truthful.**

3.1.1 System Architecture

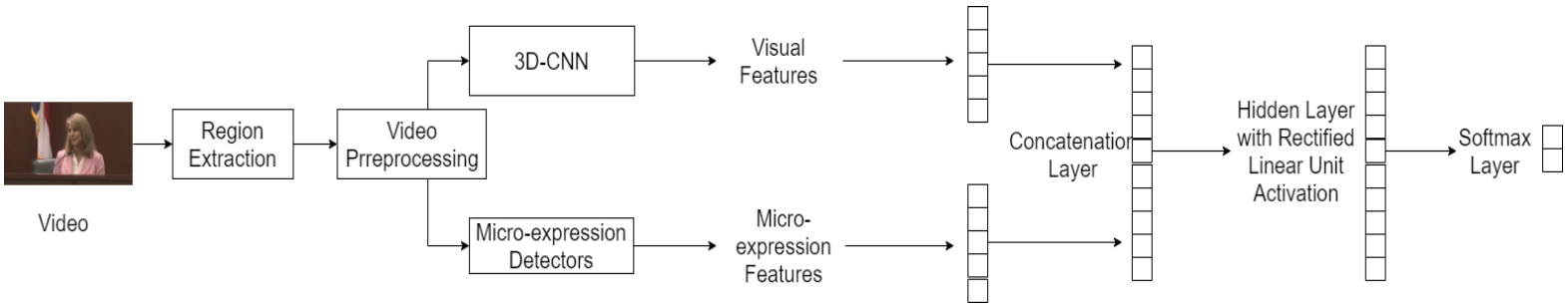

Figure 3.1: System Architecture

First, the input is a whole video, it gets passed to the region extraction module in order to eliminate unnecessary background objects. Then the cropped video goes to the video preprocessing module where the video gets prepared for each of the succeeding feature modules. In the end, the features are concatenated and predictions are made on the concatenated features.

3.1.2 System Users

- Intended Users:
  The system is built for the public: the user just uploads a video to the system to detect if the subject in it is **deceptive** or **truthful**.
- User Characteristics
  - Familiar with computer/website applications.

3.2 System Analysis & Design

3.2.1 Use Case Diagram

3.2.1.1 Diagram

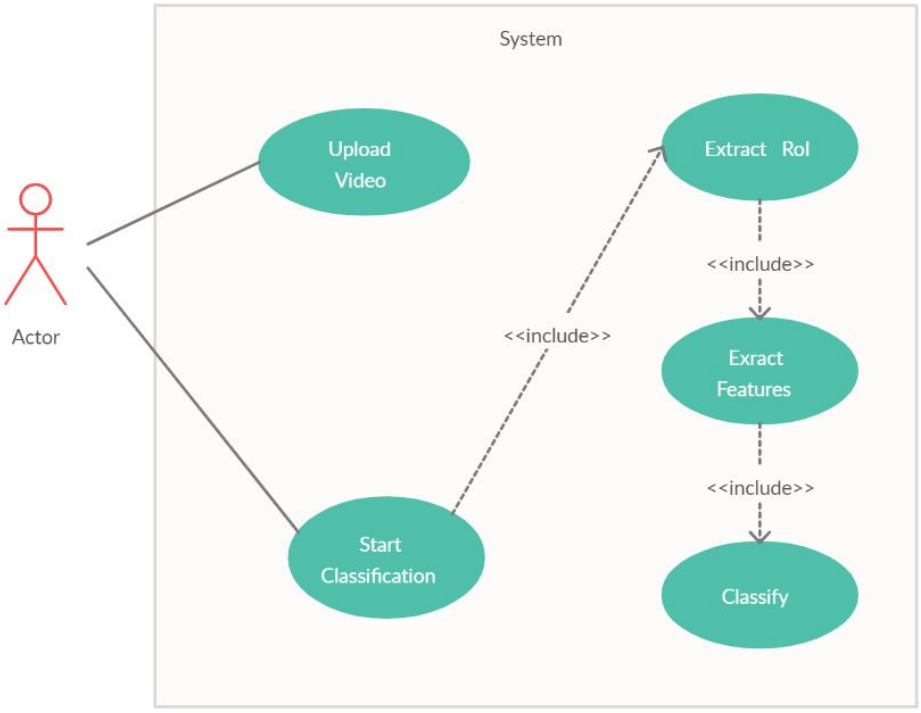

Figure 3.2: Use Case Diagram

3.2.1.2 Description of Use Cases

1. Upload Video
   Description: Takes a video as an input parameter.
   User : Actor.
2. Start Classification
   Description: Sends a classification request to the backend module.
   User : System.
   Expected Output: a classification verdict.
3. Extract RoI
   Description: Extracts the region of interest in the input video.

User : System.

Expected Output: A new video of the region of interest in the frames.

4. Extract Features

   Description: Extracts the visual features in the video.

   User : System.

   Expected Output: A vector of the extracted visual features.

5. Classify

   Description: Classifies the resulting features into one of the 2 classes, Deceptive or Truthful.

   User : System.

   Expected Output: A classification verdict of Deceptive or Truthful.

3.2.2 Flow of Events

- The user uploads a video to the website.
- Requests for a classification.
- The website fires the module with the uploaded video.
- The module processes the input video and outputs a classification.
- The classification output is sent back to the website.
- The website displays the output to the user.

3.2.3 Sequence Diagram

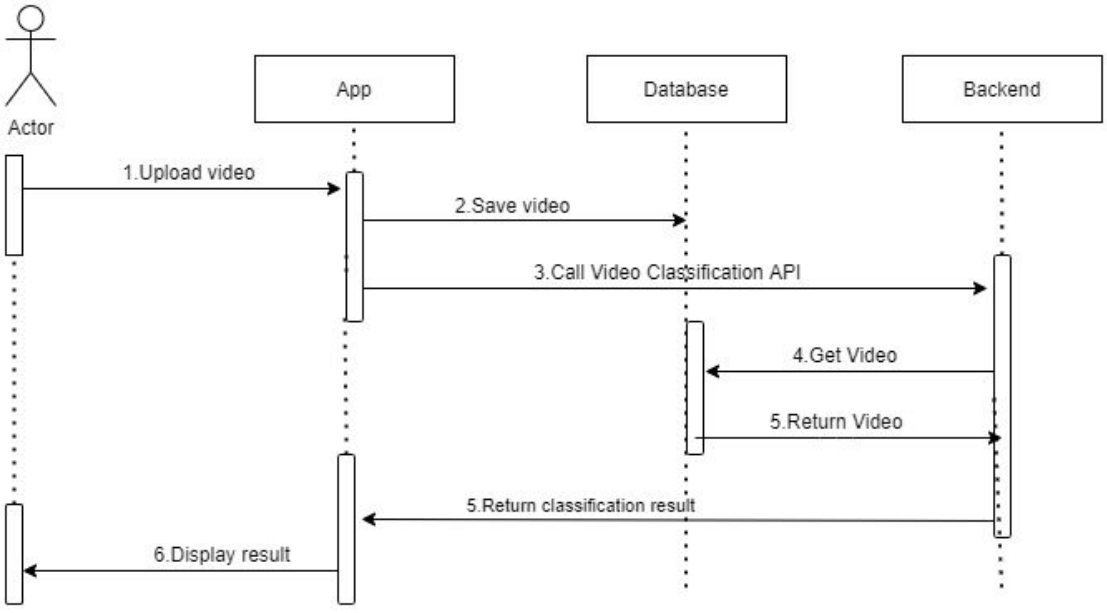

Figure 3.3: Sequence Diagram

4. Dataset

For evaluating our deception detection model, we use a real-life deception detection dataset by [20]. This dataset contains 121 video clips of courtroom trials. Out of these 121 videos, 61 of them are of deceptive nature while the remaining 60 are of truthful nature.

The dataset contains multiple videos from one subject. In order to avoid bleeding of personalities between train and test set, we performed subject-based splitting. This ensures that videos of the same subjects are not in both training and test sets. In total, there are 59 different subjects present in the dataset.

The dataset also comes annotated with the microexpressions found in each video. There is an Excel sheet that contains columns representing 39 possible microexpressions.

Figure 4.1: Dataset Screenshots

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	OtherGestures	Smile	Laugh	Scowl	otherEyebrowMovement	Frown	Raise	OtherEyeMovements	Close-R	X-Open	Close-BE	gazeInterlocutor
2	trial_lie_001.mp4	1	0	0	0	1	0	0	1	0	0	0	1
3	trial_lie_002.mp4	1	0	0	0	0	1	0	1	0	0	0	0
4	trial_lie_003.mp4	1	0	0	0	0	1	0	0	1	0	0	1
5	trial_lie_004.mp4	1	0	0	0	1	0	0	1	0	0	0	0
6	trial_lie_005.mp4	1	0	0	0	0	1	0	1	0	0	0	0
7	trial_lie_006.mp4	1	0	0	0	1	0	0	1	0	0	0	1
8	trial_lie_007.mp4	1	0	0	0	0	0	1	1	0	0	0	1
9	trial_lie_008.mp4	1	0	0	0	1	0	0	1	0	0	0	1
10	trial_lie_009.mp4	1	0	0	0	1	0	0	1	0	0	0	1
11	trial_lie_010.mp4	0	1	0	0	0	0	1	1	0	0	0	1
12	trial_lie_011.mp4	0	0	0	1	0	0	1	0	0	0	1	0
13	trial_lie_012.mp4	1	0	0	0	1	0	0	1	0	0	0	1
14	trial_lie_013.mp4	1	0	0	0	1	0	0	0	0	1	0	1
15	trial_lie_014.mp4	1	0	0	0	1	0	0	1	0	0	0	1
16	trial_lie_015.mp4	1	0	0	0	0	0	1	0	0	1	0	1
17	trial_lie_016.mp4	1	0	0	0	0	0	1	1	0	0	0	0

Figure 4.2: Microexpressions Annotations

5. Implementation

5.1 Environment Setup & Tools

We created this project with lots of scripts, we used “python” programming language because it was used in most of the recently published work, also it was the easiest, most useful language to use due to it’s libraries, which some of them are designed specifically to help develop deep learning models.

We mainly used jupyter notebooks for our scripts, however we used 3 environments to run those scripts due to the high processing needs some of these scripts required. As for version control and how we merged our work together, we used “Github”.

5.1.1 Environments

1. Localhost
    - While developing our scripts (preprocessing, training and testing), we tried running them locally, but the processing needs for these scripts were really high so localhost wasn’t sufficient, it took too much time and lots of memory.
2. Google Colab
    - Secondly, we tried google colab, it worked for most of our scripts, however we faced lot’s of issues with it as the memory was very limited.
3. Google Cloud platform
    - Last but not least, we tried google cloud platform for scripts with high computation needs.
    - Specs of the virtual machine instance:
        - Machine Type: n1-highmem-8 (8 vCPUs, 52 GB memory)
        - GPUs: 1 x NVIDIA Tesla V100
        - Operating System: debian-10-buster-v20200413
        - Disk size: 100 GB

5.1.2 Packages and Libraries

1. Glob: this module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell.

2. OpenCV (cv2): an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.
3. NumPy: the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
4. Os: this module provides a portable way of using operating system dependent functionality.
5. Tqdm: is a progress bar library with good support for nested loops and Jupyter/IPython notebooks.
6. Tensorflow: an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
7. Keras: a deep learning framework that follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages.
8. Pandas: a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
9. Cvlib: asimple, high level, easy-to-use open source Computer Vision library for Python.
10. Scipy: a Python-based ecosystem of open-source software for mathematics, science, and engineering.
11. Random: implements pseudo-random number generators for various distributions.
12. Matplotlib.pyplot: a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a

figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

13. Sklearn.model_selection: split arrays or matrices into random train and test subsets
14. google.colab: we used files and drive packages from this library to be able to connect with google drive from google colab

5.2 Data Preprocessing

5.2.1 Subject-based Splitting

The dataset contains multiple videos from one subject. In order to avoid bleeding of personalities between train and test set, we do subject-based splitting to ensure that videos of the same subjects are not in both training and test sets. This was done by hand in a word document.

Figure 5.2 includes a snapshot from the subject-based splitting.

[0 – 60]: Deceptive

[61 - 120]: Truthful

Total = 121 videos, 57 subjects (paper says they are 56 subjects: 21 females, 35 males)

1. [0, 1, 2, 3, 4, 5, 117, 118, 119] – 9 **[DONE]**

2. [6, 7, 8, 9, 10, 11, 12, 63, 64, 65, 66, 67] – 12 **[DONE]**

3. [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 114, 115, 116] – 21 **[DONE]**

4. [31] – 1 **[DONE]**

Figure 5.1: Subject-based splitting

5.2.2 Train-Test Splitting

After the subject-based splitting, we needed to select which subjects to use for training and which subjects to use for testing. We tried to make the two sets as diverse as possible, with regards to the sex and skin color of the subjects. This was done by hand.

Table 1. Includes the numbering details.

Table 1. Subject-based splitting

	Train Set	Test Set
# videos	78	18
# truthful videos	34	14
# deceptive videos	44	4
# subjects	14	18
# females	9	7
# males	5	11
# white people	12	13
# people with color	2	5

5.3 Region of Interest Extraction Module

The real-life courtroom dataset we're using includes people in a courtroom or in an interviewing setting. As can be seen in Figure 4.1, there's "too much" background.

Our solution for deception detection is mainly focused on visual features and micro-expressions of the person who's speaking, so the background and any other object in the videos are useless to us. We needed to extract our region of interest, which is the person speaking. We used YOLO (2.5.3.5) for that purpose.

YOLO was our best option because it's the most recent deep learning model for object detection, and it's also the fastest one so far. YOLO's advantages over the other deep learning models are stated in 2.5.3.

Here's how we cropped each video: We passed the video's frames to YOLO. It returned, for each frame, a list of bounding boxes and a list of labels. We ignored

any bounding boxes that had any label other than ‘person’. If more than one bounding box is found in one frame, they were merged to form one bigger bounding box enclosing them, so for each frame we had only one bounding box.

We then found the maximum bounding box (the one with the largest area) across all frames. We cropped every frame of the video according to the maximum bounding box.

Results:

5.4 Data Clipping Module

After the region of interest has been extracted from each video, and after the train-test splitting, we needed to further preprocess the videos before sending them to the 3D CNN. We needed all the videos to be of the same length (same number of frames), we had two options:

1. Upsample every video to be of the same length as the longest video, by duplicating the frames.
2. Divide every video into shorter subclips having the same length of the shortest video.

The first option wasn't doable because with the resources we have, we wouldn't have been able to train the 3D CNN with long videos as input, and as the role of the 3D CNN is to extract visual features, dividing each video into subclips wouldn't affect the results, so we went with the second option.
This is how the preprocessing went:

1. Find the length of the shortest video (113 frames).
2. For every video, sequentially take 113 frames as a subclip, until less than 113 frames remain.
3. If the frames are less than 57 (50% of 113), we discard them. If more than 57 frames remain, we take some frames from the previous subclip to make them 113 frames.

This resulted in 806 subclips, (692 train videos, 114 test videos).

5.5 Visual Feature Extraction Module

5.5.1 3D-CNN

As the field of **Deception Detection** hasn't been widely explored by other researchers, we haven't found any pre-trained modules for **Expression Detection**, that's why we had to build a module from scratch using an **Action Detection** module paper [21] as our base.

The paper we decided to follow "Deep Learning Approach" mentioned another paper [21] for a 3D-CNN architecture for action detection with strong results due to the good accuracies achieved.

By following the architecture in the aforementioned paper "3D Convolutional Neural Networks for Human Action Recognition" we started implementing the following architecture in Figure 2.3.

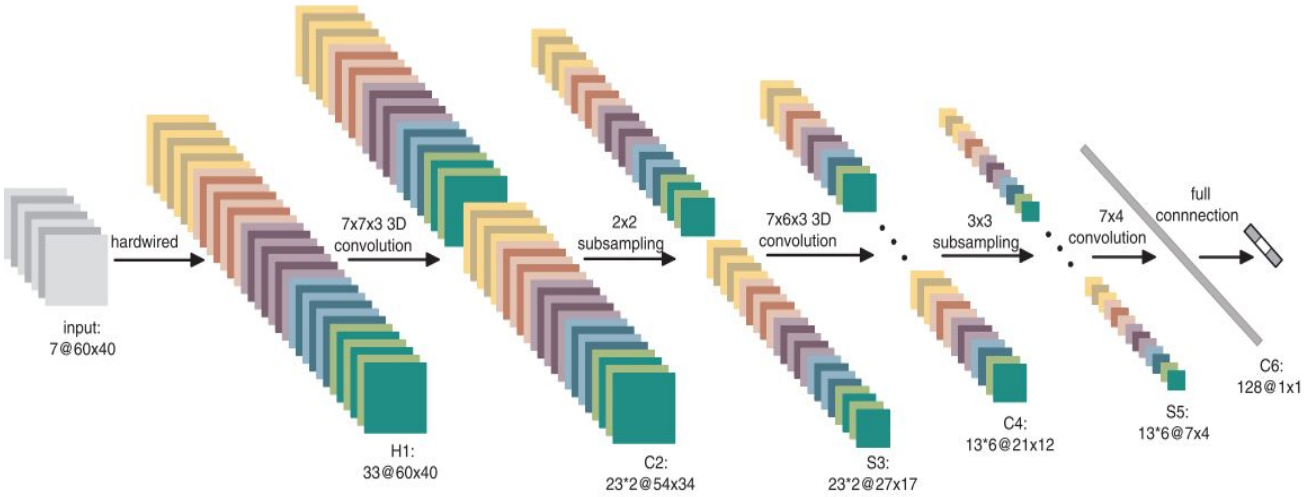

Figure 5.2: 3D CNN Architecture

Its structure consisted of 7 layers, 1 hardwired layer, 3 convolution layers, 2 subsampling layers and 1 fully connected layer.

- **Input Layer:**
  - Input videos were divided into 7-frame volumes with 60x40 spatial dimensions and sent in batches to the first layer.
  - We used a subsampling approach to reduce the number of frames used while conserving the integrity of the dataset with different values for different trials, mainly a step of 2.
- **First Layer (Hardwired Layer H1):**
  - Takes an input 7-frame volume with the dimensions (7@60x40)
  - Calculates the x and y-gradients to each frame.
  - Calculates the optical flow between each 2 consecutive frames.
  - Concatenates the original grey 7 frames, the 2*7 gradient frames for each frame and 6 optical-flow frames for each 2 consecutive frames in one volume.
  - Resulting in an output volume of 33-frame volume with the dimensions (33@60x40).

- **Second Layer (1st Convolution Layer C2)**
    - Applying a 3D convolution with filter (7x7x3) on the output of **H1**:
        - On the first 7 grey frames [0-6].
        - On the following 7 x-gradient frames [7-13].
        - On the following 7 x-gradient frames [14-20].
        - On the following 6 x-optical flow frames [21-26].
        - On the following 6 y-optical flow frames [27-32]
    - Activation function used is **tanh**.
    - Applies the past 3D convolution operation on the input volume twice.
    - Resulting in 2 separate output volumes each with dimensions (23@54x34).
- **Third Layer (1st Subsampling Layer S3)**
    - Applies a subsampling operation with a filter (2x2).
    - Once on each of the 2 output volumes of the **C2** output**.**
    - Wrestling in 2 separate volumes, each with dimensions (23@27x17).
- **Fourth Layer (2nd Convolution Layer C4)**
    - Applying a 3D convolution with filter (7x6x3) on the output of **S3**:
        - On the first 7 grey frames [0-4].
        - On the following 5 x-gradient frames [5-9].
        - On the following 5 x-gradient frames [10-14].
        - On the following 4 x-optical flow frames [15-18].
        - On the following 4 y-optical flow frames [19-23]
    - Activation function used is **tanh**.
    - Applies the past 3D convolution operation on each of the 2 input volumes 3 times.
    - Resulting in 6 different output volumes, each with dimensions (13@21x12).
- **Fifth Layer (2nd Subsampling Layer S5)**
    - Applies a subsampling operation with a filter (3x3).
    - Once on each of the 6 output volumes of the **C4** output.
    - Wrestling in 6 separate volumes, each with dimensions (13@7x4).

- **Sixth Layer (3rd Convolution Layer C6)**
    - Applying a 2D convolution with filter (7x4) on the output of **H1**:
        - On the first 3 grey frames [0-2].
        - On the following 3 x-gradient frames [3-5].
        - On the following 3 x-gradient frames [6-8].
        - On the following 2 x-optical flow frames [9-10].
        - On the following 2 y-optical flow frames [11-12]
    - Activation function used is **tanh**.
    - Applies the past 2D convolution operation on each of the 6 input volumes 1 time.
    - Concatenates the resulting 6 volumes in 1 volume with dimensions (128@1x1).
- **Seventh Layer (Fully connected Layer FC)**
    - Applies a Dense operation with 128 units on the output of **C6.**
    - Activation function used is **tanh**.
- **Output Layer**
    - Classification output is obtained using another Dense operation with 2 units on the output of **FC.**
    - Activation function used is **softmax**.
- **Video classification**
    - In prediction and testing per video, we take the majority voting of all the video clips' classification.

5.5.2 Visual Feature Extraction MLP

For extracting visual features from the videos, we use 3D-CNN. 3D-CNN has achieved state-of-the-art results in object classification on tridimensional data. 3D-CNN not only extracts features from each image frame, but also extracts spatiotemporal features from the whole video which helps in identifying the facial expressions such as smile, fear, or stress.

The input to 3D-CNN is a video v of dimension (c; f; h;w), where c represents the number of channels and f; h; w are the number of frames, height, and width of

each frame respectively. A 3D convolutional lter, fl of dimension (fm; c; fd; fh; fw) is applied, where fm = number of feature maps, c = number of channels, fd = number of frames (also called depth of the lter), fh = height of the lter, and fw = width of the lter. This lter, fl, produces an output, convout of dimension (fm; c; f - fd + 1; h - fh + 1;w - fw + 1) after sliding across the input video, v. Max pooling is applied to convout with window size being (mp;mp;mp). Subsequently, this output is fed to a dense layer of size df and softmax. The activations of this dense layer is used as the visual feature representation of the input video, v.

In our experiments, we consider only RBG channel images, hence c = 3. We use 32 feature maps and 3D filters of size, fd = fh = fw = 5. Hence, the dimension of the filter, fl, is 32 3 5 5 5 . The window size, mp, of the max pooling layer is 3. Finally, we obtain a feature vector, vf , of dimension 300 for an input video, v.

- **First Layer (CONV)**
  It is a 3D convolution layer with kernel size 5 , with no padding and ReLU activation function.
- **Second Layer (POOL)**
  It is a 3D max pooling layer with kernel size 3.
- **Third Layer (Vf)**
  Fully-connected layer of 300 nodes for the visual features with ReLU activation function.
- **Fourth Layer (Hidden_Layer)**
  Fully-connected layer of 1024 nodes with ReLU activation function.
- **Fifth Layer (dropout)**
  Dropout layer for regularization purposes with 50% drop probability.
- **Sixth Layer (MLP_output)**
  Output layer of two nodes and Softmax activation function.

5.6 Data Augmentation Module

5.6.1 First trial: (Extending videos with noise)

First solution to make all videos with the same size, was to augment data by extending all videos to have the same number of frames. After looking at all video

frames, the maximum number of frames in all of the videos was 2443, so accordingly, all videos were set to be extended to 2443 frames.

Our solution for adding extra frames was to repeat the frames of each video until it reaches the maximum number of frames, but the repeated frames are altered with gaussian noise.

How it worked:

1. First of all we read all the dataset, and return 2 lists, list of the videos and a list containing the fps (frames per second) for each video.
2. We used the “cv2” library to read the videos by using the “VideoCapture” method, and “CAP_PROP_FPS” property to get fps (frames per second) for each video.
3. After that we started by looping on each video, and checking the number of frames the video has using “cv2.CAP_PROP_FRAME_COUNT” and if it was less than maximum frames, then that video gets extended.
4. In the function that extends the video, we start by getting video frames and saving them in a list. Then we add noise to those frames and save them in a new list. After that we check how many times the video is gonna repeat itself and append the noisy frames list to the original frames list with that amount. If after that the frames are still less than the video, then we append them from the noisy frames list with the remaining number of frames.

Figure 5.3: Frame without noise

5. As for adding noise to frames, we loop on each frame and add Gaussian noise to it. We create a gaussian noise image, then add it to each channel, the parameter for the Gaussian image is sigma, where sigma is a number to the power of another number. We identified sigma to be = 20^0.5, if we increased the 20 number the noise of the image increases and vice versa. Then the frame after adding noise to it is saved as a NumPy array and returned.

Figure 5.4: Frame with noise

6. After that we have a list of lists, each list contains the new frames of the video, these lists are then converted back to a video each, using the fps we obtained at the beginning and save them to the folder using "cv2.VideoWriter" method.

5.6.2 Analysing dataset

So after applying the first solution, the videos were pretty large, having a dataset of 2443 frame each was pretty huge, especially when trying to train that kind of size per video, the initial number of parameters exceeded 500k and it increased with every layer, so, we had to search for an alternative solution, one that could minimize the number of frames per video.

How did we analyse the dataset?

1. We created a script, first step was reading all videos and saving the number of frames for each video

2. Second step was to check for outliers. In the first solution of data augmentation, we extended every video so that it will match the largest video we have, but what if that large video was an outlier?
3. After sorting the list that contained video frames number for each video, it turned out that the maximum video of 2443 we were using, was in fact an outlier, and the average video have about 748 frames, and there were like 4 outlier videos with these values:

[1600, 1872, 2144, 2162, 2443]

As we can see, the number of video frames increased dramatically, when as a matter of fact, the first video before those, was 1440 frames.

After that realization, we decided to switch the number of frames we use in the VGG model to be 1440 frames instead of 2443, which has a huge impact on the performance of the model.

But why did we choose 1440 instead of taking the average number of frames 748? Because each video contains micro-expressions, that we don't know when exactly in the video do they occur, so if we skipped that much frames, the accuracy of the training labels wouldn't be correct, as we could have skipped a micro-expression, and then later on, the label indicates that this micro-expression appears in the video when it's not anymore, and then we'd be training the model with wrong data. Instead, we decided to go with 1440, so that we'd reserve the accuracy of the data as much as we can.

But now we need to handle the rest of the videos, to make them all have the same frames number, and here comes our next solution and second trial.

5.6.3 Second trial: (Extend/shrink video to average)

Second solution was to choose a different video to normalize the other videos to it, so the new video frames number was 1440. The code works the same way as the previous one for the videos to be extended. But if the video was to be shrunk from for example 2000 frames to 1440, then we have to remove frames instead of adding them.

How it works:

1. We calculate skipping rate for the video frames, after that rate we skip a frame, so that the video doesn't get affected.
2. Skipping rate is calculated as follows:
    - We calculate the number of extra frames in the video
    - Then divide maximum_frames by (number of extra frames + 1)
3. We then loop on all the frames of the video, if the length of the list of the new video equals to the maximum number of frames we break the loop
4. Then inside the loop we check the current index with the skipping rate, if the current index is divisible by the index rate, then we skip this frame and append it to the list of skipped frames
5. Otherwise, we append the frame to the new video list
6. If after the loop ends, the video list is less than the maximum number of frames due to the skipping, we add the missing number of frames back to the new video list. The missing frames are added from the skipped frames list at the same place they were skipped from.
7. Then the new list is returned

5.7 Micro-Expressions Extraction Module

5.7.1 VGG

1. What are Micro-Expressions?
    1.1. Micro-expressions are facial expressions that occur within a fraction of a second. This involuntary emotional leakage exposes a person's true emotions.
    1.2. At 1/25th of a second, micro-expressions can be difficult to recognize. Yet with training you may learn to spot them as they occur in real-time.

2. How did we deal with detecting them in the videos?
    2.1. Micro-expressions are an important part of training our deception detection model, while training our model, we already had the micro expressions available in each video in an external sheet. But what about when we need to test an unseen video from an external dataset, how can we know the micro-expressions in it? So we came up with

the solution of training a VGG model to detect micro-expressions in videos.

2.2. To be able to train to train the VGG model, we needed 2 things, the input and the output (label). The input was the video, which we applied Data Augmentation upon to be able to use it (check section #), and the label of each video. But for this case the label was different, for one video there could be tenth of micro-expressions available in that video, so to solve that, we decided on creating new classes (labels) for the micro-expressions we have.

3. Creating Micro-Expressions classes

3.1. Since we have from the dataset a sheet that contains all micro-expressions available in each video, we created a python script that reads that sheet, and converts the entries to a list. Where each entry is a list of zeros and ones, zero if that micro-expression doesn't exist in
the video and one if it appears in the video.

3.2. After that each list was concatenated to form a string of zeros and ones, and then all strings were put in a set so that we can obtain the unique strings. These new strings which were 105, are now our new labels, where each class represents a set of micro-expressions.

4. One Hot Encoding

One-hot encoding is used to represent the labels of classes, it represents each class using n bits where n is the number of classes. All bits except one are '0', and one bit is '1', the index of the '1' value represents the class number.

For example, if we have 3 classes, their one-hot encoded labels are going to be: 001, 010, 100.

We used one-hot encoding to represent the labels of the 105 micro-expressions classes we have, because VGG uses one-hot encoding in the output layer.

6. Experiments

6.1 Micro-expressions extraction with VGG

Videos are 1440 frames but we can't run such videos so we skip some frames.

Experiment #1:

- Specifications:
  - 96 videos (288, 224, 224, 3) skipping 5 frames
  - 100 epochs
  - 0.001 learning rate
- Results: OOM on first epoch

Experiment #2:

- Specifications:
  - 96 videos (144, 224, 224, 3) skipping 10 frames
  - 100 epochs
  - 0.001 learning rate
- Results: OOM on first epoch

Experiment #3:

- Specifications:
  - 96 videos (288, 112, 112, 3) skipping 5 frames
  - 100 epochs
  - 0.001 learning rate
- Results: OOM on first epoch

Experiment #4:

- Specifications:
  - 96 videos (144, 112, 122, 3) skipping 10 frames
  - 100 epochs
  - 0.001 learning rate
- Results: OOM on first epoch

Experiment #5:

- Specifications:
  - 96 videos (144, 112, 122, 3) skipping 10 frames and removing some blocks
  - 100 epochs
  - 0.001 learning rate
- Results: OOM on first epoch

Conclusion:

Due to the resources limitations we failed to train the model, although all experiments were conducted on Colab and Google Cloud.

6.2 Visual Features Extraction

6.2.1 3D-CNN

The dataset was first preprocessed where each volume resulted in samples of volumes, and each volume contained 7 frames.

Train: 11764 samples. Validation/Test: 1938 samples.

All the experiments were run on 20 epochs. Table 2 contains the accuracies acquired on the architecture explained in Section 5.4.1.

Table 2. 3D-CNN Experiments

Experiment Number	Batch Size	Checkpoint Weight used	Train Set Accuracy	Validation Set Accuracy	Majority Voting Test Accuracy
1	4	No	99.7%	57%	64%
2	4	Yes	99.7%	66%	60%
3	6	No	100%	60%	64%
4	6	Yes	100%	68%	**68%**
5	8	No	99.99%	55%	48%
6	8	Yes	99.99%	60%	60%
7	16	No	99.8%	59%	60%
8	16	Yes	99.8%	67%	60%
9	32	No	99.8%	55%	64%
10	32	Yes	99.8%	63%	56%

Conclusion:

From Table 2. We can see that due to the nature of the architecture being deep it overfitted on the training samples and failed to generalize on the unseen samples.

6.2.2 MLP

Experiment #1:

- Specifications:
  - 96 videos (112, 214, 214, 3).
  - 20 epochs
  - Batch size: 2
  - 0.001 learning rate
- Results: OOM on first epoch.

Experiment #2:

- Specifications:
  - 96 videos (28, 214, 214, 3).
  - 20 epochs, with early stopping.
  - Batch size: 4
- Results: the model's accuracy after early stopping was 52%. Using the weights from a checkpoint that tracked the validation set loss results in accuracy 48% after predicting the testset using majority voting.

Experiment #3:

- Specifications:
  - 96 videos (28, 214, 214, 3).
  - 50 epochs.
  - Batch size: 4
- Results: Using the weights from a checkpoint that tracked the validation set loss results in accuracy 60% after predicting the testset using majority voting.

Any experiment conducted with batch size more than 4 resulted in OOM.

Conclusion:

Due to the limitations caused by the available resources, downsampling was attempted which resulted in loss of temporal data found in-between the frames transitions.

The intuition behind this model that it would outperform any other visual feature extractor as in [22]. Unfortunately, due to the absence of the input specifications and the downsampling techniques made, we were not able to redo their experiment.

6.3 Visual Features + Micro-expressions

Experiment:

We concatenated the ground truth micro-expressions annotations that came with the dataset with the input volume (7 frames of a video subclip) before sending it to the 3D CNN, we considered the micro-expressions of one subclip to be the same of the original video, so we just took the 39 bits representing the appearances of the microexpressions and concatenated them with the input vector before training.

Results: We reached an accuracy of 78%.

Conclusion:

Micro-expressions improved the model's ability to detect deception. The accuracy could have been significantly further improved if we had the micro-expressions of each subclip, or if we had the processing power needed to train the model using complete videos.

7. User Manual

1. The user chooses the video that needs to be classified into deceptive or truthful.

Figure 7.1: Welcome Page

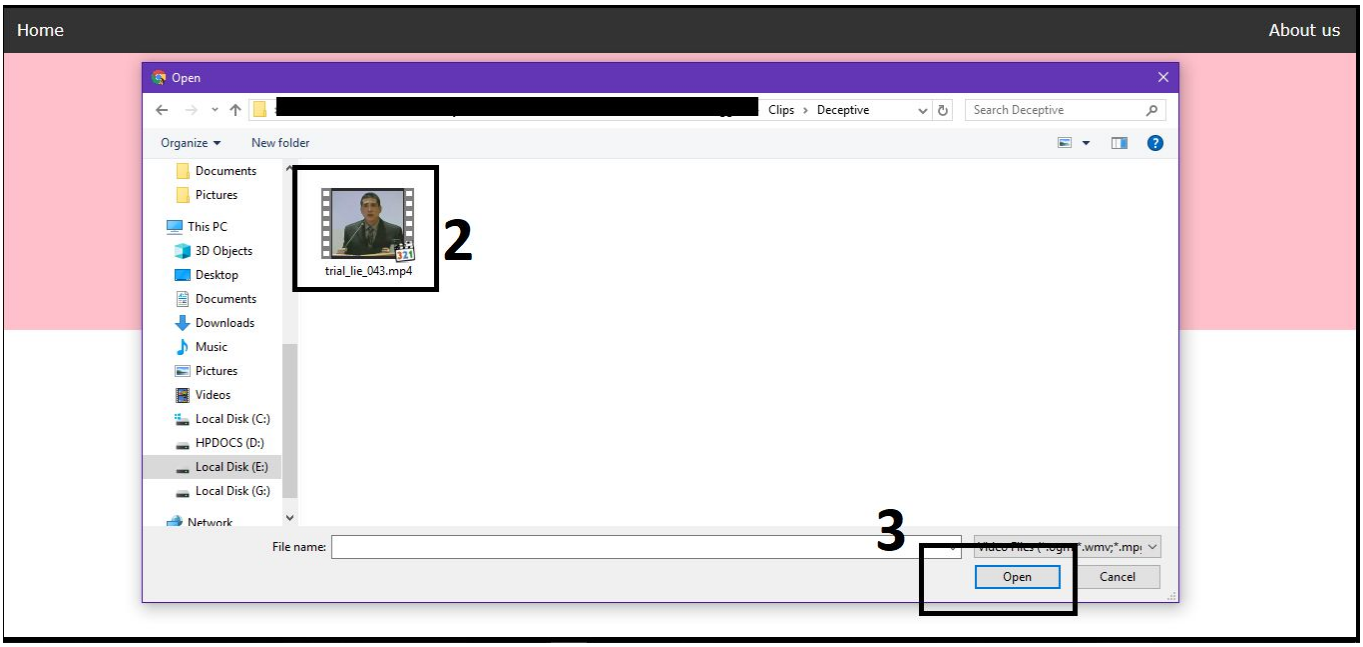

Figure 7.2: Video Selection

2. The user clicks upload after choosing the video.

Figure 7.3: Uploading Video

3. The user will be redirected to another page to see the result of the video classification.

Figure 7.4: Classifier's Verdict

8. Conclusion & Future Work

8.1 Conclusion

The aim of this project was to explore the state-of-the-art architectures in the deception detection field with one being recently published achieving accuracy of 96.14% [22]. But their experiments were vague with most of the settings left unmentioned.

We implemented a 3D-CNN Deception Detection classifier that takes a video cropped into equal sized clips with only the RoI in them and uses major voting from all the clips in classifying the original video, with a maximum accuracy reached of **68%**.

Using VGG we attempted to extract the micro-expressions from each video prior to classifying it, but due to the limitation of the memory and computation resources that we had, we weren't successful.

By using the already available micro-expressions annotations that came with the dataset, we concatenated it with the output of the fully connected layer in the 3D-CNN Deception Detection classifier and found that the resulting accuracy increased noticeably up to **78%** accuracy, in one of the trials.

In one approach we attempted following the "DeepLearning Approach" [22] paper's approach to Deception Detection by implementing the MLP module mentioned in it, but due to the missing information about the used hyper parameters such as their down sampling approach, the maximum accuracy reached was **60%**.

8.2 Future work

1. To increase the size of the dataset.
2. Gather a less domain-dependent dataset.
3. To acquire more resources in memory and computation power and run the VGG Micro-Expression extraction model.
4. Use audio and text features extraction modules.

5. Enhance performance by:
    - Concatenating the extracted Micro-Expressions to the extracted visual features from the 3D-CNN model.
    - Concatenating the output of the audio and text extraction modules to the models.

References

[1] Bond, C. F., Jr., & DePaulo, B. M. (2006). Accuracy of Deception Judgments. Personality and Social Psychology Review, 10(3), 214–234. https://doi.org/10.1207/s15327957pspr1003_2

[2] A. Vrij. Detecting Lies and Deceit: The Psychology of Lying and the Implications for Professional Practice. Wiley series in the psychology of crime, policing and law. Wiley, 2001.

[3] T. Gannon, A. Beech, and T. Ward. Risk Assessment and the Polygraph, pages 129–154. John Wiley and Sons Ltd, 2009.

[4] S. Feng, R. Banerjee, and Y. Choi. Syntactic stylometry for deception detection. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2, ACL '12, pages 171–175, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[5] J. Hirschberg, S. Benus, J. Brenier, F. Enos, S. Friedman, S. Gilman, C. Gir, G. Graciarena, A. Kathol, and L. Michaelis. Distinguishing deceptive from non-deceptive speech. In In Proceedings of Interspeech 2005 - Eurospeech, pages 1833–1836, 2005.

[6] M. Newman, J. Pennebaker, D. Berry, and J. Richards. Lying words: Predicting deception from linguistic styles. Personality and Social Psychology Bulletin, 29, 2003.

[7] P. Ekman. Telling Lies: Clues to Deceit in the Marketplace, Politics and Marriage. Norton, W.W. and Company, 2001.

[8] Zetter, Kim. (2017, June 04). What a Half-Smile Really Means." from https://www.wired.com/2003/09/what-a-half-smile-really-means/

[9] K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition", arXiv:1409.1556.

[10] Ross Girshick et al., Rich feature hierarchies for accurate object detection and semantic segmentation, 2013, arXiv:1311.2524v5.

[11] Ross Girshick, Fast R-CNN, 2015, arXiv:1504.08083v2.

[12] Shaoqing Ren et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2015, arXiv:1506.01497v3.

[13] Joseph Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, 2015, arXiv:1506.02640v5.

[14] Ekman, P.: Telling lies: Clues to deceit in the marketplace, politics, and marriage (revised edition). WW Norton & Company (2009).

[15] Krishnamurthy, Gangeshwar, et al. "A deep learning approach for multimodal deception detection." arXiv preprint arXiv:1803.00344 (2018).

[16] Zhe Wu, Bharat Singh, Larry S. Davis, V. S. Subrahmanian, “Deception Detection in Videos”, arXiv:1712.04415v1 (2018).

[17] D. Avola, L. Cinque, G. L. Foresti, and D. Pannone, Automatic Deception Detection in RGB videos using Facial Action Units, 2019.

[18] Mingyu Ding, An Zhao, Zhiwu Lu, Tao Xiang, Ji-Rong Wen (cs.CV), Face-Focused Cross-Stream Network for Deception Detection in Videos, arXiv:1812.04429 (2018).

[19] Michel Owayjan, Ahmad Kashour, Nancy Al Haddad, Mohamad Fadel, and Ghinwa Al Souki, “The Design and Development of a Lie Detection System using Facial Micro-Expressions” (2012).

[20] Perez-Rosas, V., Abouelenien, M., Mihalcea, R., Burzo, M.: Deception detection using real-life trial data. In: Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, ACM (2015).

[21] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotem-poral features with 3d convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision. (2015)

[22] Gangeshwar Krishnamurthy, Navonil Majumder, Soujanya Poria, and Erik Cambria. A Deep Learning Approach for Multimodal Deception Detection, 2018.