ADVERSARIAL AUTOENCODERS FOR ANOMALOUS EVENT DETECTION IN IMAGES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Asimenia Dimokranitou

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF THESIS APPROVAL

Dr. Gavriil Tsechpenakis, Chair

Department of Computer and Information Science

Dr. Jiang Yu Zheng

Department of Computer and Information Science

Dr. Mihran Tuceryan

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

Head of Departmental Graduate Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

NN	Neural Networks
CNN	Convolutional Neural Networks
LSTM	Long-Short Term Memory
AAE	Adversarial Autoencoers
VAE	Variational Autoencoders
GAN	Generative Adversarial Network
CAE	Convolutional Autoencoders
HOG	Histogram of Oriented Gradients
SIFT	Sift-Invariant Feature Transform
Conv-LSTM	Convolutional Long-Short Term Memory
ReLU	Reftified Linear Units
HOSF	Histogram of Oriented Social Force
ONEIROS	Open-ended Neuro-Electronic Intelligent Robot Operating System

ABSTRACT

Dimokranitou, Asimenia M.S., Purdue University, May 2017. Adversarial Autoencoders for Anomalous Event Detection in Images. Major Professor: Gavriil Tsechpenakis.

Detection of anomalous events in image sequences is a problem in computer vision with various applications, such as public security, health monitoring and intrusion detection. Despite the various applications, anomaly detection remains an ill-defined problem. Several definitions exist, the most commonly used defines an anomaly as a low probability event. Anomaly detection is a challenging problem mainly because of the lack of abnormal observations in the data. Thus, usually it is considered an unsupervised learning problem. Our approach is based on autoencoders in combination with Generative Adversarial Networks. The method is called Adversarial Autoencoders [1], and it is a probabilistic autoencoder, that attempts to match the aggregated posterior of the hidden code vector of the autoencoder, with an arbitrary prior distribution. The adversarial error of the learned autoencoder is low for regular events and high for irregular events. We compare our approach with state of the art methods and describe our results with respect to accuracy and efficiency.

1. INTRODUCTION

As the availability of data increases, the need to find techniques, that could deal with this great amount of data, is becoming more urgent. Humans are able to learn complex interactions and distinguish meaningful and meaningless scenes. We can instantaneously understand interactions between two people or people and their environment and determine if an action seems normal or not. Although humans can easily classify complex events or annotate a video, the large amount of data that comes with problems like this, makes these tasks very difficult and in many cases impossible. Therefore, there is an essential need to use machines in order to solve these problems.

One of the problems, where a great amount of data is available, and thus, automated methods have proved to be very beneficial, is anomaly detection. Anomaly detection in surveillance videos is an important task for public security, health monitoring, intrusion detection, etc. There are several definitions and interpretations, but in most cases anomaly is defined as a low probability event in the observed data. Thus, detecting the anomaly manually is a very time consuming and meticulous procedure, and automated detection will greatly assist in a more effective detection of these events.

This thesis focuses on the detection of abnormal events in images. The nature of these data makes the abnormality detection very challenging. First, since abnormality is defined as a low probability event, data from the abnormal class are rare and in most cases nonexistent. This makes the detection an unsupervised learning problem. Also, there is a great variability within the same class due to variations in appearance and motion. As a consequence, it is difficult to find a representation of the class that reflects all the variations. More problems arise due to environmental variations, namely background clutter, occlusions, viewpoint changes, etc.

The goal of this thesis is the classification of images as normal or abnormal. As mentioned above due to lack of abnormal instances on the training data, we consider abnormality detection an unsupervised learning problem. Thus, the method used in this thesis is based on autoencoders and more specifically Adversarial Autoencoders [1] which can be described as a probabilistic autoencoder, that attempts to match the aggregated posterior of the hidden code vector of the autoencoder with an arbitrary prior distribution. The reconstruction error of the learned autoencoder is low for regular events and high for irregular events.

The advantage of neural network techniques, over methods that perform classification based on handcrafted features, is that we do not have to pick the features. Neural networks are able to figure out which features are more suitable and meaningful for every situation, therefore, they adapt to unexpected events. As a result, we do not have to predefine the type of anomaly that we want to detect.

In this work, we will classify anomalies that occur in outdoors scenes, the crowd varies from very sparse to dense. We apply our method on UCSD Anomaly Detection Dataset [2], Peds1 and Peds2. In Figures 1.1-1.4 we can see some examples of normal and abnormal events from the datasets. Peds1 consists of clips of groups of people walking towards and away from the camera, and some amount of perspective distortion. It contains 34 training video samples and 36 testing video samples. Peds2 contains clips of pedestrians moving parallel to the camera plane. It contains 16 training video samples and 12 testing video samples.

1.1 Thesis Outline

The remaining of the thesis is organized as follows. In chapter 2, we review existing work in anomaly detection and discuss some popular techniques. In Chapter 3, we give a detailed explanation of the methods used in this work. We present experiments and results in Chapter 4 and conclusions along with future work in Chapter 5.

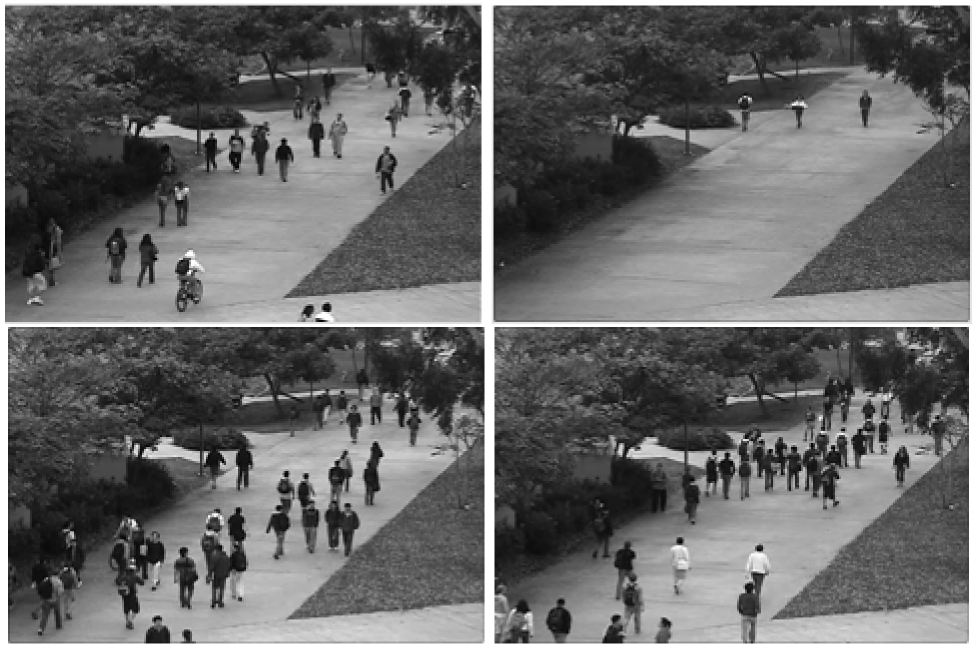

Fig. 1.1. Examples of normal events from UCSD Peds1. The crowd varies from very sparse to dense.

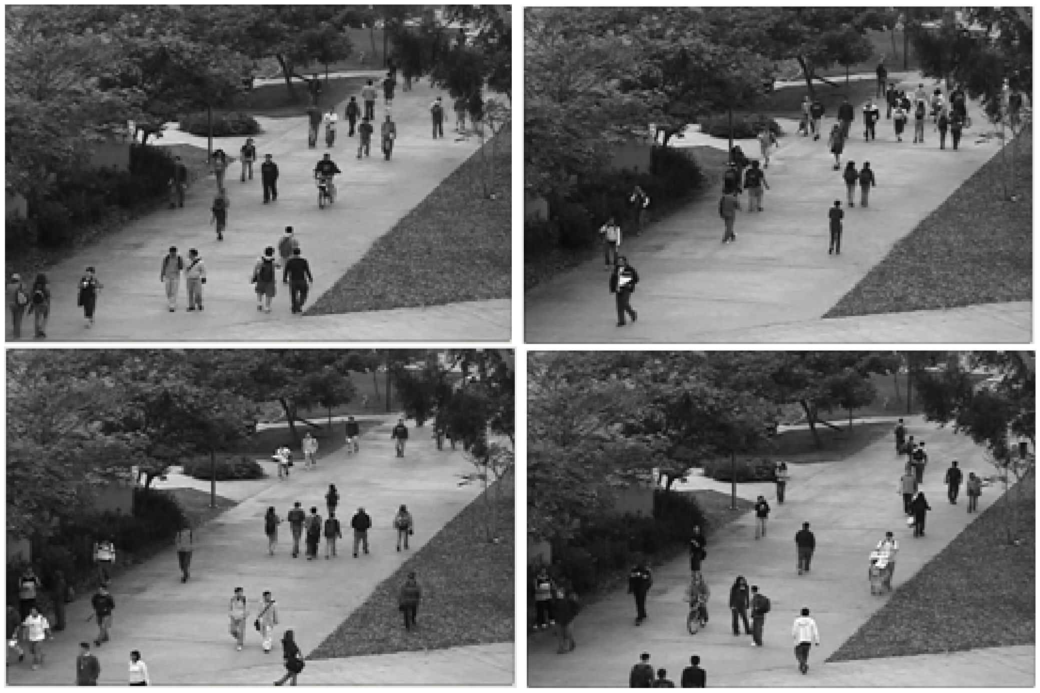

Fig. 1.2. Exampes of abnormal events from UCSD Peds1. From top left to bottom right, the abnormalities are: bike, skate, walking on grass, cart.

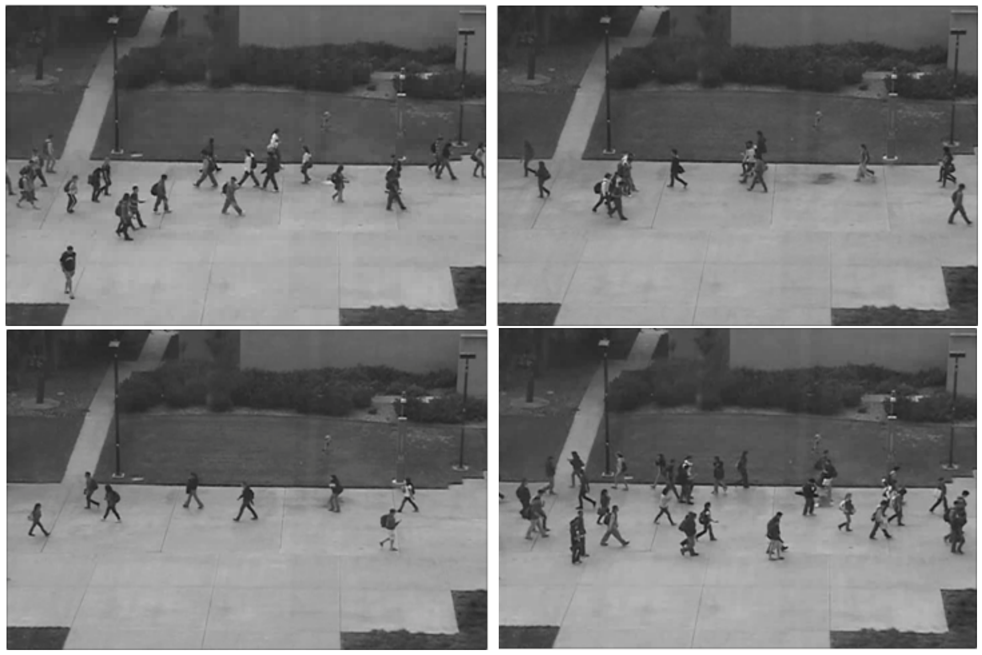

Fig. 1.3. Exampes of normal events from UCSD Peds2.

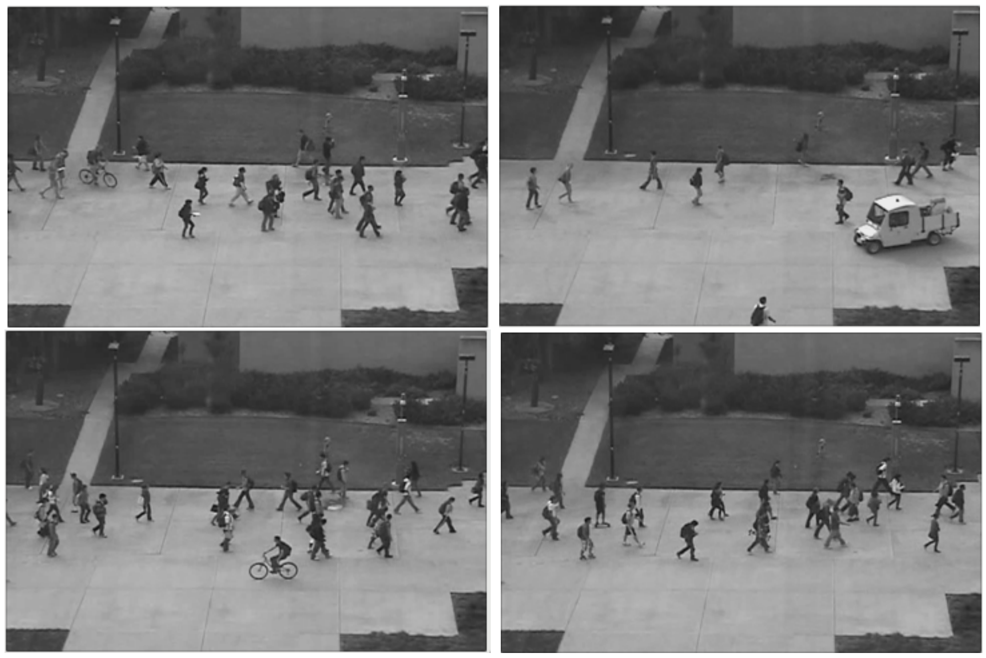

Fig. 1.4. Exampes of abnormal events from UCSD Peds2. The abnormalities from top left to bottom right are: bike, cart, bike, skate.

2. RELATED WORK

In this chapter, we review existing work in the area of anomaly detection. The existing approaches can be separated into two main categories, local feature based methods and deep learning methods. We will briefly describe these methods along with the related work for each one.

2.1 Local features based methods

Anomaly detection is a binary classification problem, where the abnormal labels are uncommon and in many cases nonexistent. As a consequence, we are limited to using only the normal instances. During the training process, we only learn the normal patterns and then, anomalies are defined as events that deviate from these normal patterns. This technique is used in the majority of works in anomaly detection [3, 4]. The idea behind local features based methods is that we first extract handcrafted features from the data (optical flow, Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), etc), and then we classify the events based on these features.

2.1.1 Trajectory based approaches

Trajectories have been widely used in anomaly detection [5–7] and video analysis in general. In trajectory-based methods, the trajectories of the training data are calculated and any significant deviation of the testing data signifies an anomaly. In [5], a method is proposed based on sparse reconstruction analysis of trajectory where the motion trajectories are represented as vectors based on approximating cubic B-spline curves. In [6], Piciarelli et al. propose an approach based on single-class support vector machine clustering. In [7], the trajectories are acquired by Kanade-Lucas-Tomasi Feature Tracker and are then modeled by Multi-Observation Hidden Markov Model. Although trajectory-based methods are suitable for anomaly detection in sparse scenes, it is unsuited for crowded scenes, since it is based on tracking, that still poses a great challenge in computer vision, especially in complex environments like crowded scenes due to occlusions and shadows.

2.1.2 Non-tracking approaches

Other approaches rely on the extraction of local low level features. In [8], normal patterns at each local node are learned by capturing the distribution of its typical optical flow with a Mixture of Probabilistic Principal Component Analyzers. In [9], histograms are used to measure the probability of optical flow in a local patch. Optical flow is also used in [10], where foreground detection is initially performed and then motion, size and texture features are extracted. In [11], Histogram of Oriented Gradients is used on spatio-temporal video volumes. In [12], Yen and Wang use the histogram of oriented social force (HOSF) as the feature vector and then a dictionary of codewords is trained to include typical HOSFs. Also, [13] and [14] follow a sparse coding approach in order to detect unusual events in videos. The main factor, that makes dictionaries less effective, is the fact that they contain many noisy elements.

Although methods that are based on these handcrafted features seem to have good results, they encounter difficulties when unexpected situations (features) occur. Deep learning methods can overcome this problem, since they are able to learn the important features for every situation.

2.2 Deep Learning methods

Deep learning methods are either new or recently reintroduced. They have several applications in computer vision including object and action recognition and human detection [15,16]. Their success relies on the fact that they are not based on handcrafted features. Instead, they are able to learn the features directly from the data. Recently, deep learning techniques were applied in anomaly detection problems [4,17,18].

2.2.1 Supervised methods

In supervised methods data from both classes (normal and abnormal) are present. Spatial-temporal Convolutional Neural Networks for anomaly detection were firstly introduced in [17]. Both appearance and motion were represented by capturing features from spatial and temporal dimensions. Despite the success of supervised methods, their use is limited, due to the fact that abnormal events occur rarely in real-world, and as a consequence there is lack of representation of abnormal events in datasets.

2.2.2 Semi-supervised methods

To overcome this problem, semi-supervised deep learning methods were introduced for anomaly detection problems. In [18], Hasan et al. propose two methods based on Autoencoders. At the first one they combine handcrafted features with Deep Learning methods by learning a fully connected Autoencoder on those features. For the second, they propose an end-to-end learning framework by building a fully convolutional feed-forward autoencoder to learn the local features. Medel and Savakis were the first to employ a Convolutional Long Short-Term Memory (Conv-LSTM) network in [4]. They were able to predict future frames using a limited number of input frames. In [19], Chong and Tay build an end-to-end trainable model based on Autoencoders. They combine spatial and temporal feature extractors using ConvLSTM.

3. ANOMALY DETECTION WITH ADVERSARIAL AUTOENCODERS

Our approach is a combination of Convolutional Autoencoders and Generative Adversarial Networks, called Adversarial Autoencoders. We use Keras [20] with Theano [21] as backend for the implementation of our method.

3.1 Neural Networks, Theano and Keras

In the next section we discuss the characteristics of Neural Networks, Convolutional Neural Networks, Theano and Keras.

3.1.1 Neural Networks and Convolutional Neural Networks

A Neural Network is "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." (In Neural Network Primer: Part I by Maureen Caudill, AI Expert, Feb. 1989). Since all the nodes in the network are connected, Figure 3.1, the complexity of the network is very high and we are limited to using Neural Networks (NN) only when the size of the dataset is relatively small.

Therefore, a variation of Neural Networks was introduced, the Convolutional Neural Networks (CNN). The main difference between CNNs and NNs is the fact that CNNs are sparsely rather than fully connected in the input layer. CNNs mimic the pattern of the neurons in the animal visual cortex. Hubel and Wiesel in an experiment conducted in 1959 discovered that there are neurons dedicated to perceive specific line orientations, and when combined together, they can produce visual perception. CNNs are based on this idea.

A CNN consists of a number of different layers namely Convolutional, Pooling, Fully connected, Rectified Linear Units (ReLU) and Loss, Figure 3.2. Although the architecture may differ from network to network, the main structure remains the same in all CNNs. The first layer in every network is the convolutional, that consists of the set of filters. Each filter is convolved across the width and height of the input volume. At the end an activation map of each filter is learned. Between two Convolutional Layers, there is usually a Pooling Layer. This layer performs a subsampling in order to smooth (reduce noise of) its input. In the case of max pooling for example, the image is divided in regions and only the maximum value of each region is collected. Another layer is the ReLU that controls how the signal flows between the layers. The ReLU function is $f\left(x\right) = max\left\{0, x\right\}$ and although there exist other activation functions ($f\left(x\right) = tanh\left(x\right)$, $f\left(x\right) = \left(1 + e^{-x}\right)^{-1}$ hyperbolic tangent and sigmoid respectively), ReLU is prefered due to its fast training time. After the Convolutional and Pooling Layers, there exist a fully connected layer where all neurons are connected to those at previous layers (same as Neural Networks). The last layer of the network is the Loss Layer, that specifies the penalization between the predicted and the true values. Several loss functions exist namely softmax, euclidean and sigmoid cross entropy.

3.1.2 Theano and Keras

Theano [21, 22] is an open source software library developed by the LISA (now MILA) machine learning group at University of Montreal. It is used for the evaluation and optimization of mathematical expressions in Python. It can run in both CPU and GPU, and in many cases it outruns C when implemented on a GPU. The syntax is symbolic, which means that expressions are defined in an abstract sense and are then compiled to do the calculations. Theano was designed to perform the kind of calculations needed for large scale neural networks and since its release in 2010 it is used for both research and commercial applications.

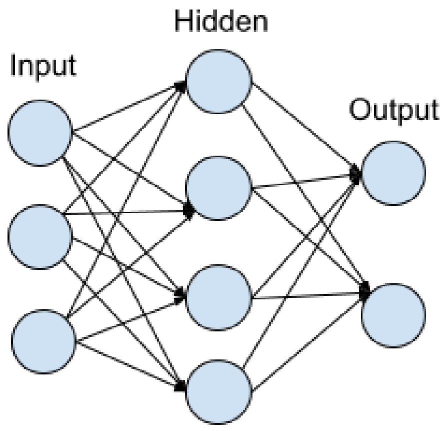

Fig. 3.1. An artificial Neural Network with two layers. The nodes of each layers are fully connected.

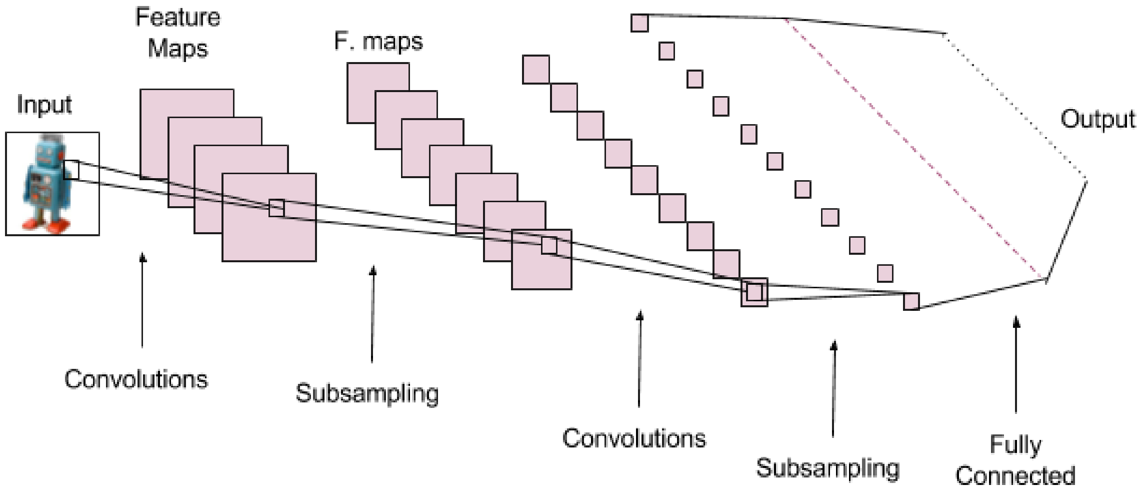

Fig. 3.2. Typical CNN architecture. The network contains Convolutional, Subsampling and Fully connected layers.

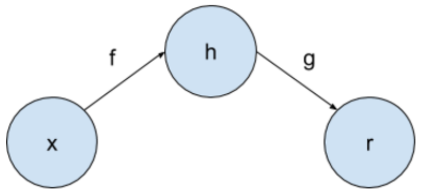

Fig. 3.3. General structure of an autoencoder. The input x is mapped to output r through the hidden layer h. The components f and g of the autoencoder represent the encoder and the decoder respectively.

Keras [20] is a high-level neural network interface written in Python and running on top of Tensorflow [23] and Theano and it was part of ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). The library supports convolutional and recurrent networks as well as combinations of the two. It is a user friendly, simple interface that allows fast experimentation and it runs on both CPU and GPU. Most importantly, all the neural layers, activation functions, optimizers, etc are standalone modules that can be combined to create models with different architectures.

3.2 Autoencoders

An autoencoder neural network is an unsupervised learning algorithm. Traditionally, it was used for feature learning and dimensionality reduction [24]. Its main goal is to copy its input to its output, in other words, it attempts to reconstruct its original input. The network consists of an encoder and a decoder function, $h = f(x)$ and $r = g(h)$ where x is the input, h is the hidden layer and r is the output, Figure 3.3. The most trivial way to map x to r is by using the identity function, which would not yield the desirable results. Also, if $g(f(x)) = x$ for every x in the data, using autoencoders would be meaningless. Figure 3.4 shows an example of an autoencoder.

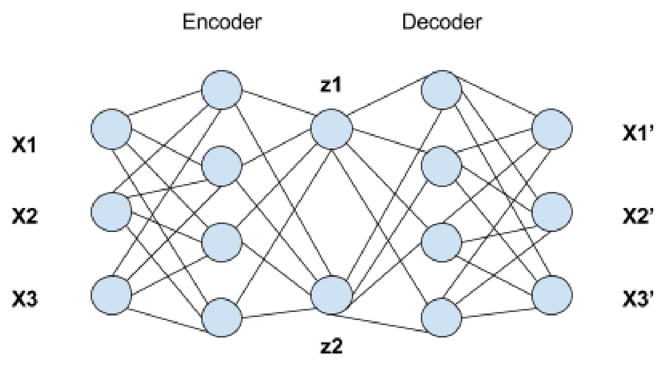

Fig. 3.4. Architecture of an autoencoer with 3 hidden layers.

The general structure of an autoencoder, maps an input x to an output r (called reconstruction) through an internal representation or code h. The autoencoder has two components: the encoder f (mapping x to h) and the decoder g (mapping h to r).

For this reason, autoencoders are not allowed to precisely copy the input, they rather copy approximations of the input. Moreover, they are built with the ability to prioritize which aspects of the input should be copied, as a consequence, most of the time, it learns useful properties of the data. Another technique that is used to ameliorate the learning ability of the network, is to limit the number of hidden units. These autoencoders are called undercomplete. Undercomplete autoencoders try to minimize a loss function by penalizing the output that is not similar to the input. In the case where the decoder of the autoencoder is linear, the results are very similar to PCA. When the decoder is not linear, we can consider it a robust nonlinear PCA. Autoencoders can be considered as feedforward networks, and are often trained using backpropagation (conjugate gradient descent, steepest descent, etc.).

Another case exists where the dimension of the hidden layer is bigger than the input. These autoencoders are called overcomplete and they are not able to learn anything useful about the data distribution, even though they can easily map the input to the output. In order to construct the right architecture for an autoencoder,

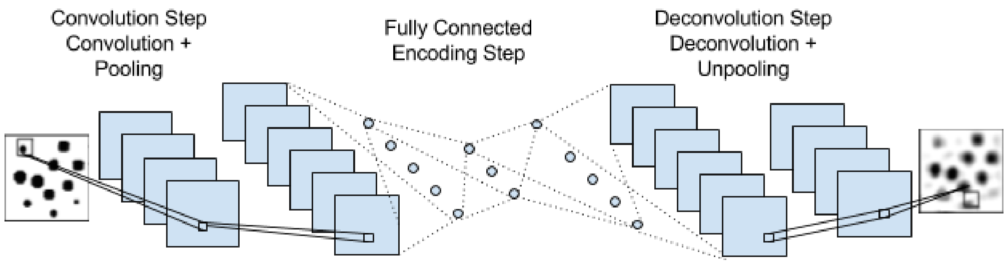

Fig. 3.5. Typical Convolutional Autoencoder architecture. The encoder consists of convolution and pooling layers as well as activation maps stacked atop one another. The decoder consists of deconvolution, unpooling and activation layers as well.

the complexity of the distribution of the data should be taken into consideration. Apart from having small dimensionality in the hidden layer, we can make an autoencoder have the desirable results by making the representation sparse, by penalizing the derivative of the representation or by making it more robust to noise.

In our model we use Convolutional Autoencoders (CAE). Convolutional Autoencoders are autoencoders that have convolutional layers in their encoding and decoding parts. The difference between CAEs and CNNs is that CAEs are trained to learn filters in order to extract the desirable features to reconstruct their input, while CNNs after obtaining the features move on to the classification task. Figure 3.5 shows a CAE.

3.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) was introduced by Ian Goodfellow in [25] and it is a method for estimating generative models by an adversarial process.

3.3.1 Generative vs. Discriminative models

Generative models learn the joint probability, $p\left(x, y\right)$, where x denotes the input data and y the class label. They predict the labels using Bayes rule to calculate $p\left(y|x\right)$ and then pick the label with the highest probability. Some examples of generative models are the Gaussian Mixture Models, Naive Bayes, and Hidden Markov models.

Discriminative models, on the other hand, directly learn how to map the input data x to the class labels y. In other words they learn the conditional distribution $p\left(y|x\right)$. Examples of discriminative models include Logistic Regression [26], Linear Regression [27], and Random Forest [28].

3.3.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) is a system of two neural networks competing in a zero-sum game. The framework consists of two models that are trained at the same time. The generative model G attempts to map a latent space to the data distribution. The discriminative model D tries to discriminate samples that come from G, from samples that come from the training data. The objective of the generative network is to deceive the discriminator into believing that the fabricated data come from the actual data distribution. In the space of arbitrary functions G and D, there exists a unique solution; G can produce data that match the data distribution and D is equal to $\frac{1}{2}$ everywhere. The generator knows how to adapt its parameters in order to fool the discriminator, by backpropagating the classification loss from D back to G. Figure 3.6 shows the architecture of a GAN.

The generator and discriminator are represented by two functions $G\left(\mathbf{z}; \theta_g\right)$, where z is the input noise variables and θ_g are the parameters of the generator, and $D\left(\mathbf{x}; \theta_d\right)$, where x is the observed variables and θ_d are the parameters of the discriminator. They both want to minimize their cost function, $J_G\left(\theta_D, \theta_G\right)$ and $J_D\left(\theta_D, \theta_G\right)$ respectively, by only controlling their own parameters. The solution is a tuple $\left(\theta_D, \theta_G\right)$ that minimizes J_D and J_G.

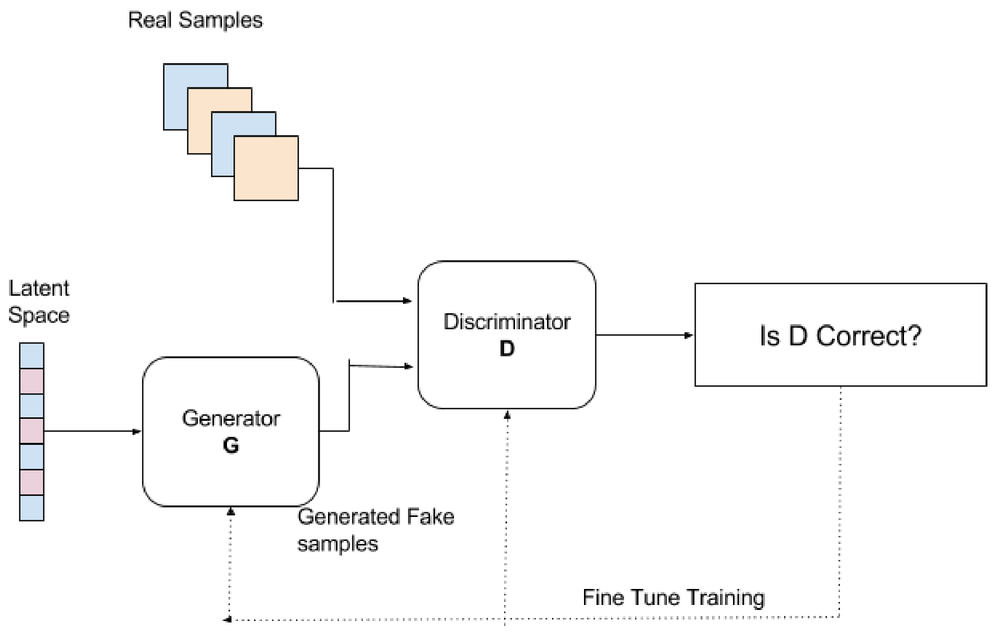

Fig. 3.6. Architecture of a Generative Adversarial Network. The generator G takes as input latent variables that he later pass to the discriminator D. The discriminator takes as input the real samples along with the output of the generator and attempts to discriminator between instances that come from the generator and real samples. If D is not correct, G tries to produce data closer to the real data in order to fool D.

The training process is a stochastic gradient descent. In every step, we have two minibatches, one that consists of values x from the dataset, and the other that is drawn from the model's prior over latent variables. The gradient steps that update θ_D to reduce J_D and θ_G to reduce J_G are made at the same time.

The cost used for the discriminator is:

$$J_D\left(\theta_D, \theta_G\right) = -\frac{1}{2}\mathbf{E}_{x\sim p_{data}}\left[logD\left(\mathbf{x}\right)\right] - \frac{1}{2}\mathbf{E}_z\left[log\left(1 - D\left(G\left(z\right)\right)\right)\right].$$

Since we consider this process a zero-sum game, the cost for the generator is $J_G = -J_D$. We define $V\left(\theta_D, \theta_G\right) = -J_D\left(\theta_D, \theta_G\right)$. Zero-sum games are also called minimax games because their solution involves minimization in an outer loop and maximization in an inner loop:

$$\underset{G}{min}\underset{D}{max}V\left(D, G\right) = \mathbf{E}_{x\sim p_{data}}\left[logD\left(\mathbf{x}\right)\right] + \mathbf{E}_z\left[log\left(1 - D\left(G\left(z\right)\right)\right)\right].$$

GANs seem to have many advantages over other methods like Variational Autoencoders (VAE), Boltzmann machines [29], etc. First, unlike VAE, they are asymptotically consistent, which means that once the equilibrium between the two models is found, we are guaranteed that we found the true distribution that generates the data. Moreover, no Markov chains are needed neither to train the adversarial network nor to draw samples from it, unlike Boltzmann machines. Also, GANs seem to produce the best samples compared to other methods. The main disadvantage of GANs, is that, the training process requires to find an equilibrium between the generator and the discriminator, which in most cases is more difficult than optimizing an objective function. Unlike other deep learning methods, that require a great amount of labeled data in order to generalize well, GANs can be trained with missing data and can also improve the performance of classifiers when limited data is available.

3.4 Adversarial Autoencoders

An Adversarial Autoencoder (AAE) [1] is an approach that can turn an autoencoder into a generative model. Adversarial Autoencoders combine autoencoders with

Generative Adversarial Networks. The AAE has two objectives, first, to minimize the reconstruction error of the autoencoder and second, to match a prior to the posterior distribution (hidden code vector).

Figure 3.7 shows the architecture of the AAEs. The top line represents the autoencoder, where x denotes the input data, $q\left(z|x\right)$ is the encoding distribution and z is the latent code vector obtained after the encoding process of the autoencoder. The bottom line represents the discriminator that predicts if a sample comes from the autoencoder (hidden layer) or from a sampled distribution, $p\left(z\right)$ represents the prior distribution we want to impose on the codes. We calculate $q\left(z\right)$ by

$$q\left(z\right) = \int_x q\left(z|x\right) p_d\left(x\right) dx$$

where $p_d\left(x\right)$ is the data distribution. As we mentioned above, we calculate the regularization error of the AAE by matching the aggregated posterior, $q\left(z\right)$, to an arbitrary prior, $p\left(z\right)$, while the autoencoder aims to minimize the reconstruction error. In our network, the encoder of the autoencoder acts like the generator of the adversarial network, and its goal is to fool the discriminator into believing that $q\left(z\right)$ comes from the actual data distribution $p\left(z\right)$.

We perform the training jointly for both the adversarial network and the autoencoder with stochastic gradient descent. The autoencoder deals with the minimization of the reconstruction error by updating the encoder and the decoder, while the adversarial network, updates the discriminator to distinguish the true data samples from those computed by the autoencoder (hidden layer). Then, the adversarial network updates the generator to produce better data and fool the discriminator. After the training, the decoder defines a generative model that maps the prior to the data distribution.

3.4.1 Architecture

As we discussed above, we propose a Adversarial Autoencoder method for learning regular patterns in images. The architecture of the autoencoder includes 4 convolu-

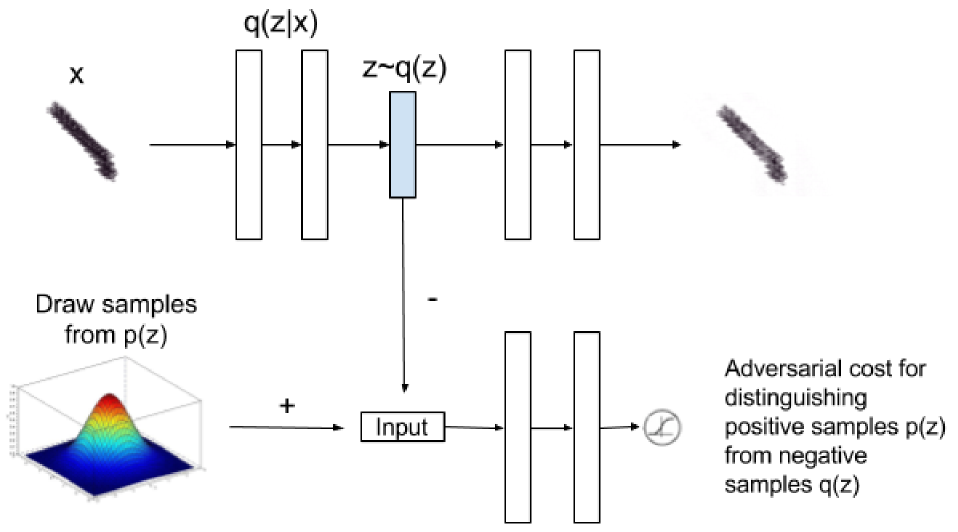
q(z|x)
x
z~q(z)
-
Draw samples from p(z)
+
Input
Adversarial cost for distinguishing positive samples p(z) from negative samples q(z)

Fig. 3.7. Architecture of Adversarial Autoencoders. Top line represents the autoencoder. Bottom line represents the discriminator of the GAN.

tional layers, each of which is followed by a dropout and a max pooling layer. The convolutional layers are connected with a leaky ReLU. The first convolutional layer has 512 filters, which means that is produces 512 feature maps with a resolution of 10 x 10. The pooling layers have kernels of size 2 x 2 pixels and perform max pooling. The rest of the convolutional layers are as follows: 256 feature maps of 10 x 10 pixels, 128 feature maps of 10 x 10 pixels and 64 feature maps of 10 x 10 respectively. The dropout is set to 0.5, which means with that the network will use the information that goes into a neuron with probability 0.5. We use L1-L2 regularization. The discriminator consists of 256 hidden dimensions. As prior distribution for the generator of the network we use the Gaussian distribution. The activation layer of the generator is a sigmoid function and the optimization function is Adam optimizer [30].

4. EXPERIMENTS AND RESULTS

4.1 Dataset

We train our model with two of the most commonly used datasets for Anomaly detection: UCSD Peds1 and Peds2. The videos are created with a fixed position camera for each dataset. Training data contain only normal events, while testing data have both normal and abnormal events. UCSD Peds1 dataset has 34 training and 36 testing videos, where each video contains 200 frames. The videos consist of groups of people walking towards and away from the camera. UCSD Peds2 dataset has 16 training and 12 testing videos, where the number of frames is different for each video. The videos consist of walking pedestrians parallel to the camera plane. Anomalies of the two datasets include bikers, skaters, carts, wheelchairs and people walking in the grass area as well as anomalous pedestrian motion patterns.

4.2 Experiments

We trained two different models, one with the Peds1 and one with the Peds2. First, we resize the frames from their initial size (238 x 158,360 x 240 for Peds1 and Peds2 respectively) to 192 x 192 and then, we normalize them. We trained our the first model (Peds1) for 50 epochs with batch size 32 and the second model (Peds2) for 100 epochs with batch size 32.

Figures 4.1-4.2 show the reconstructed images. The left columns are the actual data and the right columns are the synthesized images. As we observe in Figure 4.1, after 1 epoch the model is able to learn only a vague representation of the background and after 10 epochs, it is able to learn the background in more details. It is only after 20 epochs, when people are starting to show in the synthesized data and this signifies

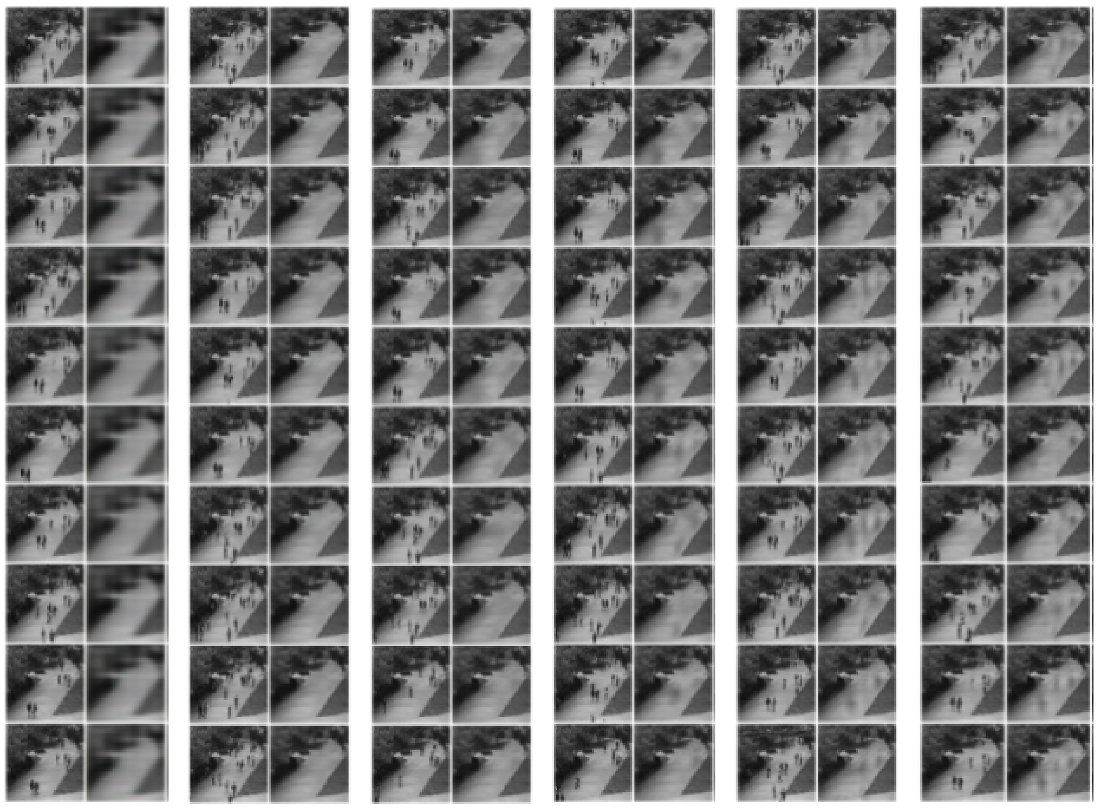

Fig. 4.1. Results of the AAE after 1, 10, 20, 30, 40, 50 epochs (from left to right) in random frames of Peds1. The first column is the original data, the right column is the synthesized data.

the moment when the model starts learning the normal events. After 50 epochs, normal events are learned in more details and we know with some certainty that the model is now able to classify the events as normal and abnormal. Similarly, for the Peds2 dataset, although after 38 epochs the model has learned the background, it is only after 84 epochs when we can clearly observe that normal events are learned.

First, we train our model with the training samples, that only include normal events. Then, in order to evaluate our model, we test new samples. The idea is that since the model was trained only with normal events, it will be able to distinguish positive from negative samples of normal events easily, while it will have difficulties

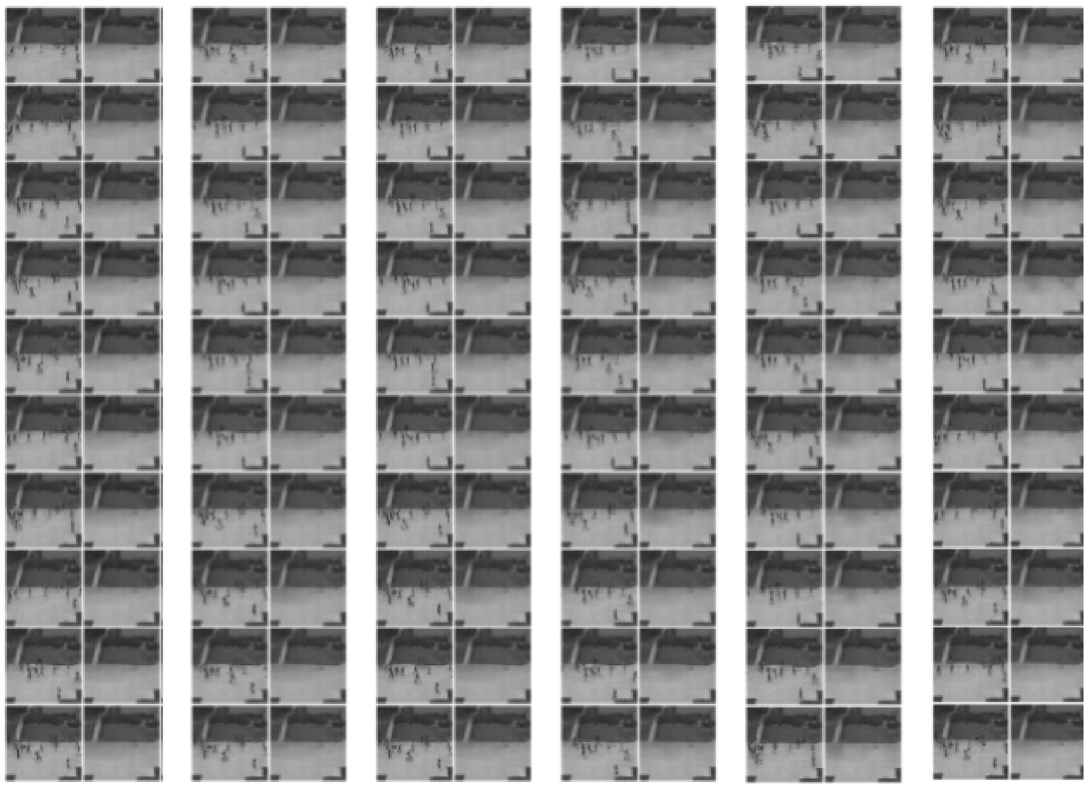

Fig. 4.2. Results of the AAE after 38, 48, 65, 75, 84, 100 epochs (from left to right) in random frames of Peds2. The first column is the original data, the right column is the synthesized data.

distinguishing positive from negative samples of abnormal events. We evaluate the model with a regularity score, that is calculated based on the adversarial error.

For every image t, the adversarial error is calculated by:

$$e\left(x,y,t\right) = \left\|I\left(x,y,t\right) - f_W\left(I\left(x,y,t\right)\right)\right\|_2$$

where f_W is the learned model and I is the pixel intensity at (x, y)

We compute the regularity score $g\left(t\right)$ of a frame t as follows:

$$g\left(t\right) = 1 - \frac{e\left(t\right) - min\, e\left(t\right)}{max\, e\left(t\right)}$$

As a result, samples with regular events will have high regularity score, whereas samples with abnormal events will have low regularity scores, since the model was not trained with abnormal events.

Similar to [4, 18], we reduce the noisy and meaningless minima in the regularity score, with the Persistence1D [31] algorithm, which is used to identify meaningful local minima. As in [4, 18] we assume local minima within 50 frames, belong to the same abnormal event.

4.3 Results

The algorithm is implemented in Python. We use Keras with Theano as backend, Keras is implemented in Python. For the evaluation of our results we use MATLAB (Persistence1d). We perform the experiments on a i5 processor with 8G RAM and NVIDIA 1050Ti. The training process takes about 500 seconds per frame and the testing is real-time (about 1 sec/video).

Our model performs comparably to the state-of-the-art abnormal event detection methods, Table 4.1. We detect 37 out of 40 abnormal events of Peds1 and 11 out of 12 abnormal events in Peds2. Also, a small number of false alarms occur, the precision of our model is higher than [4, 18]. However, we have a lower recall compares to [4, 18].

Table 4.1.
Comparing abnormal event detection performance across UCSD Peds1 and UCSD Peds2.

		Dataset	
		UCSD Peds1	UCSD Peds2
	# Anomalous Events	40	12
True Positive/False Alarm	Ours	37/5	11/0
	[4]	40/7	12/1
	[18]	38/6	12/1
Precision	Ours	0.880	1
	[4]	.851	.923
	[18]	.864	.923
Recall	Ours	0.925	0.916
	[4]	1	1
	[18]	.95	1

Figures 4.3-4.4 show the plot of the regularity score for UCSD Peds1 and Peds2 respectively. Low regularity score denotes the presence of abnormal events.

In Figure 4.3, the red area represents the ground truth and the green area represents the abnormal events detected by our model. As in [4,18], we consider a detected abnormal region as a correct detection if it overlaps with the ground truth by at least 50%. The blue line represents the regularity score of each frame and the magenta line represents the smoothed result after the Persistence1d algorithm. As mentioned above, two local minima that are within 50 frames are considered to be part of the same abnormal event. We can observe that in the first four cases, we have successfully identified the abnormal events. In the fifth plot, we correctly classified the abnormality that occurs in the video, however we also detected a false alarm. In the last plot, two abnormalities occur, we were able to detect only one.

In Figure 4.4, we have correctly detected the abnormal events that occur in these test videos. The fourth plot represents Test 12 and we can see that the detected area does not overlap with the ground truth by 50%, thus, this represents an undetected abnormal event.

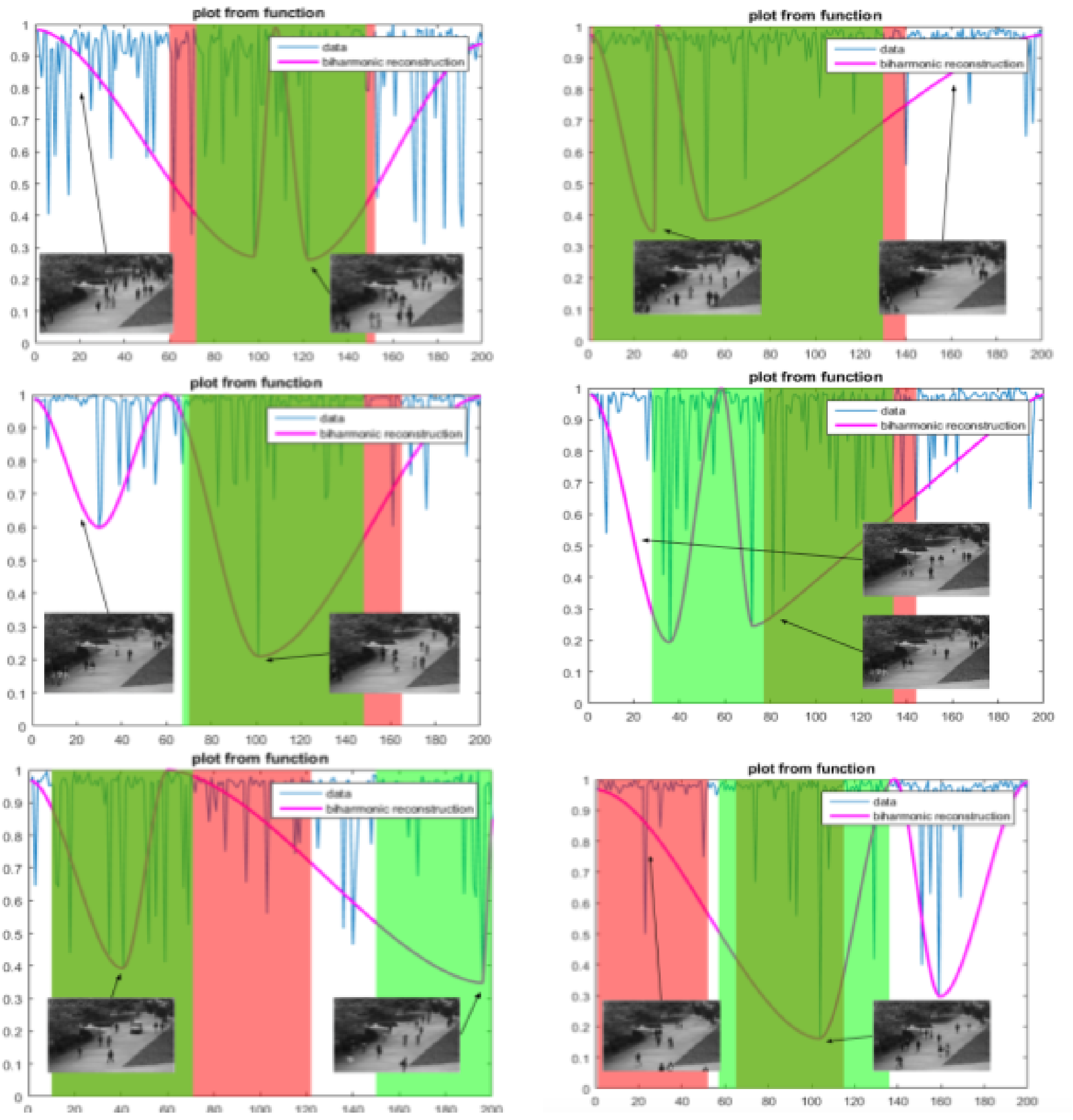

Fig. 4.3. Learned regularity of video sequences in UCSD Peds1. X-axis refers to frame number while Y-axis refers to regularity score. From upper left to bottom right, the events are Test 1, 10, 11, 26, 27, 32.

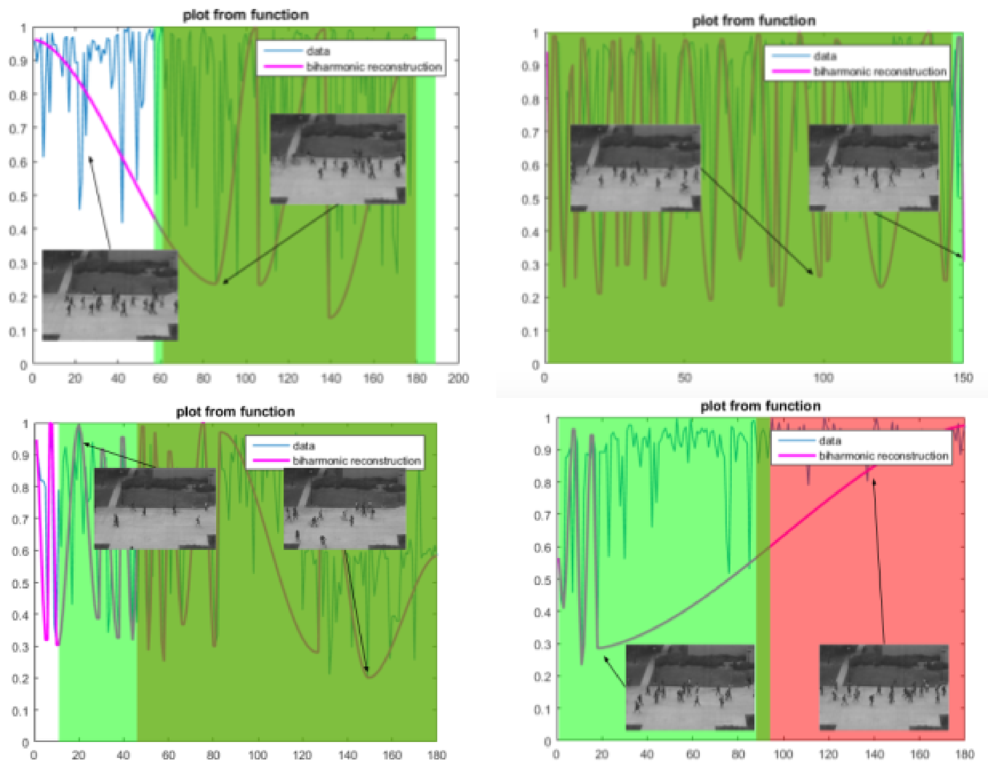

Fig. 4.4. Learned regularity of video sequences in UCSD Peds2. X-axis refers to frame number while Y-axis refers to regularity score. From upper left to bottom right, the events are Test 1, 3, 7, 12.

5. CONCLUSION AND FUTURE WORK

This thesis presents an approach for detecting anomalous events in images. It is based on neural networks and more specifically in Adversarial Autoencoders. This approach was never used before for anomaly detection neither in images nor in image sequences. Our method performs comparably to the state-of-the-art abnormal event detection methods.

The main advantage of our method is the fact that we do not have to use hand-crafted features. The network is able to choose the most important features for our task. This means that we do not have to predefine the anomaly in our data. Anomaly is defined as an event that deviates from the events observed in the training data. Another advantage of this method, is that there is a lot of room for improvements. The main disadvantage of this method is that, although the data we use come from videos, we do not take advantage of the temporal sequence.

Anomaly detection is an active field in Computer Vision. The literature review indicates that many unsupervised methods, mainly based on autoencoders, emerged recently and hopefully will further improve the classification of anomalous events.

5.1 Future Work

Several modifications can be made to this model in order to better detect anomalous events. First, we could use more data. The complexity of Neural Networks makes the need for big amounts of data essential in order to avoid overfitting. However, we always have to deal with a direct trade-off between overfitting and model complexity. If the model is very complex, and we only have a small amount of data at our disposal, it is very likely that we will encounter overfitting. On the other hand, if the model is not very complicated, we risk to lose useful information. We should always aim to find the balance between the two.

One way to achieve this is by combining multiple datasets. Other datasets available for anomaly detection are the CUHK Avenue [14] and Subway (Enter and Exit) [9]. That way, not only our model will be generalizable across the datasets, but we will also have more data to train our model. With bigger amounts of data, we have the advantage to make the network deeper. We anticipate that the results of this work will further improve if more layers are used.

In this work, we detected anomalous events in images, we could further extend this method in order to detect anomalous events in sequences of images. One way to do this, is by inserting memory into the network in order for it to take into consideration previous and next images. One popular technique used for this purpose, is Long short-term memory (LSTM) [32]. Briefly, LSTM allows the network to selectively remember and forget previous data. Its goal is to learn and exploit long-term dependencies in the given data. We strongly believe that the incorporation of LSTM in the Adversarial Autoencoder network will improve the classification results not only in video level, but also in frame level.

REFERENCES

REFERENCES

[1] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow, "Adversarial autoencoders," *CoRR*, vol. abs/1511.05644, 2015.

[2] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, "Anomaly detection in crowded scenes," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 1975–1981.

[3] Y. Cong, J. Yuan, and J. Liu, "Sparse reconstruction cost for abnormal event detection." in *CVPR*. IEEE Computer Society, 2011, pp. 3449–3456.

[4] J. R. Medel and A. Savakis, "Anomaly detection in video using predictive convolutional long short-term memory networks," *CoRR*, vol. abs/1612.00390, 2016.

[5] C. Li, Z. Han, Q. Ye, and J. Jiao, "Visual abnormal behavior detection based on trajectory sparse reconstruction analysis," *Neurocomput.*, vol. 119, pp. 94–100, nov 2013.

[6] C. Piciarelli, C. Micheloni, and G. L. Foresti, "Trajectory-based anomalous event detection." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 18, no. 11, pp. 1544–1554, 2008.

[7] S. Zhou, W. Shen, D. Zeng, and Z. Zhang, "Unusual event detection in crowded scenes by trajectory analysis," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 1300–1304.

[8] J. Kim and K. Grauman, "Observe locally, infer globally: A space-time mrf for detecting abnormal activities with incremental updates." in *CVPR*. IEEE Computer Society, 2009, pp. 2921–2928.

[9] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz, "Robust real-time unusual event detection using multiple fixed-location monitors." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 3, pp. 555–560, 2008.

[10] V. Reddy, C. Sanderson, and B. C. Lovell, "Improved anomaly detection in crowded scenes via cell-based analysis of foreground speed, size and texture," *CoRR*, vol. abs/1304.0886, 2013.

[11] T. Xiao, C. Zhang, H. Zha, and F. Wei, *Anomaly Detection via Local Coordinate Factorization and Spatio-Temporal Pyramid.* Springer International Publishing, 2015, pp. 66–82.

[12] S. H. Yen and C. H. Wang, "Abnormal event detection using hosf," in *2013 International Conference on IT Convergence and Security (ICITCS)*, Dec 2013, pp. 1–4.

[13] B. Zhao, L. Fei-Fei, and E. P. Xing, "Online detection of unusual events in videos via dynamic sparse coding," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11, 2011, pp. 3313–3320.

[14] C. Lu, J. Shi, and J. Jia, "Abnormal event detection at 150 fps in matlab," in *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ser. ICCV '13, 2013, pp. 2720–2727.

[15] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *CoRR*, vol. abs/1406.2199, 2014.

[16] T. Vu, A. Osokin, and I. Laptev, "Context-aware cnns for person head detection," *CoRR*, vol. abs/1511.07917, 2015.

[17] S. Zhou, W. Shen, D. Zeng, M. Fang, Y. Wei, and Z. Zhang, "Spatial temporal convolutional neural networks for anomaly detection and localization in crowded scenes," *Signal Processing: Image Communication*, vol. 47, pp. 358 – 368, 2016.

[18] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, "Learning temporal regularity in video sequences," *CoRR*, vol. abs/1604.04574, 2016.

[19] Y. S. Chong and Y. H. Tay, "Abnormal event detection in videos using spatiotemporal autoencoder," *CoRR*, vol. abs/1701.01546, 2017.

[20] F. Chollet, "keras," https://github.com/fchollet/keras, 2015.

[21] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010.

[22] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USENIX Association, 2016, pp. 265–283.

[24] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, Jul 2006.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2672–2680.

[26] P. McCullagh and J. Nelder, *Generalized Linear Models, Second Edition.* Taylor & Francis, 1989.

[27] F. Galton, "Regression Towards Mediocrity in Hereditary Stature." *The Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, pp. 246–263, 1886.

[28] T. K. Ho, “Random decision forests,” in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, 1995, pp. 278–.

[29] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “Connectionist models and their implications: Readings from cognitive science,” 1988, ch. A Learning Algorithm for Boltzmann Machines, pp. 285–307.

[30] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014.

[31] Y. Kozlov and T. Weinkauf, “Persistence1d: Extracting and filtering minima and maxima of 1d functions,” http://people.mpi-inf.mpg.de/âĹijweinkauf/notes/persistence1d.html.

[32] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, pp. 1735–1780, Nov. 1997.

[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *CoRR*, vol. abs/1603.04467, 2016.

[34] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large scale distributed deep networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS’12, 2012, pp. 1223–1231.