كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

جامعة حلوان
Helwan University

Faculty of Computers and Artificial Intelligence

Computer Science department

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Computer Science Department.

Vehicles accidents prediction system

A graduation project dissertation by:

[Alaa Mohamed Abdelrehem(20180348)]

[Mai Sherif Helal (20180638)]

[Rania Abdullah mohamed (20180234)]

[Ahmed Roshdy Mohamed (20170025)]

[Karim Hassan Mansour (20180426)]

Supervised by: Dr.Wessam el-Behaidy

June 2022

Acknowledgement

We thank the almighty for giving us the courage and perseverance in completing the graduation project. This project itself is acknowledgement for all those people who have given us their heartfelt co-operation in making this project a grand success. With extreme jubilance and deepest gratitude, we would like to thank our mentor, Dr.Wessam el-Behaidy, for her constant encouragement and support. We are profoundly grateful towards the unmatched services rendered by all the team members. Our special thanks to Eng.Wael Eid for his technical support and all the faculty of the computers and artificial intelligence and peers for their valuable advice at every stage of this work. Finally, we would like to express our deep sense of gratitude and earnest thanksgiving to our dear parents for their moralsupport and heartfelt cooperation in doing the graduation project.

Abstract

According to the World Health Organization, there are about 1.35 million deaths and 20-50 million injuries because of the car accident globally every year. Especially, a certain proportion of deaths and injuries are due to untimely treatment and secondary accidents, which results from that rescue agency and vehicles around accidents cannot obtain quick response about the accident. Therefore, it is vital important to develop an efficient accident prediction system, which can significantly reduce both the number of deaths and injuries as well as the impact and severity of accidents. Under this background, many fundamental projects, and studies to develop efficient prediction methods have been launched for developing and testing. The proposed system predicts vehicles accidents, it uses Auto encoder with conditional generative adversarial networks for future frame prediction and the predicted frames passes through convolutional neural networks and the result of CNN is to classify which it is accident or not under accuracy 95%.

Contents

List of figures

List of tables

Chapter one

Introduction

In this chapter, we describe the main objective of the project, the problem that must be solved, and what is our perception of the application of the methods, and we say what will happen in the other chapters.

1.1 Brief Introduction

According to the World Health Organization, there are about 1.35 million deaths and 20-50 million injuries because of the car accident globally every year. Especially, a certain proportion of deaths and injuries are due to untimely treatment and secondary accidents, which results from that rescue agency and vehicles around accidents cannot obtain quick response about the accident. Therefore, it is important to develop an efficient accident detection system, which can significantly reduce both the number of deaths and injuries as well as the impact and severity of accidents. Under this background, many fundamental projects, and studies to develop efficient detection methods have been launched for developing and testing which will result in making Ambulance & fire fighting vehicles able to reach the Accident location faster.

1.2 Motivation

As we present in the first part of introduction, our proposed system is accident detection system using cameras inside cars. Accidents take a lot of lives not every day, we could say every hour passes that there's an Accident somewhere on our planet. Then we start thinking, what if we could predict the Accidents before it happens a few minutes ago or even seconds? Isn’t this time could save one precious human life? Then if we could implement our proposed system, it would deserve that effort. As in

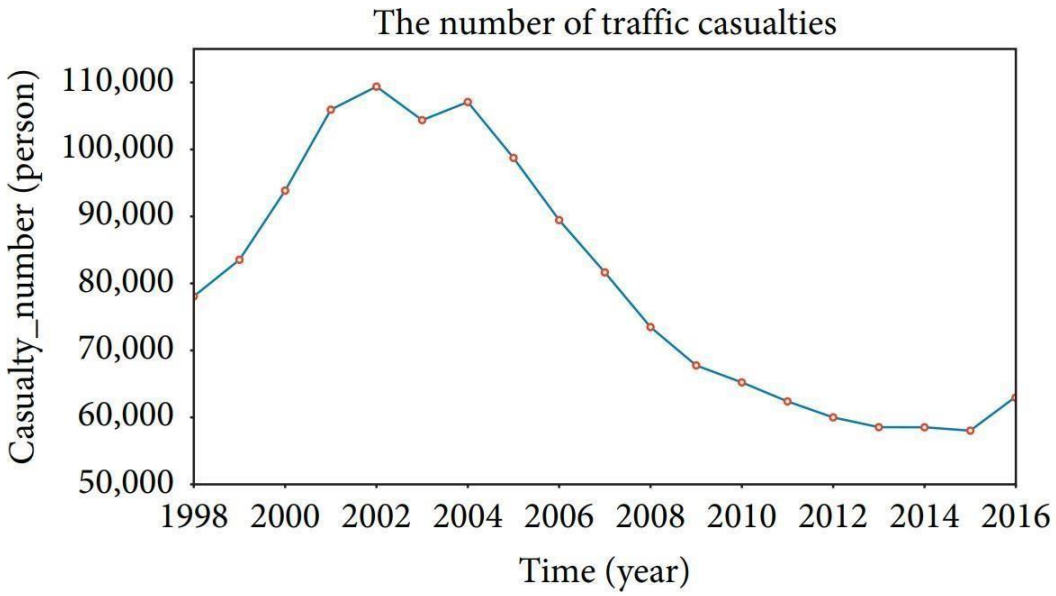


Figure 1.1: number of deaths

the figure below shows how the number of deaths decreased through theyears when a similar system applied in China between 1998 & 2016 [2].

1.3 Problem definition

In the last decades, the detection of accidents was by humans using radars and monitoring cameras which lead to low performance.

Recently, the publishing of AI and machine learning techniques helped in developing vehicle trajectory prediction.

It aims to decrease the number of deaths, as it notifies the ambulance earlier by predicting future frames and classify it if there is an accident or not. Cause the number of vehicle road accidents increases daily.

1.4 Similar Systems

1.4.1 An LSTM Network for Highway Trajectory Prediction

Hegde et al. [1] describe the problem of vehicle trajectory prediction as soon, transportation autonomy will be ubiquitous, and trajectory prediction will aid decision-making. Theydevelop data-driven vehicle trajectory prediction model using Generative Adversarial Networks (GANS). They use Bilateral filtering and Adaptive Gaussian thresholding in preprocessing phase and to extract features they use the feature extractor long short-term memory (LSTM). YOLOV3 algorithm is used in object detection and tracking phase. In the final phase: trajectory prediction, they predicted itusing GAN. The VisDrone dataset and the TRAF dataset were used in thisstudy. The Average Displacement Error (ADE) and The Final Displacement Error (FDE) are the evaluating metrics used. When compared to current trajectory prediction methods such as linear regressor, LSTM, Social LSTM, and others, the displacement errors achieved using GAN are significantly lower.

1.4.2 Traffic Accident Prediction Based on LSTM-GBRT Model

Zhang et al. [2] describe the problem of Road traffic accidents are a concrete manifestation of road traffic safety levels, it can be applied in the traffic management system to help make scientific decisions. Traffic Accident Prediction Model Based on long short-term memory (LSTM), and gradient boosted regression trees (GBRT). The LSTM-GBRT model is proposed by combining the two methods, long short-term memory (LSTM), and gradient boosted regression trees (GBRT). The LSTM neural network is used to extract the features with time-dependent information. The features are trained by the GBRT model to predict traffic accidents. By using Boosting Ensemble Learning Framework, technical framework that combines multiple different models to perform the corresponding tasks to achieve more efficient and accurate arrival. After training, RMSLE (root mean square

logarithmic error) and R-square (statical measure of fitting) are the evaluating metrics used to measure performance. Compared with the traditional regression model like the traditional BP neural network model, the LSTM neural network model, and the GBRT model, The experimental results show that the LSTM-GBRT model has the strongest ability to fit the data and the variable has the best interpretability for the predicted.

1.4.3 Anomaly prediction system

Emad et al. [3] describe the problem of predicting the anomalies before it happens. This was achieved by using various techniques. Starting from the feature extractor “C3D algorithm”. Retrospective Cycle GAN for future frame prediction. And multiple instance learning “MIL” to classify the predicted frames as anomaly “positive bags” or normal “negative bags”. The datasets used in this paper were The CUHK Avenue Dataset & ShanghaiTech Dataset. In the Avenue Dataset, the accuracy improved by 8.471% to 68.94% compared with the anomaly detection model with no future frame prediction. Whereas the ShanghaiTech Dataset showed no noticeable improvement and reached 88.26%. They proved that we could depend on the future frames generated from the GAN model for anomaly prediction.

1.5 Documentation overview

1.5.1 Chapter two (Background)

- introduces all approaches that were implemented to solve our Problem.

1.5.2 Chapter three (Related work)

- In this chapter we mention the summarization of all the papers that Related to our system.

1.5.3 Chapter four (proposed model)

- This chapter explains our system architecture in detail.

1.5.4 Chapter five (system results and performance)

- This chapter mentions the expected results and describes the performance of the implemented system.

1.5.5 Chapter six (conclusion and future work)

- discusses the conclusions of the study and gives some recommendations for Possible future work.

1.5.6 References

- Here we put the citation of each paper in related work.

Chapter Two

Background

In this chapter we explain every algorithm and how it works with its basic architecture in general.

2.1 Auto encoder

A type of artificial neural network called an auto encoder is used to develop efficient coding for unlabeled data (unsupervised learning). By attempting to recreate the input from the encoding, the encoding is checked and enhanced. By training the network to disregard inconsequential input ("noise"), the auto encoder learns a representation (encoding) for a set of data, generally for dimensionality reduction.

There are variants that try to push learnt representations to take on useful properties. Regularized auto encoders (Sparse, Denoising, and Contractive) are examples of regularized auto encoders, which are useful for learning representations for later classification tasks, and Variational auto encoders, which may be used as generative models. Face recognition, feature identification, anomaly detection, and understanding the meaning of words are all challenges that auto encoders are used to solve. Auto encoders are also generative models in that they may produce new data at random that is like the input data (training data).

2.1.1 Basic architecture

-An auto encoder is made up of two parts: an encoder that converts the input into code and a decoder that converts the code back into the input.

-Duplicating the signal is the easiest approach to do the copying work flawlessly. Instead, auto encoders are frequently compelled to approximate the input, maintaining just the most important parts of the data in the copy.

-Auto encoders have been a popular concept for decades. Applications stretch back to the 1980s. Their most common usage was for dimensionality reduction or feature learning, but the notion has now spread to learning generative data models. Some of the most powerful AIs in the 2010s involved auto encoders stacked inside deep neural networks.

-An auto encoder is a feed forward, non-recurrent neural network that employs an input layer and an output layer coupled by one or more hidden layers, comparable to single layer perceptron that participate in multilayer perceptron (MLP). The number of nodes (neurons) in the output layer is the same as in the input layer. Instead of forecasting a target value display style Y given inputs display style X, it reconstructs its inputs (minimizing the difference between the input and the output). Auto encoders learn unsupervised as a result.

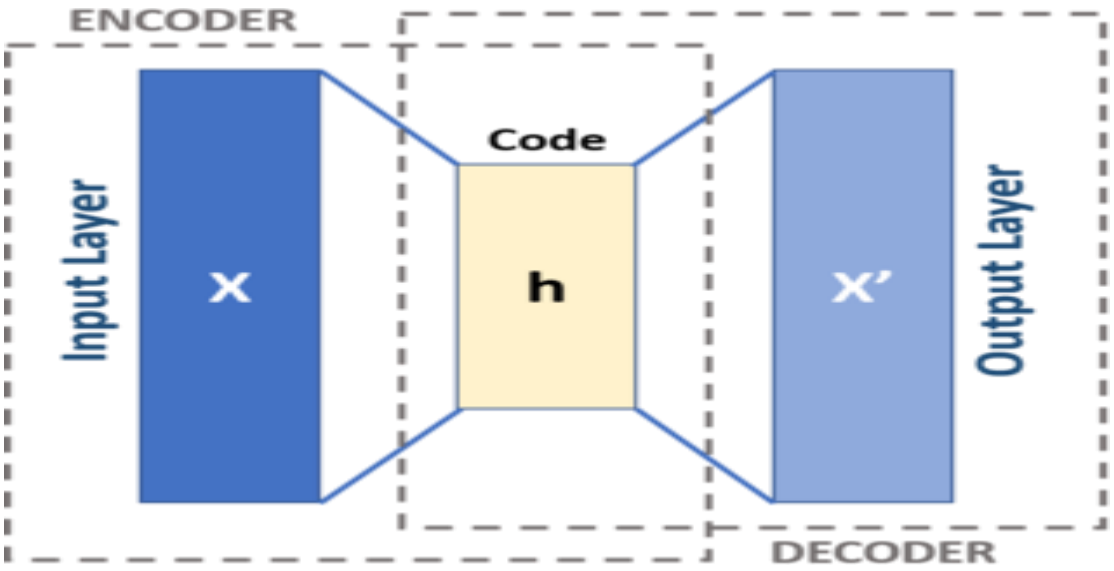


Figure 2.1: Auto encoder

2.1.2 advantages of depth

Auto encoders are often trained with a single layer encoder and a single layer decoder but using many-layered (deep) encoders and decoders offers many advantages.

- Depth can exponentially reduce the computational cost of representing some functions.
- Depth can exponentially decrease the amount of training data needed to learn some functions.
- Experimentally, deep auto encoders yield better compression compared to shallow or linear auto encoders.

2.1.3 Training

For training many-layered deep auto encoders, Geoffrey Hinton developed the deep belief network approach. His approach is considering each adjacent pair of two layers as a constrained Boltzmann

machine so that pre-training approximates a suitable solution, then fine-tuning the results through back propagation.

For deep auto-encoders, researchers have questioned whether combined training (i.e., training the entire architecture together with a single global reconstruction target to improve) is superior. In comparison to the layer wise technique, joint training learns stronger data models and more representative features for classification, according to 2015 research. Their trials, however, revealed that the effectiveness of joint training is highly dependent on the regularization procedures used.

2.1.4 Types of Auto encoder

2.1.4.1 Sequence-to-sequence auto encoder

If your inputs are sequences rather than vectors or 2D pictures, you may wish to utilize an encoder and decoder, such as an LSTM, to capture temporal structure. To create an LSTM-based auto encoder, first use an LSTM encoder to convert your input sequences into a single vector containing information about the entire sequence, then repeat this vector n times (where n is the number of time steps in the output sequence), and finally use an LSTM decoder to convert this constant sequence into the target sequence.

2.1.4.2 Variational auto encoder (VAE)

Variational auto encoders are a newer and more innovative approach to auto encoding.

You might be wondering what a variational auto encoder is. It's a form of auto encoder that learns encoded representations with additional limitations. It's an auto encoder that learns a latent variable model for the data it's given. Instead of teaching your neural network to learn any

function, you're teaching it to learn the parameters of a probability distribution that models your data. You can generate fresh input data samples by sampling points from this distribution: a VAE is a "generative model."

How does a variational auto encoder work?

To begin, an encoder network converts the input samples x into two latent space parameters called z mean and z log sigma. Then, using z = z mean + exp (z log sigma) * epsilon, we take a random sample of comparable points z from the latent normal distribution that is presumed to have generated the data. Finally, a decoder network connects these latent space points to the incoming data.

The model's parameters are trained using two loss functions: a reconstruction loss that forces the decoded samples to match the initial inputs (just like our previous auto encoders), and a regularization term based on the KL divergence between the learned latent distribution and the prior distribution. Although it aids in learning well-formed latent spaces and reduces over fitting to the training data, this latter phrase might be dropped completely.

2.1.4.3 Convolutional auto encoder

Because our inputs are pictures, convolutional neural networks (convnets) make sense as encoders and decoders. Convolutional auto encoders are always used on pictures in practical contexts since they perform significantly better.

The encoder will consist of a stack of Conv2D and MaxPooling2D layers (max pooling being used for spatial down-sampling), while the decoder will consist of a stack of Conv2D and UpSampling2D layers.

2.1.5 Applications

Dimensionality reduction, Principal component analysis, Information retrieval, Anomaly detection, Image processing, Drug discovery Popularity prediction, Machine translation.

2.2 Generative adversarial networks (GAN)

In June 2014, Ian Good fellow and his colleagues created the generative adversarial network (GAN), a family of machine learning frameworks. Based on minimax game underpins Generative Adversarial Networks. Whereas the first model, known as the Generator, generates future frames directly, the second model, known as the Discriminator, tries to discriminate between samples pulled from the training data and samples generated from the Generator model. This technique learns to generate additional data with the same statistics as the training set given a training set. Training via the discriminator, which is dynamically updated as well. This essentially implies that the generator is taught to deceive the discriminator rather than reduce the distance to a certain picture. This allows the model to learn without being supervised. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully learning, and reinforcement learning. The essential concept of a GAN is "indirect".

Candidates are generated by the generative network, which is then evaluated by the discriminative network. The competition is based on data distributions. Typically, the generating network learns to map from a latent space to a data distribution of interest, whereas the discriminative network separates the generator's candidates from the real data distribution. The goal of generative network training is to improve the discriminative network's error rate.

The discriminator's first training data is a well-known dataset. It is trained by feeding it samples from the training dataset until it achieves an acceptable level of accuracy. The generator develops its skills dependent on whether it can mislead the discriminator. The generator is usually seeded with randomized input from a predetermined latent space (e.g., a multivariate normal distribution).
Following that, the discriminator evaluates the candidates created by

the generator. Both networks are subjected to independent back propagation techniques for the generator to create better samples and the discriminator to grow more adept at detecting fake samples. The discriminator is usually a convolutional neural network, while the generator is usually a DE convolutional neural network when used for image generation. When GANs fail to generalize properly, they frequently suffer from "mode collapse," missing whole modes from the input data. A GAN trained on the MNIST dataset, which contains several examples of each digit, may timidly delete a portion of the digits from its output, for example. Some researchers believe the main cause is a bad discriminative network that misses the pattern of omission, while others blame a weak objective function choice.

2.2.1 Types of GAN

2.2.1.1 The conditional generative adversarial network (CGAN)

is a machine learning method for training generative models that is an extension of the generative adversarial network (GAN). Mehdi Mirza and Simon Osindero initially proposed the notion in a 2014 study titled Conditional Generative Adversarial Nets.

CGAN is a deep learning approach that uses a conditional setup, which means that the generator and discriminator are both conditioned on auxiliary input such as class labels or data from other modalities. As a consequence, by feeding it varied contextual information, the ideal model may learn multi-modal mapping from inputs to outputs.

2.2.1.1.1 Applications of CGANs

There is a wide-ranging suite of possible applications for CGANs.

- Image-to-image translation - CGANs were demonstrated to be effective at synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images, among other tasks. This led to the development of CGAN-based software, pix2pixHD.
- Text-to-image synthesis - an experimental TensorFlow implementation of synthesizing images that builds on top of the implementation of TensorFlow / Tensor Layer DCGANs (deep convolutional Generative Adversarial Networks).
- Video generation - deep neural network that can predict the future frames in a natural video sequence.
- Convolutional face generation - CGANs use to generate faces withspecific attributes from nothing but random noise.
- Generating shadow maps - introduced an additional sensitivity parameter to the generator that effectively parameterized the loss of the trained detector, proving more efficient than previous state-of-the-art methods.
- Diversity-sensitive computer vision tasks - explicitly regularizes the generator to produce diverse outputs depending on latent codes, with demonstrated effectiveness on three CGAN tasks: image-to-image translation, image inpainting, and video prediction generation.

2.2.1.2 Bidirectional GAN

Inverse methods like Bidirectional GAN (BiGAN) and Adversarial Auto encoders learn a mapping from data to the latent space, like how the normal GAN model learns a mapping from a latent space to the data distribution. Like the encoder of a variational auto encoder, this inverse mapping allows actual or produced data instances to be projected back into latent space. Semi-supervised learning, interpretablemachine learning, and neural machine translation are just a few of the applications of bidirectional models.

2.2.1.3 Retrospective CycleGAN

Retrospective CycleGAN is a strategy for automatically training image-to-image translation models without using paired samples. The models are trained unsupervised with a set of photos from the source and target domains that are not connected in any way.

2.2.1.3.1 Applications on Retrospective CycleGAN

Style Transfer, Object Transfiguration, Season Transfer, Photograph Generation from Paintings, Photograph Enhancement and Future frame prediction.

2.2.1.3.2 Objective function

In the training of the Retrospective cycle GAN, the objective function consists of four loss functions divided into two categories: two reconstruction losses and two adversarial losses. The objective function also contains three non-zero weights loss functions (λ1, λ2, and λ3) in different parts used for balancing the train process.

$$L = L_{\text{image}} + \lambda_1 L_{LoG} + \lambda_2 L_{\text{adv}}^{\text{frame}} + \lambda_3 L_{\text{adv}}^{\text{seq}}$$

Reconstruction Losses:

One is the error between two images. And the others are for error between images as well but after applying gaussian filter.

$$L_{\text{image}} + \lambda_1 L_{LoG}$$

Adversarial Losses:

$$\lambda_2 L_{\text{adv}}^{\text{frame}} + \lambda_3 L_{\text{adv}}^{\text{seq}}$$

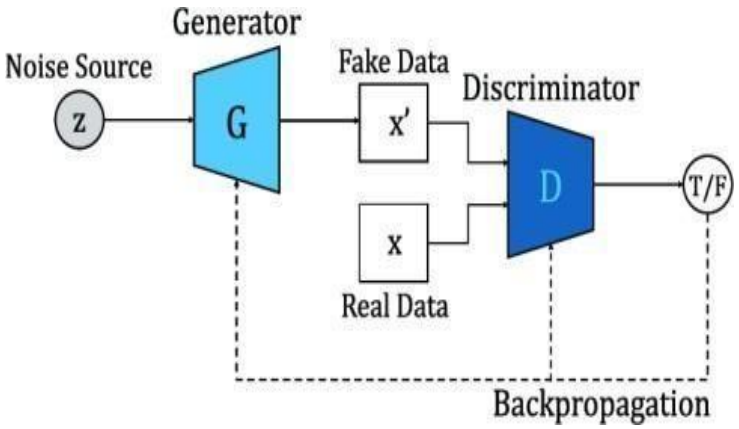
Adversarial Loss on frames

$$l_{\text{A}}(p, q) = \max_{G} \min_{D_A} [(D_A(q) - 1)^2 + (D_A(G(p)))^2].$$

Adversarial Loss on sequences

$$l_{\text{B}}(p, r) = \max_{G} \min_{D_B} [(D_B(r) - 1)^2 + (D_B(G_c(p)))^2].$$

2.2.2 Basic architecture of Generative adversarial networks



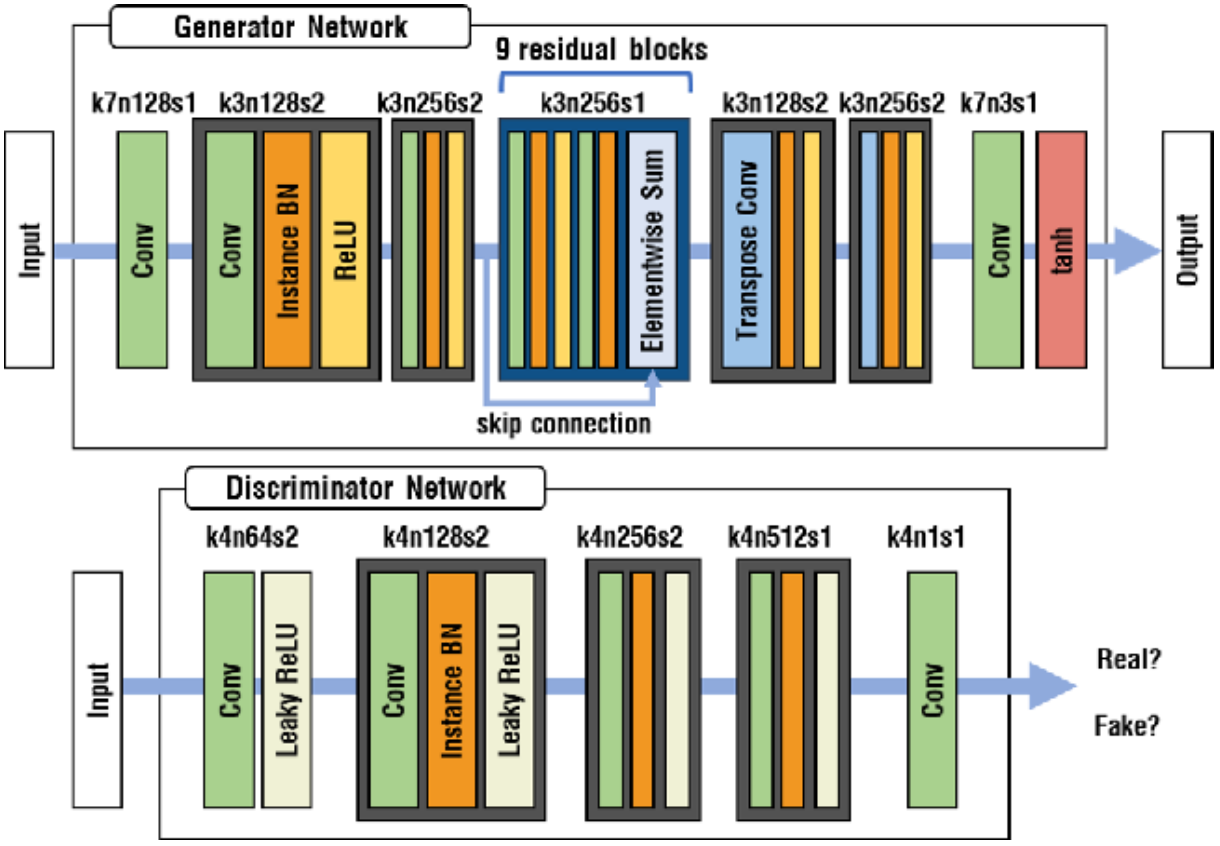


Figure 2.2: GAN

The generator and discriminator networks architecture are illustrated in the figure. The generator network architecture consists of 4 convolution layers, 9 residual blocks, and 2 transpose convolution layers. The discriminator network architecture consists of 5 convolution layers with leaky rectified linear units. The two discriminators (frame and sequence) have the same network architecture except the number of input images.

2.3 Convolutional neural networks

2.3.1 What are neural networks?

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

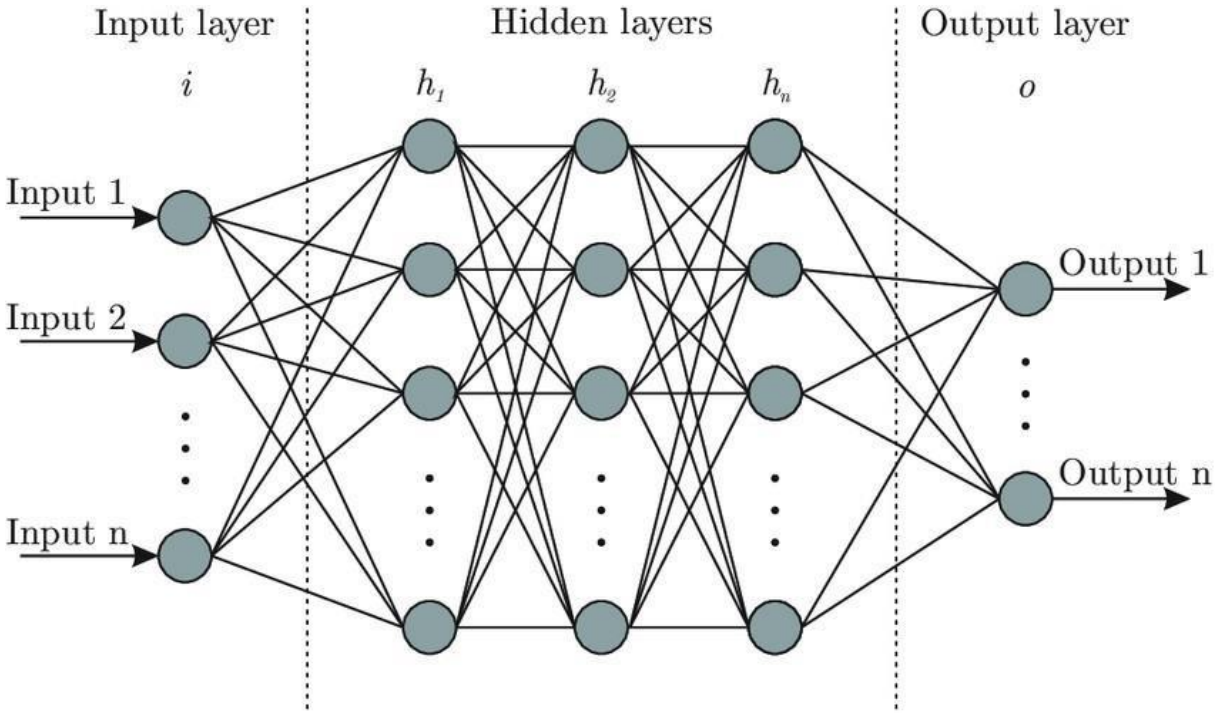


Figure 2.3: artificial neural networks

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts.

Artificial Neural Networks are used in various classification tasks like image, audio, words. Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network.

In a regular Neural Network, there are three types of layers:

-Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).

-Hidden Layer**:** The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.

-Output Layer**:** The output from the hidden layer is then fed into a logistic function like sigmoid or SoftMax which converts the output of each class into the probability score of each class.

The data is then fed into the model and output from each layer is obtained this step is called feed forward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. After that, we back propagate into the model by calculating the derivatives. This step is called Back propagation which basically is used to minimize the loss.

A Convolutional Neural Network or CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. Deep Learning thus recognizes objects in an image by using a CNN.

Now let's talk about a bit of mathematics that is involved in the whole convolution process:

- Convolution layers consist of a set of learnable filters, every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input).

- For example, if we must run convolution on an image with dimension 34x34x3. The possible size of filters can be axax3, where 'a' can be 3, 5, 7, etc. but small as compared to image dimension.

- During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.

- As we slide our filters, we'll get a 2-D output for each filter and we'll stack them together and as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

2.3.2 Layers in Convolutional Neural Network

A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

2.3.2.1 Types of layers

-Input Layer: This layer holds the raw input of the image.

-Convolution Layer: This layer computes the output volume by computing the dot product between all filters and image patches.

-Activation Function Layer: This layer will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max (0, x), Sigmoid: 1/(1+e^-x),Tanh, Leaky RELU, etc.

-Pool Layer: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents over fitting. Two common types of pooling layers are max pooling and average pooling.

-Fully-Connected Layer: This layer is a regular neural network layer that takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.

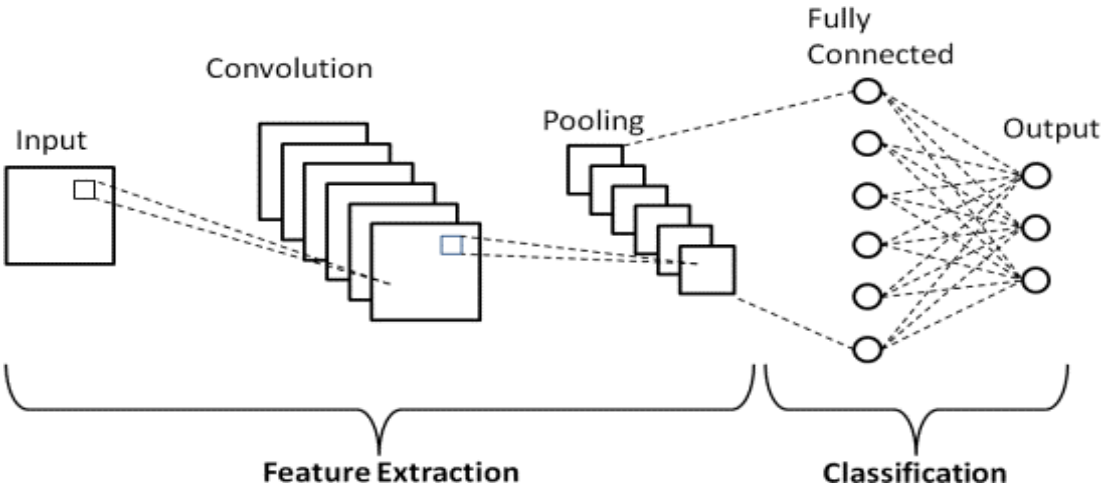


Figure 2.4: CNN

Chapter Three
Related work

In computer vision, predicting anomalous actions is a difficult challenge. For starters, there is no universal definition for anomalous phenomena. The concept of anomalous behavior changes depending on the situation and context. While riding a bike on the street is considered a typical activity, it may be classified as aberrant if it occurs in pedestrian paths. One typical strategy to dealing with this problem is to regard the unseen events as anomalous. However, gathering all typical events is impossible, and there will always be unseen scenes containing normal activity. As a result, the second issue is a lack of labelled data. Detecting and classifying such events is a time-consuming process. Furthermore, dealing with video input is tough since the issue space is a high dimensionless space with a huge number of redundant information, making it difficult to obtain a meaningful feature that can capture relevant information. Another issue is that some abnormal actions are undetectable at a low timescale, which is typical of frame level computation. In this section, they discuss several methods for detecting and predicting anomalous events in videos.

Hegde et al. [1] describe the problem of vehicle trajectory predictionas soon, transportation autonomy will be ubiquitous, and trajectory prediction will aid decision-making. They develop data-driven vehicle trajectory prediction model using Generative Adversarial Networks (GANS). They use Bilateral filtering and Adaptive Gaussian thresholding in preprocessing phase and to extract features they use the feature extractor long short-term memory (LSTM). YOLOV3 algorithm is used in object detection and tracking phase. In the final phase: trajectory prediction, they predicted it using GAN. The VisDrone dataset and the TRAF dataset were used in this study. The Average Displacement Error (ADE) and The Final Displacement Error (FDE) are the evaluating metrics used. When compared to current trajectory

prediction methods such as linear regressor, LSTM, Social LSTM, and others, the displacement errors achieved using GAN are significantly lower.

Han et al. [2] discussed the problem of diminishing gradients plagues recurrent neural networks (RNNs), making it difficult to learn lengthy data sequences. The update gate and the reset gate are used by gated to determine which hidden state information recurrent unit (GRU) should be updated (passed) or the hidden state reset. Time-based backpropagation algorithms, Adam gradient descent parameter optimization methods, the architectures of the RNN, long short-term memory (LSTM), and GRU are some of the theories and models linked to neural networks. From prediction methods: 1-GRU based short-term real-time trajectory prediction ,2- data preprocessing and 3-training the GRU neural network. The used data set is the historical flight data. The three-dimensionalcoordinate point error between the anticipated and true values is calculated quantitatively using the root mean square error (RMSE) method (evaluating metrics). The RMSE error value of the autoregressive moving average model (ARIMA) and Holt-Winter predictions is higher than that of the neural network prediction, and the real-time short-term trajectory prediction based on GRU neural network is the best of these prediction models.

Altché et al. [3] looked at the challenge of utilizing previously seen using data to forecast future trajectories of cars travelling on a highway long short-term memory (LSTM); these predictions may then be used to plan the movements of an autonomous vehicle. In the feature extraction phase, they aim at only using features which can be reasonably easily measured using on-board sensors such as GNSS and LiDAR, barring range in the data preparation phase they used a first order or occlusion issues. Savitzky-Golay filter which performs well for signal differentiation. In the learning model they choose to utilize (LSTM) network as our artificial neural network. LSTMs are a type of recurrent neural network (RNN) that is particularly well suited for time series analysis. They used the Next Generation Simulation (NGSIM) dataset. This network was proven to outperform the prior state-of-the-art, with

an average root mean square error (RMSE) error of about 70 cm for the lateral location 10 seconds in the future and less than 3ms1 for longitudinal velocity withthe same horizon.

Xiao et al. [4] discussed the various factors that affected on High-resolution remote-sensing images that are widely used for object detection. This paper introduces a new rotationally invariant object detection descriptor that can help with object recognition problems caused by diverse object rotations. To establish rotational invariance of the histogram of directed gradients, they use orientation normalization, feature space mapping, and an elliptic Fourier transform. The results show that the suggested approach detects objects with good rotational invariance. They used support vector machine (SVM) to train the classifier and detect object. Most of the datasets for car detection were gathered from parking lots. They selected parking lots from major cities across the world, including New York and Tokyo. To evaluate the detection precision for aircraft identification, they use three alternative classifiers: radial-basis-function-based support vector machines (RBF-SVMs), linear-SVMs, and the adaptive boosting (AdaBoost) classifier. RBF-SVM has the highest accuracy 81.99%.

Khosroshahi et al. [5] aimed to achieve safety in autonomous driving.Driving a car necessitates interaction with other road users. To navigate safely, an intelligent driving system must comprehend and predict the actions of its surrounding agents. They use a Recurrent Neural Network (RNN) and 3D trajectory cues to create a framework for automatically classifying the activities of nearby on-road actors. In the phase of feature extraction, they followed 3 steps: 1-linear changes, 2-angular changes, 3- angular changes histogram. In prediction model: RNN can use contextual temporal information for mapping input sequences to output sequences. The KITTI benchmark was used for training and testing. When compared to S = 2, classifiers with a three-layer architecture (S = 3) exhibited better accuracy. However, at S = 4, adding additional layer does not appear to increase accuracy substantially.

Shirazi et al. [6] described the problem of vehicle trajectory turning at intersections. To monitor intersection videos and collect vehicle trajectories with labels known as turning movements, a vision-based tracking system is being developed. Deep neural networks (DNN) are further investigated on a third intersection with different video settings. The models used in this paper are DNN and long short-term memory (LSTM) networks. The obtained models are then used to forecast future vehicle track movements and classify turning predictions. The trajectory collection system based on detection and tracking. from trajectory prediction methods: Vehicle kinematic, Ridge regression, LSTM, DNN. The developed tracking system collects vehicle trajectories of two intersection videos with rate of 15 frames per seconds (FPS) and 2-h time. The effectiveness of long short-term memory networks in early predicting turning motions with more than 92 % accuracy is demonstrated by the future 2 s evaluation of trajectories.

Tung et al. [7] discussed the problem of information overload which makes manual surveillance increasingly difficult. A human operator must continually examine a vast array of video streams for suspicious behavior in a typical surveillance system. An intelligent vision-based surveillance system's goal is to automate surveillance's monitoring and event detection components, only notifying the operator when odd The proposed behavior or other events of interest are observed. methods in this paper are: 1-learning the scene structure, 2- learning a region transition model, 3- classifying trajectories in progress. the used data sets: 1-Dee and Hogg's carpark dataset, 2-Naftel and Khalid's laboratory dataset, 3-Piciarelli et al.'s synthetic dataset the performance of the classifier depends on the completeness of the training set.

Zhang et al. [8] described the problem of traffic accidents as a concrete manifestation of road traffic safety levels, and it can be applied in the traffic management system for scientific decision-making. They proposed a prediction model for traffic accident that combines long

short-term memory (LSTM), and gradient boosted regression trees (GBRT). The LSTM neural network is used to extract the features with time-dependent information. These features are trained by the GBRT model to predict traffic accidents. The used boosting ensemble-learning framework is a technical framework that combines multiple different models to perform the corresponding tasks to achieve more efficient and accurate results. After training, the root mean square logarithmic error (RMSLE) and R-square (statistical measure of fitting) are the evaluating metrics used to measure performance. Compared with the traditional regression model like the traditional BP neural network model, the LSTM neural network model, and the GBRT model, the experimental results show that the LSTM-GBRT model has the strongest ability to fit the data and the variable has the best predictability.

Mansoor et al. [9] participate in problem solving of road traffic crashes. The persons in a crash may not receive ideal treatment due to their injury severity at an early stage, which cause many victims. A two-layer ensemble machine learning model is proposed to predict road traffic crash severity. The first layer integrates four base machine learning models: k-nearest neighbor, decision tree, adaptive boosting, and support vector machine. Whereas the second layer classifies the crash severity based on the feed forward neural network (FNN) model. The K-nearest neighbor (KNN) is firstly chosen if the information about data distribution is very little or none. The goal of decision tree (DT) model is to predict a target value based on many input features. The main purpose of adaptive boosting (AdaBoost) is to train multiple weak learners using the same training dataset and then construct a strong learner by grouping the weak learners. The main aim of the support vector machine (SVM) technique is to find the best hyper plane, which maximizes the margin between support vectors. A feed forward neural network (FNN) was eventually developed to predict crash severity. Classification by FNN is an iterative process for adjusting weights and bias based on the provided information. The two-layer ensemble model outperformed all other base models, with an accuracy of 79.3% and F1 scores of 0.78 and 0.80 for fatal and non-fatal crashes.

Wang et al. [10] explain the problem of action recognition, which attempts to classify different human actions in videos. The current main-stream methods generally utilize ImageNet-pretrained model as features extractor, which it is not the optimal choice to pertain a model for classifying videos on a huge image dataset. By comparing recurrent neural network (RNN) and long short-term memory (LSTM), it is found the LSTM can address the tough issue of gradient vanishing and alleviate gradient explosion problem to some extent. For that, LSTM is more extensively applied in sequence modeling issue. In this model, they use Inception 3D CNN as the feature extractor, which pertained on Kinetics dataset I3D. Then, the output feature vectors of I3D are fed into LSTM network to get high-level temporal features. The model performance on the UCF-101 dataset is more efficient and advanced than some other has reached to main-stream methods like (iDT, C3D, TwoStream,….) 94.3%.

Salahadin Seid Yassin et al. [11] developed a hybrid K-means and random forest (RF) approaches, to get the most determinant road accident variables. K-means extract hidden information from road accident data and creates a new feature in the training set. RF employed to classify severity prediction. After comparing with other classification techniques, the proposed approach disclosed an accuracy of 99.86%.

Ronneberger et al. [12] they presented a network and training strategy that relies on the strong use of data augmentation to use the available annotated samples more efficiently. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. They demonstrate the application of the u-net can be applied at different segmentation tasks, like the segmentation of neuronal structures in electron microscopic recordings task and cell segmentation task in light microscopic image. They explained that the u-net architecture achieves very good performance on very different biomedical segmentation applications. Thanks to data augmentation with elastic deformations, it only needs very few annotated images and has a very reasonable training time of only 10

They proved that the u-net hours on a NVidia Titan GPU (6 GB). architecture can be applied easily to many more tasks.

Kwon et al. [13] describe the problem of generating future frames by giving a set of sequential frames. The retrospective cycle constraints generative adversarial network (Cycle GAN) is used to predict accurate and clear future frames over time. Cycle GAN is simultaneously training a generator network to generate fake frames and a discriminator networkto distinguishing that fake frames. In this proposed model through a series of experiments and empirical evaluations on multiple datasets, itsresults outperform the baseline methods. The model accuracy on [KITTI
, Caltech pedestrian] (91%), UCF101 (94%), CUHK Avenue (98%) and ShanghaiTech (97%) datasets.

Carreira et al. [14] introduce a new two-stream inflated 3D convNet (I3D) model that is based on 2D convNet inflation. The model is evaluated upon the new Kinetics Human Action Video dataset. This dataset has 400 human action classes and over 400 clips per class, which is collected from YouTube videos. After the I3D model is pre-trained on Kinetics, action classification reaches 97.9% on UCF-101 dataset and 80.2% on HMDB-51 dataset, which outperforms the state-of-the-art in action classification.

Kaur et al. [15] describe the problem of predicting future frames of a video sequence. Video prediction is the task of predicting subsequent frames, given a sequence of video frames. By combining the strengths of the two approaches used in “Modified Stochastic Adversarial Video Prediction” and “Eidetic 3D LSTM". The architecture is modified so that it can predict future for more time stamps and alleviate mode collapse problem. The datasets used in this method and their accuracy are UCF101 Dataset (89%), Moving MNIST Dataset (86%) and Penn Action Dataset (95%).

Isola et al. [16] describe the problem of translation to image-to-image. This paper, exploring that GANs in the conditional setting. Just as GANs learn a generative model of data, conditional GANs (C-GANs) learn a conditional generative model. This makes C-GANs suitable for image-to-image translation tasks. Training a conditional GAN to map edges➔photo. The discriminator [D] learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator [G] learns to fool the discriminator. Details of training on each of these datasets are provided in the supplemental materials online. architecture for semantic segmentation, and train it on the cityscapes dataset. investigating if C-GANs achieve this effect on the Cityscapes (87%) dataset.

Rossi et al. [17] propose and compare Deep Learning models based on Long Short-Term Memory and Generative Adversarial Network architectures. Predicting trajectories starting from Floating Car Data (FCD) is a complex task that comes with different challenges, namely Vehicle to Infrastructure (V2I) interaction, Vehicle to Vehicle (V2V) interaction, multimodality, and generalizability. Multimodality and generalizability have been neglected the most, and this work attempts to fill this gap by proposing and defining new datasets, metrics, and methods to help understand and predict vehicle trajectories. Experiments have been conducted using newly collected datasets in four large Italian cities (Rome, Milan, Naples, and Turin). The results prove that, although LSTM-based models are superior in unimodalscenarios, generative models perform best in those where the effects of multimodality are higher.

Suzuki et al. [18] have extended the well-known Laws' texture energy approach to handle 3D solid textures. The extended 3D Laws' convolution masks make it possible to analyze 3D solid texture databases. Although there are many benchmark datasets which are available for 2D textures, it is difficult to find benchmark datasets for 3D solid textures. Several software program modules were implemented to test the three-dimensional Laws masks. The software programs modules

include (1) 3D solid texture synthesis, (2) 3D solid texture convolution, (3) 3D solid texture retrieval and (4) 3D solid texture classification. In the 3D solid texture syntheses experiment the program was implemented by C language. It took over 50 hours to synthesize the 1200 solid textures (32×32×32), In 3D solid texture retrieval experiment there are 27 shape features for the three dimensional Laws' masks of lengths three, and 125 for length five and in 3D solid texture classification experiment the masks of length three (3×3×3) matched 45 labels correctly (90.0 percent), and the masks of length five (5×5×5) matched 44 labels correctly (88.0 percent).

Vu et al. [19] present 3D convolutional neural network (3D-CNN) as anend-to-end model to label a target task among four sensorimotor tasks for each functional magnetic resonance imaging (fMRI) volume. The sensorimotor dataset that was used for evaluating the fcDNN was also applied to their 3D-CNN model. 3D-CNN with ReLU activation function showed the lowest average error rate. In terms of the comparison between the ReLU and Tanh, the ReLU activation function in the fcDNN yielded a significantly low error rate compared to the Tanh activation function, Using the 3D-CNN, ReLU activation function resulted in a markedly lower averaged error rate than Tanh activation function.

AYUSH MUDGAL et al. [20] describe the problem of accident detection, they used HRNN method, and this method uses two layers of long short-term memory neural networks. The first neural network (NN) is a recurrent network that analyzes the time-dependent sequence of the images within each video. The second takes the encoding of the first NN and builds a second NN that reflects which videos contain accidents and which do not. The resulting model enables a prediction of whether new dash cam footage has an accident, they use the CADP dataset for videos containing accidents and the DETRAC dataset, which is originally made for object detection of vehicles, as the videos does not contain accidents. To expand their dataset, they will also search the internet for videos containing cases of accidents. For the final dataset, they had 256 videos with car, bus, bike etc. accidents recorded in the CCTV camera at

the corners of the street, the final accident forecasting model can predict accidents about 2 seconds before they happen with an accuracy of 80%.

Liuwen et al. [21] describe the problem of anomaly detection. Introducing a new idea in this field. They predict the normal events and compare them with the ground truth, and if we find a difference between the predicted frames and its ground truth that will be classified as anomalies. They have used some techniques to achieve this proposed method. The model consists of U Net with conditional GAN to predict realistic frames. Then by using measurements of the prediction frames like peak signal to noise ratio 'PSNR'. They also used different datasets CUHK Avenue, UCSD dataset and ShanghaiTech dataset. And they used the ROC curve to calculate the accuracy of the model by calculating the area under the curve 'AUC'. And the proposed method reached 81.8% with UCSD Ped 1 and 93.5% with UCSD Ped 2. They also made an evaluation with the Toy dataset and AUC reached 98.9%.

Farley et al. [22] describe the problem of generative modeling problem. Generative adversarial networks are a kind of artificial intelligence algorithm designed to solve the generative modeling problem. The goal of a generative model is to study a collection of training examples and learn the probability distribution that generated them. Generative Adversarial Networks (GANs) are then able to generate more examples from the estimated probability distribution. Generative models based on deep learning are common, but GANs are among the most successful generative models (especially in terms of their ability to generate realistic high-resolution images). GANs have been successfully applied to a wide variety of tasks (mostly in research settings) but continue to present unique challenges and research opportunities because they are based on game theory while most other approaches to generative modeling are based on optimization.

Zhao et al. [23] describe the problem of Detection and tracking of

pedestrians and vehicles using roadside LiDAR sensors. In this study, the LiDAR data collected from the test site were processed in the order of the following steps: background filtering, clustering, object classification, and tracking of movements. Background filtering is a prerequisite but rather independent process, the researchers have developed models prior to reaching the current stage of work. After the background filtering process, an intermediate process was developed to select the region of interest (ROI) so that the focus of the following steps can be placed on the selected ROIs such as a road segment or an intersection. The focus of this study is therefore placed on the clustering algorithm, identification of pedestrians and vehicles, tracking, and the examination of the effectiveness of the integrated approach. As can be seen, all movements including the through/left-turn/right-turn movements of vehicles, and the crossing movement of pedestrians were clearly tracked. Approximately an accuracy of 95% can be reached within about 30 m detection range (in one direction) from the LiDAR sensor, and failure was caused by the same issues such as occlusion and distance.

Narayanan et al. [24] describe the problem of anomaly Detection System for Vehicles. The model consists of three phases, first one is data Collection Phase: This is the first step in which the stream of CAN bus data is collected from real vehicles for analysis. They employed the OBD-II port present in most vehicles for this purpose. Second one is model Generation Phase: In this phase they analyze the collected data and generate a model. Since Hidden Markov Models (HMM) can abstract the time series data, they use them to model this scenario. Third one is anomaly Detection Phase: This is the final phase in which they detect anomalous behaviors using generated model and posterior probabilities. The model can be integrated with all current and future car systems as a plug-n-play device or as a system module programmed on the on-board car computer. They also can add their system to old cars using their pre-existing OBD port and attaching a small raspberry-pie chip to the OBD port to collect, analyze, and issue alerts. New cars can have this feature preinstalled.

Sultani et al. [25] describe the problem of anomaly detection in

surveillance videos. They propose a deep learning approach to detect real world anomalies in surveillance videos. They learn a general model of anomaly detection using a deep MIL framework with weakly labeled data. The multiple instance learning “MIL” classifies the videos as anomaly “positive bags” or normal “negative bags”. They also used different datasets like: UMN dataset, UCSD Ped1 and Ped2 datasets, Avenue dataset, Subway Exit and Subway Entrance datasets, BOSS dataset is collected from a surveillance camera mounted in a train, Abnormal Crowd dataset and they also collected a large-scale dataset. For Measuring the accuracy, they calculated the area under the ROC curve ‘AUC ‘and they reached 75.41 with a remarkable record according to the previous work at the same idea.

Swathy et al. [26] introduced a survey on vehicle detection and tracking techniques in video surveillance. In this paper, presents different vehicle tracking and detection techniques on surveillance videos. Although there are many sensors, it possesses high maintenance cost. They talked about the intelligent transportation system ‘ITS’. And they described the elements of traffic analysis system first one is object-based approach, background subtraction which they talked about four types of its types: Gaussian mixture model ‘GMM’, adaptive multi_cue background subtraction, averaging and kalman filter. And two more vehicle detection methods: feature-based methods and motion-based methods. Second section of elements of the traffic analysis system is vehicle tracking methods which are: Region-Based Tracking Methods, Contour Tracking Methods, 3D Model-Based Tracking Methods, Feature-Based Tracking Methods and Color and Pattern-Based Methods.

Ganokratanaaet al. [27] describe the problem of anomaly event detection using GANs for surveillance videos. Anomalous event detection is advantageous for real-time video surveillance systems in

terms of safety and security. Current works mostly run offline and struggle with abnormal event detection in crowded scenes. We propose unsupervised anomaly event detection using Generative Adversarial Network (GAN) with Optical Flow to obtain spatiotemporal features in appearance and motion representations. In training, GAN is used to train only the normal event images to generate their corresponding optical flow image. Hence, in testing, since the model knows only the normal patterns, any unknown events are considered as the anomaly event which can be detected by subtracting the pixels between the generated and the real optical flow images. We implement on the publicly available benchmark datasets and compare with state-of-the-art methods. Experiment results show that our model is effective for anomaly event detection in real-time surveillance videos. UCSD ped1 and ped2 datasets and the UMN dataset. For Measuring the accuracy, they calculated the area under the ROC curve 'AUC 'and they reached 0.99 on UMN dataset with a remarkable record according to the previous work at the same idea.

Weber et al. [28] describe the problem of anomaly detection for automotive CAN communication. Due to the steadily increasing connectivity combined with the trend towards autonomous driving, cyber security is essential for future vehicles. The implementation of an intrusion detection system (IDS) can be one building block in a security architecture. Since the electric and electronic (E/E) subsystem of a vehicle is static, the usage of anomaly detection mechanisms within an IDS is promising. This paper introduces a hybrid anomaly detection system for embedded electronic control units (ECU), which combines the advantages of an efficient specification-based system withthe advanced detection measures provided by machine learning. The system is presented for - but not limited to - the detection of anomalies in automotive Controller Area Network (CAN) communication. The second part of this paper focuses on the machine learning aspect of the proposed system. The usage of Lightweight On-line Detector of Anomalies is investigated in more detail. After introducing its working principle, the application of this algorithm on the time series of a CAN communication signal is presented. To evaluate the performance of the proposed system, a synthetic CAN signal is used. The signal sequence as well as the corresponding CAN messages are generated with CANoe

from Vector Informatics, which is a simulation and analysis tool for automotive networks.

Dogru et al. [29] describe the problem of traffic accident detection. This paper presents an intelligent traffic accident detection system in which vehicles exchange their microscopic vehicle variables with each other. The proposed system uses simulated data collected from vehicular ad-hoc networks (VANETs) based on the speeds and coordinates of the vehicles and then, it sends traffic alerts to the drivers. Furthermore, it shows how machine learning methods can be exploited to detect accidents on freeways in ITS. It is shown that if position and velocity values of every vehicle are given, vehicles' behavior could be analyzed, and accidents can be detected easily. Supervised machine learning algorithms such as Artificial Neural Networks (ANN), Support Vector Machine (SVM), and Random Forests (RF) are implemented on traffic data to develop a model to distinguish accident cases from normal cases. The performance of the RF algorithm, in terms of its accuracy, was found superior to ANN and SVM algorithms. RF algorithm has shown better performance with 91.56% accuracy than SVM with 88.71% and ANN with 90.02% accuracy.

Yipeng et al. [30] Accurate short-term traffic flow forecast may give quick and accurate traffic condition information, which can assist in making travel decisions and reducing traffic congestion. Deep learning (DL) is a new paradigm for analyzing massive data generated by everyday transportation in cities. They offer a revolutionary end-to-end deep learning architecture made up of two modules in this study. They created a Conv-LSTM module that can extract the spatial-temporal information of traffic flow data by combining convolution with LSTM. In addition, a Bi-directional LSTM module is used to evaluate past traffic flow data from the forecast point to get the traffic flow periodicity characteristic. The experimental findings on a real dataset indicate that the suggested technique outperforms existing approachesin terms of prediction accuracy.

Chapter Four

Proposed model

This chapter describe the design of the system, tools, hardware, evaluation metrics and the datasets used in our system.

4.1 High level design of the implemented system

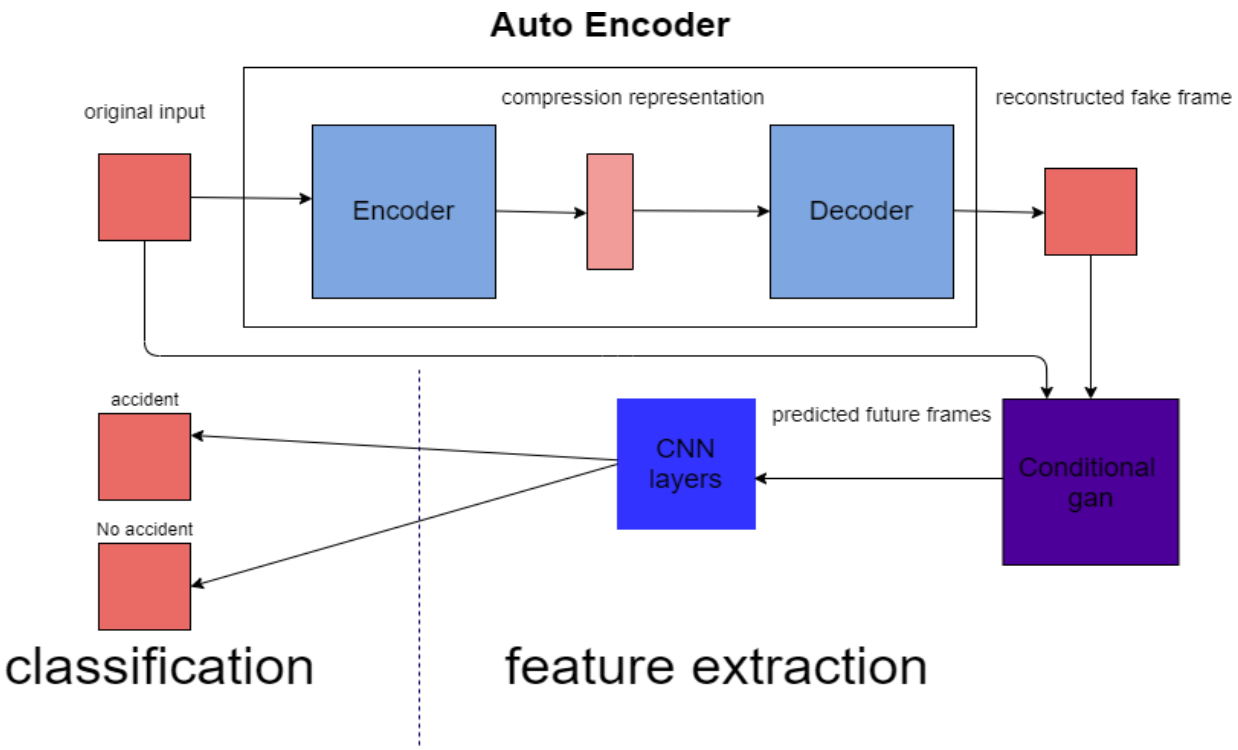


Figure 4.1: high level design of the system

4.1.1 Encoder -> input: original photo , output : compression representation

12 layers

-CONV2D: (3 layers) creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

- No. of filters used: 128 in conv1, conv2

-256 in conv3

-Kernel size: (7,7) in conv1
-(3,3) in conv2, conv3

-MaxPooling2D: (3 layers) Down samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool size) for each channel of the input. The window is shifted by strides along each dimension.

-Pool size: (2,2)

-Batch Normalization: (3 layers) applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

-Leaky Relu: (3 layers) version of a Rectified Linear Unit
It allows a small gradient when the unit is not active

-Alpha: 0.2

4.1.2 Decoder-> input: compression representation, output : fake frame

-layers

-CONV2D: (4 layers) creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

-No. of filters used: 256 in conv1, conv2
-128 in conv3
-1 in conv4

-Kernel size: (3,3) in conv1, conv2, conv3
-(7,7) in conv4

-Batch Normalization: (5 layers) applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

-UpSampling2D: (3 layers) Repeats the rows and columns of the data by size [0] and size [1] respectively.

-Up sampling factors for rows and columns: (2,2)

-Leaky Relu: (2 layers) version of a Rectified Linear Unit It allows a small gradient when the unit is not active

-Alpha: 0.2

4.1.3 conditional Gan-> input: original frame + fake frame, output: predicted frame

4.1.3.1 Discriminator:

8 layers:

-CONV2D: (4 layers) creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

-No. of filters used: 64 in conv1

-128 in conv2

-256 in conv3

-4 in conv4

-Kernel size: (4,4) in conv1, conv2, conv3

-(1,1) in conv4

-MaxPooling2D: (3 layers) Down samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window

(Of size defined by pool size) for each channel of the input. The window is shifted by strides along each dimension.

-Pool size: (2,2)

-Dense: (1 layer) implements the
operation: output = activation (dot (input, kernel)
+ bias) where activation is the element-wise
activation function passed as
the activation argument, kernel is a weights matrix
created by the layer, and bias is a bias vector created by the layer (only applicable if use bias is True). These are all attributes of Dense.

- Units: (1)

4.1.3.2 Generator:

8 layers

-CONV2D: (1 layer) creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

-No. of filters used: 1 in conv1

-Kernel size: (7,7) in conv1

-Dense: (1 layer) implements the
operation: output = activation (dot (input, kernel)
+ bias) where activation is the element-wise
activation function passed as
the activation argument, kernel is a weights matrix
created by the layer, and bias is a bias vector created by the layer (only applicable if use bias is True). These are all attributes of Dense.

-Kernel initializer: (7,7)

-Leaky Relu: (3 layers) version of a Rectified Linear Unit It allows a small gradient when the unit is not active

-Alpha: 0.2

-Conv2DTranspose: (2 layers) use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.

-No. of filters used: 128 in both

-Kernel size: (4,4)

-Reshape: (1 layer) reshapes inputs into the given shape.

-Target shape: (7,7)

4.1.4 CNN layers -> input: predicted frames, output : normal or accident

13 layers:

-CONV2D: (4 layers) creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

-No. of filters used: 16 in conv1

-32in conv2

-64 in conv3, conv4

-Kernel size: (3,3) in conv1, conv2, conv3, conv4

-MaxPooling2D: (4 layers) Down samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window

(Of size defined by pool size) for each channel of the input. The window is shifted by strides along each dimension.

-Pool size: (2,2)

-Dense: (2 layer) implements the operation: output = activation (dot (input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use bias is True). These are all attributes of Dense.

-Units: (128) in dense1

-(1) in dense2

-Dropout: (1 layer) randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/ (1 - rate) such that the sum over all inputs is unchanged.

-Rate: 0.5

-Flatten: (1 layer) used to make the multidimensional input one-dimensional, commonly used in the transition from the convolution layer to the full connected layer.

4.2 Tools and hardware

4.2.1 Tools

- Google Colab, TensorFlow, OpenCV, Numpy, Keras, Scikit-image, Pickle, Matplotlib, Imageio, SciPy

4.2.2 hardware

Google Colab Hardware specifications: -

Parameter	Google Colab
GPU	Nvidia K80 / T4
GPU Memory	12GB / 16GB
GPU Memory Clock	0.82GHz / 1.59GHz
Performance	4.1 TFLOPS / 8.1 TFLOPS

Table 4.1: Google Colab hardware

4.2 evaluation

4.3.2 Mean squared error (MSE)

It is the most traditional and simple method for image quality evaluation. The main idea is to compute the squared difference between the original image and ground truth comparing pixel by pixel, then calculating the average. The lower values closer to zero are better for MSE.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

All images with the same MSE does not mean that all images contain the same noise and distortions. So, MSE doesnot consider the structural similarity of the generated images and poorly correlated to visual perception.

4.3.3 Structural Similarity Index Measure

Structural Similarity Index Measure (SSIM) is a perception-based evaluation metrics considering any change in the perception in structural similarity in images as a distortion. The main purpose of SSIM is to extract structural information from images. SSIM gets information parameters from the image as luminance contrast, and structure.

Luminance is parameter shows that distortion of the generated image is less visible in the edges. Contrast is parameter shows that distortion of the generated image is less visible in the texture. Finally, the structure is computed using the other parameters. The higher the SSIM is the better in the quality of the generated image.

4.3 Datasets

4.4.2 Car Crash Dataset “CCD”

Car Crash Dataset (CCD) is collected for traffic accident analysis. It contains real traffic accident videos captured by dash cam mounted on driving vehicles, which is critical to developing safety-guaranteed self-driving systems. CCD is distinguished from existing datasets for diversified accident annotations, including environmental attributes (day/night, snowy/rainy/good weather conditions), whether ego-vehicles involved, accident participants, and accident reason descriptions.

They collected traffic accident videos from YouTube channels and split them to get 1,500 trimmed videos. Each video contains 50 frames with 10 frames per second. The 3,000 normal videos are randomly sampled from [BDD100K dataset] (https://bdd-data.berkeley.edu/index.html).

4.4.3 Collected dataset

In this dataset, we searched for all kinds of images in the daily route car accident and normal route without accident. We spent alto of time managing and checking this data. And every time we do that, we get different results, we have repeated this process until we got the best results. All data containing 1600, 800 with accident images, and 800 without accident images. We split all data into 1200 images for training and 400 for validating. This data reach almost 90%.

4.4 System snapshots

4.4.1 Encoder Summary

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 128, 264, 1)]	0	[]
input_3 (InputLayer)	[(None, 128, 264, 1)]	0	[]
concatenate (Concatenate)	(None, 128, 264, 2)	0	['input_2[0][0]', 'input_3[0][0]']
input_4 (InputLayer)	[(None, 128, 264, 1)]	0	[]
concatenate_1 (Concatenate)	(None, 128, 264, 3)	0	['concatenate[0][0]', 'input_4[0][0]']
input_5 (InputLayer)	[(None, 128, 264, 1)]	0	[]
concatenate_2 (Concatenate)	(None, 128, 264, 4)	0	['concatenate_1[0][0]', 'input_5[0][0]']
conv2d (Conv2D)	(None, 128, 264, 128)	25216	['concatenate_2[0][0]']
max_pooling2d (MaxPooling2D)	(None, 64, 132, 128)	0	['conv2d[0][0]']
batch_normalization (BatchNormalization)	(None, 64, 132, 128)	512	['max_pooling2d[0][0]']
leaky_re_lu (LeakyReLU)	(None, 64, 132, 128)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 64, 132, 128)	147584	['leaky_re_lu[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 32, 66, 128)	0	['conv2d_1[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 32, 66, 128)	512	['max_pooling2d_1[0][0]']
leaky_re_lu_1 (LeakyReLU)	(None, 32, 66, 128)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 32, 66, 256)	295168	['leaky_re_lu_1[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 16, 33, 256)	0	['conv2d_2[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 16, 33, 256)	1024	['max_pooling2d_2[0][0]']
leaky_re_lu_2 (LeakyReLU)	(None, 16, 33, 256)	0	['batch_normalization_2[0][0]']

Total params: 470,016
Trainable params: 468,992
Non-trainable params: 1,024

Here we have an encoder model which consists of four layers replicated 3 times, first layer is conv 2D which use 128 filter of size (7, 7). Then we move to the MaxPooling layer with filter size (2,2). Then we have BatchNormalization layer which is a normalization technique done between the layers of a Neural Network instead of in the raw data. It is done along mini batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier. And last layer is LeakyReLU.

Leaky version of a Rectified Linear Unit. It allows a small gradient when the unit is not active: f(x) = alpha * x if x < 0 f(x) = x if x >= 0.

4.4.2 Decoder Summary

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 16, 33, 256)]	0
conv2d_3 (Conv2D)	(None, 16, 33, 256)	590080
up_sampling2d (UpSampling2D)	(None, 32, 66, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 32, 66, 256)	1024
leaky_re_lu_3 (LeakyReLU)	(None, 32, 66, 256)	0
conv2d_4 (Conv2D)	(None, 32, 66, 256)	590080
up_sampling2d_1 (UpSampling2D)	(None, 64, 132, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 64, 132, 256)	1024
leaky_re_lu_4 (LeakyReLU)	(None, 64, 132, 256)	0
conv2d_5 (Conv2D)	(None, 64, 132, 128)	295040
up_sampling2d_2 (UpSampling2D)	(None, 128, 264, 128)	0
batch_normalization_5 (BatchNormalization)	(None, 128, 264, 128)	512
batch_normalization_6 (BatchNormalization)	(None, 128, 264, 128)	512
conv2d_6 (Conv2D)	(None, 128, 264, 1)	6273

Total params: 1,484,545
Trainable params: 1,483,009
Non-trainable params: 1,536

The Decoder model is receiving its input from the last layer of Encoder model using a conv-2D as its first layer. Then the Up sampling layer is a simple layer with no weights that will double the dimensions of input and can be used in a generative model when followed by a traditional convolutional layer. Then the Batch Normalization layer, and LeakyReLU layer. These layers are replicated 3 times with some difference in the input shape.

4.4.3 CNN Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 16)      448

 max_pooling2d (MaxPooling2D  (None, 149, 149, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 147, 147, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 73, 73, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 71, 71, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 35, 35, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 33, 33, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout (Dropout)           (None, 16, 16, 64)        0

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 128)               2097280

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 2,157,921
Trainable params: 2,157,921
Non-trainable params: 0
_________________________________________________________________
```

-The size of each image is (300, 300, 3).

-Keras then appends an extra dimension for processing multiple batches to train multiple images in every step of a single epoch. Since batch size can vary, its size is represented by none. Hence, the input shape becomes (None, 300, 300, 3).

-Convolving a (300, 300) image with a (3,3) filter, with strides and dilation rate of 1, and 'valid' padding, results in an output of size (300 - 3 + 1, 300 - 3 + 1) = (298, 298). Since you have 16 such filters, the output shape becomes (298, 298, 16).

-The default MaxPooling kernel has a shape of (2, 2) and strides of (2, 2). Applying that to a (298, 298) image results in an image of shape (((298 - 2)//2) + 1, ((298 - 2)//2) + 1)) = (149, 149).

-This pattern can be extended to all Conv2D and MaxPooling layers.

-The Flatten layer takes all pixels along all channels and creates a 1D vector (not considering batch size). Therefore, an input of (16, 16, 64) is flattened to (16 * 16 * 64) = 16384 values.

-We can simply add a convolution layer at the top of another convolution layer since the output dimension of convolution is the same as its input dimension. We usually add the Dense layers at the top of the Convolution layer to classify the images. However, input data to the dense layer 2D array of shape (batch size, units). And the output of the convolution layer is a 4D array. Thus, we must change the dimension of output received from the convolution layer to a 2D array.

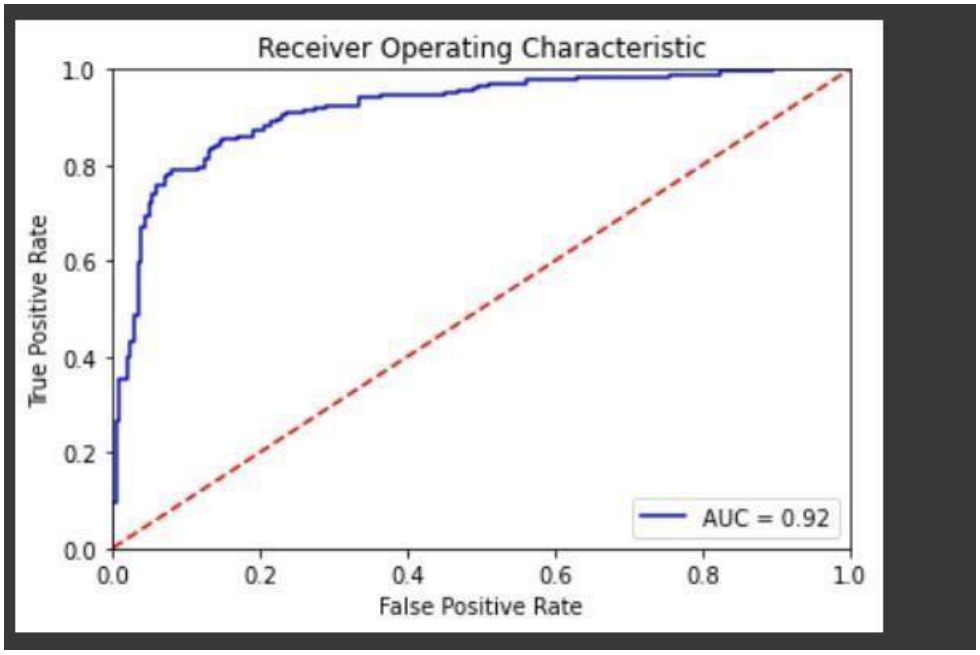
-We can do it by inserting a Flatten layer on top of the Convolution layer. Flatten layer squash the 3 dimensions of an image to a single dimension. Now we only have a 2D array of shape (batch size, squashed size), which is acceptable for dense layers. Then Dense layers, the first with 128 units, the second with 1 unit. The final output shape of your model is (None, 1). It outputs 1values per sample in the batch.

4.4.4 Fit of Model

```
Epoch 1/50
38/38 [==============================] - ETA: 0s - loss: 0.6952 - accuracy: 0.5525WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 89s 2s/step - loss: 0.6952 - accuracy: 0.5525
Epoch 2/50
38/38 [==============================] - ETA: 0s - loss: 0.5971 - accuracy: 0.6758WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 86s 2s/step - loss: 0.5971 - accuracy: 0.6758
Epoch 3/50
38/38 [==============================] - ETA: 0s - loss: 0.5170 - accuracy: 0.7508WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 88s 2s/step - loss: 0.5170 - accuracy: 0.7508
Epoch 4/50
38/38 [==============================] - ETA: 0s - loss: 0.4344 - accuracy: 0.7967WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 87s 2s/step - loss: 0.4344 - accuracy: 0.7967
Epoch 5/50
38/38 [==============================] - ETA: 0s - loss: 0.3725 - accuracy: 0.8383WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 87s 2s/step - loss: 0.3725 - accuracy: 0.8383
Epoch 6/50
38/38 [==============================] - ETA: 0s - loss: 0.3169 - accuracy: 0.8742WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 87s 2s/step - loss: 0.3169 - accuracy: 0.8742
Epoch 7/50
38/38 [==============================] - ETA: 0s - loss: 0.2648 - accuracy: 0.8883WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 86s 2s/step - loss: 0.2648 - accuracy: 0.8883
Epoch 8/50
38/38 [==============================] - ETA: 0s - loss: 0.2498 - accuracy: 0.8925WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 87s 2s/step - loss: 0.2498 - accuracy: 0.8925
Epoch 9/50
38/38 [==============================] - ETA: 0s - loss: 0.1773 - accuracy: 0.9342WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 86s 2s/step - loss: 0.1773 - accuracy: 0.9342
Epoch 10/50
38/38 [==============================] - ETA: 0s - loss: 0.1831 - accuracy: 0.9142WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 86s 2s/step - loss: 0.1831 - accuracy: 0.9142
Epoch 11/50
38/38 [==============================] - ETA: 0s - loss: 0.1353 - accuracy: 0.9442WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 85s 2s/step - loss: 0.1353 - accuracy: 0.9442
Epoch 12/50
38/38 [==============================] - ETA: 0s - loss: 0.1027 - accuracy: 0.9592
Reached 95% of accuracy
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
38/38 [==============================] - 85s 2s/step - loss: 0.1027 - accuracy: 0.9592
```

-Can take 50 epochs in training but will stop when reached **95%** ofaccuracy.

4.4.5 ROC curve

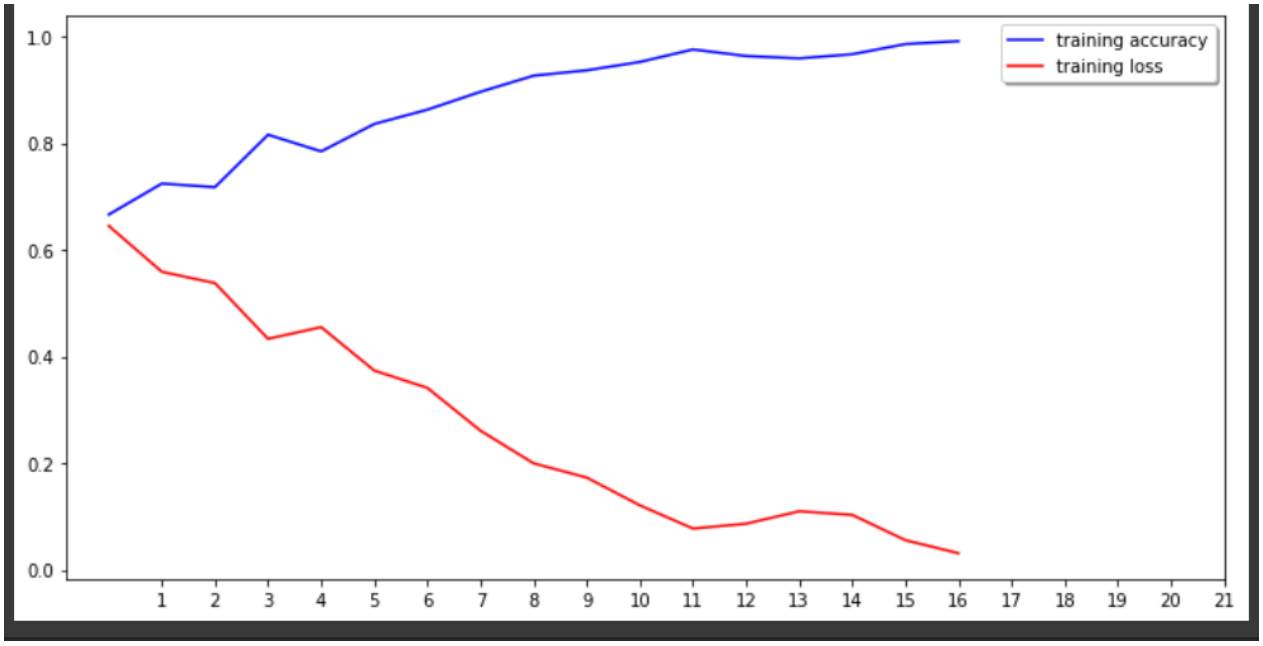


-ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the “ideal” point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better.

-The “steepness” of ROC curves is also important since it is ideal to maximize the true positive rate while minimizing the false positive rate.

-ROC curves are typically used in binary classification to study the output of a classifier. To extend ROC curve and ROC area to multi-label classification, it is necessary to binarize the output. One ROC curve can be drawn per label, but one can also draw a ROC curve by considering each element of the label indicator matrix as a binary prediction.

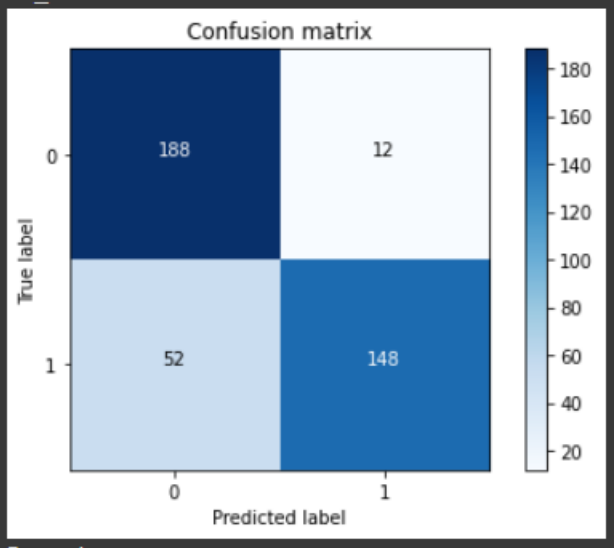
4.4.6 pyplot



-Why the x-axis ranges from 1-16 and the y-axis from 0-10. If you provide a single list or array to plot, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [1, 2, 3,,16].

4.4.7 confuse matrix

```
Accuracy :  84.0
Precision :  92.5
Sensitivity / Recall :  74.0
F1_score :  82.22
```



Report :

	precision	recall	f1-score	support
0	0.783333	0.940000	0.854545	200
1	0.925000	0.740000	0.822222	200
accuracy			0.840000	400
macro avg	0.854167	0.840000	0.838384	400
weighted avg	0.854167	0.840000	0.838384	400

-The precision is intuitively the ability of the classifier not to label as positive a sample that is negative and reached here 92%.

-The recall is intuitively the ability of the classifier to find all the positive samples and reached here 74%.

-The F-measure (F_β and F_1 measures) can be interpreted as a weighted harmonic mean of the precision and recall. A $F_\beta$measure reaches its best value at 1 and worst score at 0. With $\beta = 1$, the $F_\beta$measure leads to the F_1 measure, where the recall and the precision are equally important and reached here 82.22%.

4.5 Sample of code

4.5.1 Image data generator

```
[ ]  train_datagen = ImageDataGenerator(rescale=1./255, data_format='channels_last')
     train_generator = train_datagen.flow_from_directory(
         '/content/drive/MyDrive/train_GAN',
         target_size=(128, 264),
         batch_size=50,
         class_mode='input',
         color_mode='grayscale',
         shuffle=False
         )
```

```
Found 4500 images belonging to 90 classes.
```

```
[ ]  Xtrain=np.concatenate([train_generator.next()[0] for i in range(train_generator.__len__())])
```

-Create instance of image data generator with rescaling 1. /255 and data format “channels_last”.

-The instance calls function flow_from_directory which handles target.

-size (128,264), batch size (50) , class mode = input , color mode = gray scale with no shuffling.

-The last step is looping on all data to convert it to numpy array.

4.5.2 Auto encoder

```
[ ]  temp=code([input_img1,input_img2,input_img3,input_img4])
     dec_output=decode(temp)
     auto_encoder=Model([input_img1,input_img2,input_img3,input_img4],dec_output)
     #opt = Adam(lr=0.0002, beta_1=0.5)
     auto_encoder.compile(loss='mean_squared_error', optimizer = 'adam')
```

Concatenate encoder with decoder model to produce auto encoder.

4.5.3 Training of GAN part one

```
#Define the training of the GAN
def train(g_model, d_model, gan_model, Xtrain, n_epochs=50, n_batch=4):
  bat_per_epo = int(len(Xtrain)/ n_batch)
  half_batch = int(n_batch / 2)
  # manually enumerate epochs
  history = []
  for i in range(n_epochs):
    # enumerate batches over the training set
    print("EPOCH NUMBER ",i)
    for j in range(bat_per_epo):
      number=random.randint(0,50)
      number2=random.randint(0,50)
      print(number)
      # get randomly selected 'real' samples
      X_real, y_real = generate_real_samples(Xtrain,number+4, n_batch)
      X_real2, y_real2 = generate_real_samples(Xtrain,number2+4, n_batch)
      #print(X_real.shape,y_real.shape)
      # update discriminator model weights
      print(X_real.shape,y_real.shape)
      d_loss1 = d_model.train_on_batch(X_real,y_real)
      d_loss11 = d_model.train_on_batch(X_real2,y_real2)
      print("D_loss1:",d_loss1)
      print("D_loss11:",d_loss11)
      print("Discriminator prediction for real images ",d_model.predict(X_real))
      # generate 'fake' examples
      X_fake, y_fake = generate_fake_samples(g_model, Xtrain,number, n_batch)
      X_fake2, y_fake2 = generate_fake_samples(g_model, Xtrain,number2, n_batch)
```

-Train function take parameters (g_model, d_model, gan_model , xtrain , no_ephocs, no_batches).

-Create x_real, y_real, x_real2, y_real2

-Calculate d_loss1, d_loss11

-Create x_fake, y_fake, x_fake2, y_fake2

4.5.4 Training of GAN part two

```
# update discriminator model weights
for k in range(0,4):
  plt.figure(figsize=(100, 100))
  plt.subplot(2,13,1)
  plt.imshow(X_fake[(k),...,0],cmap='gray')
  plt.show()
d_loss_fake2= d_model.train_on_batch(X_fake, y_fake)
d_loss_fake22= d_model.train_on_batch(X_fake2, y_fake2)
discriminator_loss = 0.5 * np.add(d_loss_fake2, d_loss1)
print("D_loss_fake1:",d_loss_fake2)
print("D_loss_fake11:",d_loss_fake22)
print("Dicriminator prediction for fake images ",d_model.predict(X_fake))
# else:
#   print("SKIPPED D TRAINING")
#print(discriminator.predict(X_fake))
# prepare points in latent space as input for the generator
#[z_input, labels_input] = generate_latent_points(latent_dim, n_batch)
# create inverted labels for the fake samples
li=[.95]*(n_batch)
y_gan=np.asarray(li)
# print("OOOO",y_gan.shape)
#y_gan = np.ones(n_batch)
# update the generator via the discriminator's error
z1=Xtrain[number:number+n_batch]
z2=Xtrain[number+1:number+n_batch+1]
z3=Xtrain[number+2:number+n_batch+2]
z4=Xtrain[number+3:number+n_batch+3]
#print(z1.shape)
```

-Update discriminator model weights.

-Calculate d_loss_fake2, d_loss_fake22, discriminator_loss.

4.5.5 Training GAN part tree

```
g_loss = gan_model.train_on_batch([z1,z2,z3,z4], y_gan)
pred =g_model.predict([z1,z2,z3,z4])
x1=Xtrain[number+1:number+n_batch]
x1=np.append(x1,[pred[0]],axis=0)
x1=x1[::-1]
x2=Xtrain[number+2:number+n_batch+1]
x2=np.append(x2,[pred[1]],axis=0)
x2=x2[::-1]
x3=Xtrain[number+3:number+n_batch+2]
x3=np.append(x3,[pred[2]],axis=0)
x3=x3[::-1]
x4=Xtrain[number+4:number+n_batch+3]
x4=np.append(x4,[pred[3]],axis=0)
x4=x4[::-1]
y_gan2 = np.ones(n_batch)
li=[.95]*(n_batch)
y_gan2=np.asarray(li)
print("** ",x1.shape,x2.shape,x3.shape,x4.shape,(y_gan.shape))
g_loss2 = gan_model.train_on_batch([x1,x2,x3,x4],y_gan)
print("PRediction for GAN predictions of fake images are ",gan_model.predict([z1,z2,z3,z4]))
# if(g_loss>0.5):
g_loss3 = g_model.train_on_batch([z1,z2,z3,z4],X_real)
zz1=Xtrain[number2:number2+n_batch]
zz2=Xtrain[number2+1:number2+n_batch+1]
zz3=Xtrain[number2+2:number2+n_batch+2]
zz4=Xtrain[number2+3:number2+n_batch+3]
g_loss4 = g_model.train_on_batch([zz1,zz2,zz3,zz4],X_real2)
print("G_LOSS2 ",g_loss2)
print("G_LOSS3 ",g_loss3)
print("G_LOSS4 ",g_loss4)
print("G_loss:",g_loss)
# summarize loss on this batch
```

-Calculate g_loss

-Predict g_model

-Calculate g_loss2, g_loss3 , g_loss4

4.5.6 MSE and SSIM

```
#Define the metrics to find performance of the model
def mse(imageA, imageB):
  # the 'Mean Squared Error' between the two images is the
  # sum of the squared difference between the two images;
  # NOTE: the two images must have the same dimension
  err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
  err /= float(imageA.shape[0] * imageA.shape[1])

  # return the MSE, the lower the error, the more "similar"
  # the two images are
  return err
def compare_images(imageA, imageB, title,X_train):
  # compute the mean squared error and structural similarity

  for i in range(4):
    m = mse(imageA[i], imageB[i])
    s =ssim(imageA[i], imageB[i],multichannel=True)
    print("Input sequence is")
    plt.figure(figsize=(100, 100))
    for j in range(i,i+4):
      plt.subplot(2,13,j+1)
      plt.imshow(X_train[j,...,0],cmap='gray')
    plt.show()
    # plt.suptitle("MSE: %.4f, SSIM: %.4f" % (m, s))
  # show first image
    print("Predicted Frame is -")
    plt.figure(figsize=(100, 100))
    plt.subplot(2,13,j+1)
    plt.title("MSE: %.4f, SSIM: %.4f" % (m, s))
    plt.imshow(imageA[i,...,0],cmap='gray')
    plt.show()
  # show the second image
    print("Original Frame is -")
    plt.figure(figsize=(100, 100))
    plt.subplot(2,13,j+1)
    plt.imshow(imageB[i,...,0],cmap='gray')

  # show the images
    plt.show()
```

-Evaluation metrics mean square error structure similarity index measure within function compare_images.

4.5.7 Preparation of data

```
def create_data(CATEGORIES,DATADIR,IMG_SIZE):
    for category in CATEGORIES:
        path = os.path.join(DATADIR,category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img))
                img_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                data.append([img_array, class_num])
            except Exception as e:
                pass
    return data
```

4.5.8 callbacks condition

```
[ ]  #arret de l'entrainement en cas de l'accuracy >0.99
     class myCallbacks(tf.keras.callbacks.Callback):
         def on_epoch_end(self,epoch,logs={}):
             if(logs.get('accuracy')>0.95):
                 print ("\nReached 95% of accuracy")
                 self.model.stop_training=True
```

If conduction take train accuracy if (acc > 0.95) print massage (Reached 95% of accuracy) and stop train to avoid over fitting else countian training.

4.5.9 classification

```
[ ]  url=input('Enter URL of Image :')

     #img=imread(url)
     img=cv2.imread(url,3)
     plt.imshow(img)
     img_resize = cv2.resize(img, (300, 300))
     new_img = img_resize.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

     y_predicted = model.predict(new_img)
     print(y_predicted[0])
     print("car",cat[int(y_predicted[0])])
```

-Reading images and showing this image and making some processes resize by img_resize (300,300).

-The model takes images and predict which that have accident or not.

Chapter Five

System results and its performance

In this chapter we will discuss the effect of the model on performance with the help of MSE and SSIM.

Unfortunately, we didn't have enough resources and time to train such a heavy deep learning model, so we will put the expected results.

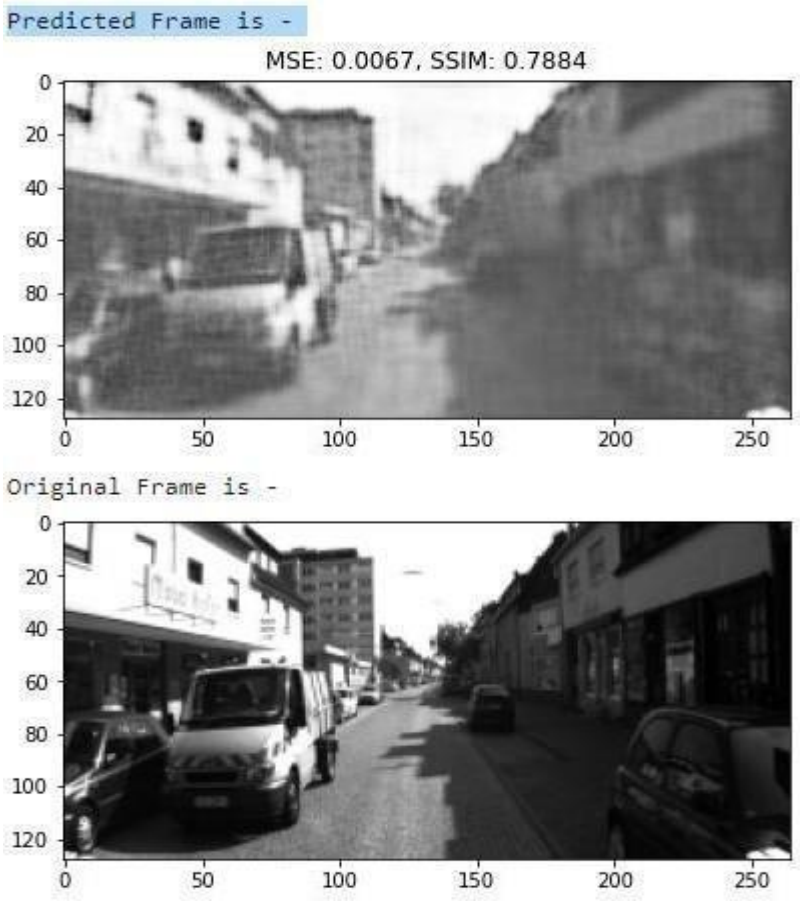


Figure 5.1: predicted frames

Here is a comparison between the predicted frame and the original frame using Mean Squared Error “MSE” and Structural Similarity IndexMeasure “SSIM”.

We think in the future when all cars are turned to smart cars such a system could save a lot of lives, all cars would be fully self-derived. The system is flexible to be trained in the future on different camera views (side view, angle view and back view), in our system we just tried to trainthe model on the front camera of a car. Our system has a lot of capabilities. It just needs resources, time, and data to achieve our goal.

Chapter six

Conclusion and future work

6.1 conclusion

This research has provided a more complete understanding of the accident's prediction. Current findings suggest that the cooperative effect of the different models plays a critical role in human being survival after a traffic road accident.

The relationship between the two models is very clear. Because it is easy and possible to provide an exact predicted frames to the convolutional neural networks phase, these predicted frames generated from Auto encoder and Conditional GAN phase, after the frames pass through CNN it will be classified which it is contain an accident or not.

More complete and accurate documentation of vehicles accident prediction will facilitate easier comparison of individual situations and lead to a more complete knowledge of the processes affecting long-term survival rates for accidents victims. Once we have a clearer understanding of the accident's prediction methods, the Competent authorities can take steps to improve the ways of victims rescue.

6.2 future work

In this project, we have reached the point of predicting accidents, and this system can be improved in the future by adding a system specialized in calculating the geographical location of the accident through GPS, and then an email is sent to the competent authorities in this way in order to rescue the victims as soon as possible, and this system can be applied in smart cars and this happens after integrating machine learning with embedded systems in Automotive industry.

References

[1] C. Hegde, S. Dash and P. Agarwal, "Vehicle Trajectory Prediction using GAN," *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2020, pp. 502-507, doi: 10.1109/I-SMAC49090.2020.9243464.

[2] P. Han, W. Wang, Q. Shi and J. Yang, "Real-time Short- Term Trajectory Prediction Based on GRU Neural Network," *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019, pp. 1-8, doi: 10.1109/DASC43569.2019.9081618.

[3] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 353-359, doi: 10.1109/ITSC.2017.8317913.

[4] Zhifeng Xiao, Qing Liu, Gefu Tang & Xiaofang Zhai (2015) Elliptic Fourier transformation-based histograms of oriented gradients for rotationally invariant object detection in remote-sensing images, International Journal of Remote Sensing, 36:2, 618-644, DOI: 10.1080/01431161.2014.999881.

[5] A. Khosroshahi, E. Ohn-Bar and M. M. Trivedi, "Surround vehicles trajectory analysis with recurrent neural networks," *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 2267-2272, doi: 10.1109/ITSC.2016.7795922.

[6] Shokrolah Shirazi, M., Morris, B.T. Trajectory prediction of vehicles turning at intersections using deep neural networks. *Machine Vision and Applications* 30, 1097–1109 (2019). https://doi.org/10.1007/s00138-019-01040-w.

[7] Frederick Tung; John S.Zelek; David A.Clausi (2011). Goal-basedtrajectory analysis for unusual behaviour detection in intelligent surveillance, Vision and Image Processing Lab, Systems Design Engineering, University of Waterloo, 200 University Ave. West, Waterloo, Ontario, Canada, N2L 3G1, pp.230-240.

[8] Zhihao Zhang; Wenzhong Yang; Silamu Wushour (2020). Traffic Accident Prediction Based on LSTM-GBRT Model, Journal of Control Science and Engineering, pp. 420-6919.

[9] U. Mansoor, N. T. Ratrout, S. M. Rahman and K. Assi, "Crash Severity Prediction Using Two-Layer Ensemble Machine Learning Model for Proactive Emergency Management," in *IEEE Access*, vol. 8, pp. 210750-210762, 2020, doi: 10.1109/ACCESS.2020.3040165.

[10] Xianyuan Wang; Zhenjiang Miao; Ruyi Zhang and Shanshan Hao (2019). I3D-LSTM: A New Model for Human Action Recognition, Xianyuan Wang *et al* 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* 569 032035.

[11] Yassin, S.S., Pooja Road accident prediction and model interpretation using a hybrid K-means and random forest algorithm approach. *SN Appl. Sci.* 2, 1576 (2020). https://doi.org/10.1007/s42452-020-3125-1.

[12] Olaf Ronneberger; Philipp Fischer; Thomas Brox (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation, conditionally accepted at MICCAI 2015, arXiv:1505.04597.

[13] Y. -H. Kwon and M. -G. Park, "Predicting Future Frames Using Retrospective Cycle GAN," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1811-1820, doi: 10.1109/CVPR.2019.00191.

[14] Carreira, J. & Zisserman, Andrew. (2017). Quo Vadis, Action Recognition?A New Model and the Kinetics Dataset. 4724-4733. 10.1109/CVPR.2017.502.

[15] Jasmeen Kaur, Sukhendu Das (2020). Future frame prediction of a videosequence, corpus ID: 2214-70238.

[16] P. Isola, J. Zhu, T. Zhou and A. A. Efros, "Image-to-Image Translation withConditional Adversarial Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967-5976, doi: 10.1109/CVPR.2017.632.

[17] Rossi L, Ajmar A, Paolanti M, Pierdicca R (2021) Vehicle trajectory prediction and generation using LSTM models and GANs. PLoS ONE16(7): e0253868. https://doi.org/10.1371/journal.pone.0253868.

[18] M. T. Suzuki, Y. Yaginuma, T. Yamada and Y. Shimizu, "A Shape FeatureExtraction Method Based on 3D Convolution Masks," *Eighth IEEE International Symposium on Multimedia (ISM'06)*, 2006, pp. 837-844, doi: 10.1109/ISM.2006.13.

[19] H. Vu, H. -C. Kim and J. -H. Lee, "3D convolutional neural network for feature extraction and classification of fMRI volumes," *2018 International Workshop on Pattern Recognition in Neuroimaging (PRNI)*,2018, pp. 1-4, doi: 10.1109/PRNI.2018.8423964.

[20] Jae Gyeong Choi, Chan Woo Kong, Gyeongho Kim, Sunghoon Lim, Car crash detection using ensemble deep learning and multimodal data from dashboard cameras, Expert Systems with Applications, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2021.115400.

[21] W. Liu, W. Luo, D. Lian and S. Gao, "Future Frame Prediction for Anomaly Detection - A New Baseline," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6536-6545, doi: 10.1109/CVPR.2018.00684.

[22] Alankrita Aggarwal, Mamta Mittal, Gopi Battineni, Generative adversarial network: An overview of theory and applications, International Journal of Information Management Data Insights, ISSN 2667-0968, https://doi.org/10.1016/j.jjimei.2020.100004.

[23] Zhao, Junxuan & Xu, Hao & Liu, Hongchao & Wu, Jianqing & Zheng, Yichen & Wu, Dayong. (2019). Detection and tracking of pedestrians and vehicles using roadside LiDAR sensors. Transportation Research Part C Emerging Technologies. 100. 68-87. 10.1016/j.trc.2019.01.007.

[24] S. N. Narayanan, S. Mittal and A. Joshi, "OBD_SecureAlert: An AnomalyDetection System for Vehicles," 2016 IEEE International Conference onSmart Computing (SMARTCOMP), 2016, pp. 1-6, doi: 10.1109/SMARTCOMP.2016.7501710.

[25] Waqas Sultani, Chen Chen, Mubarak Shah; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6479-6488.

[26] M., Swathy & P., Nirmala & P., Geethu. (2017). Survey on Vehicle Detection and Tracking Techniques in Video Surveillance. International Journal of Computer Applications. 160. 22-25. 10.5120/ijca2017913086.

[27] T. Ganokratanaa, S. Aramvith and N. Sebe, "Anomaly Event DetectionUsing Generative Adversarial Network for Surveillance Videos," 2019 Asia-Pacific Signal and Information Processing Association Annual.

[28]Weber, Marc & Klug, Simon & Zimmer, Bastian & Sax, Eric. (2018). Embedded Hybrid Anomaly Detection for Automotive CAN Communication.

[29]N. Dogru and A. Subasi, "Traffic accident detection using Random forest classifier," *2018 15th Learning and Technology Conference (L&T)*, 2018, pp. 40-45, doi:10.1109/LT.2018.8368509.

[30]yiefi zang. (2018), A Better Autoencoder for Image: Convolutional Autoencoder, corpus ID: 209442203.