Anomaly Detection in Images

Project Team Members

Sourav Yadav - A20450418

Kapil Khond - A20445656

CS 577 (Spring 2020)

Illinois Institute of Technology

Abstract: In this project we are going to design an encoder-decoder network which will detect anomaly on footwalk street by detecting prohibited vehicles like cars , trucks etc. The given network can be modified to detect other kind of anomalies in field of manufacturing, banking, healthcare, finance, telecom etc.

Introduction: An anomaly refers to a data instance that is significantly different from other instances in the dataset. Anomalies are often harmless but it can be a major problem in online banking, E-Commerce, mobile communications, or healthcare insurance. Anomalies can be statistical outliers or errors. Sometimes anomalies may indicate potential harmful event that have occurred previously. With increasing fraudulent activities, we need more machine learning techniques to identify anomalies. The goal of anomaly detection is to identify patterns in data that do not conform to a well-defined notion of normal behaviour. Early detection of anomalies and faults allows us planning preventive maintenance for model manufacturing, and thus it is crucial for process control. In this project we are going to use unsupervised learning method to train our model in which training data is not labelled.

Background: Detecting traffic and catching traffic violators has always been challenging tasks for law enforcement agencies around the world. In this project we are going to detect walkways street violators using Encoder-Decoder architecture. We are going to use UCSD dataset for training out model. The UCSD Anomaly Detection Dataset was acquired with a stationary camera mounted at an elevation, overlooking pedestrian walkways. The crowd density in the walkways was variable, ranging from sparse to very crowded.

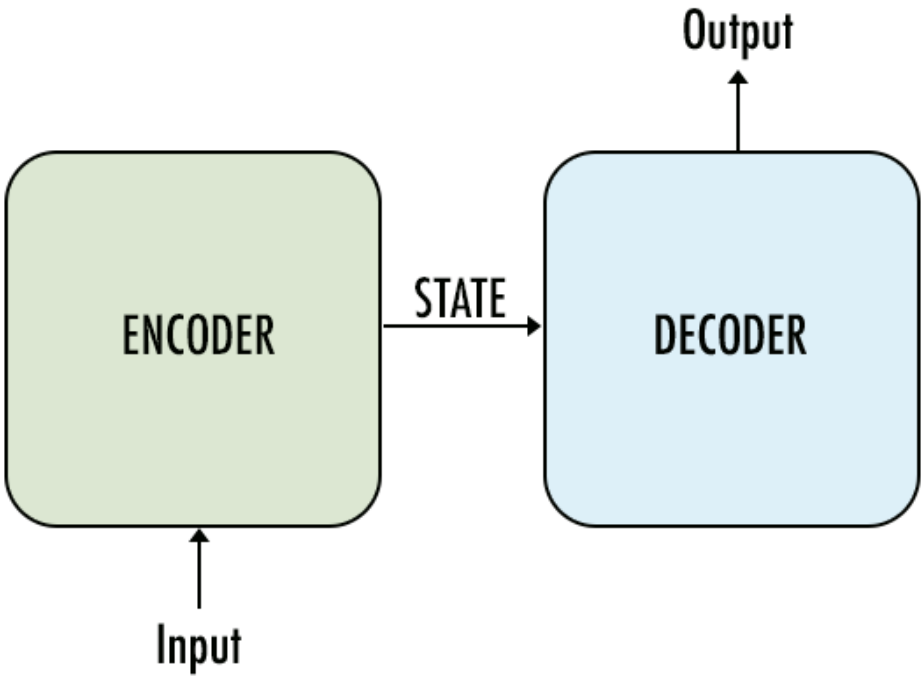
In the normal setting, the video contains only pedestrians.

Abnormal events are due to either:

- the circulation of non pedestrian entities in the walkways
- anomalous pedestrian motion patterns

Training dataset contain only pedestrian images without any anomalies while the test data set contain both normal pedestrian and anomalies like cars, bikers etc. Our model's performance depends on how accurately it detects and marks anomalies on the walkways.

Encoder-Decoder Architecture: The encoder-decoder is used in various applications domains of machine learning including machine translation and image denoising etc. An autoencoder is a neural network model that is trained to reconstruct its input. In other words, an autoencoder is trained to learn an identity function for the data distribution. By constraining to be a low dimensional vector, the training process z encourages the model to learn the most useful information for reconstructing the input.



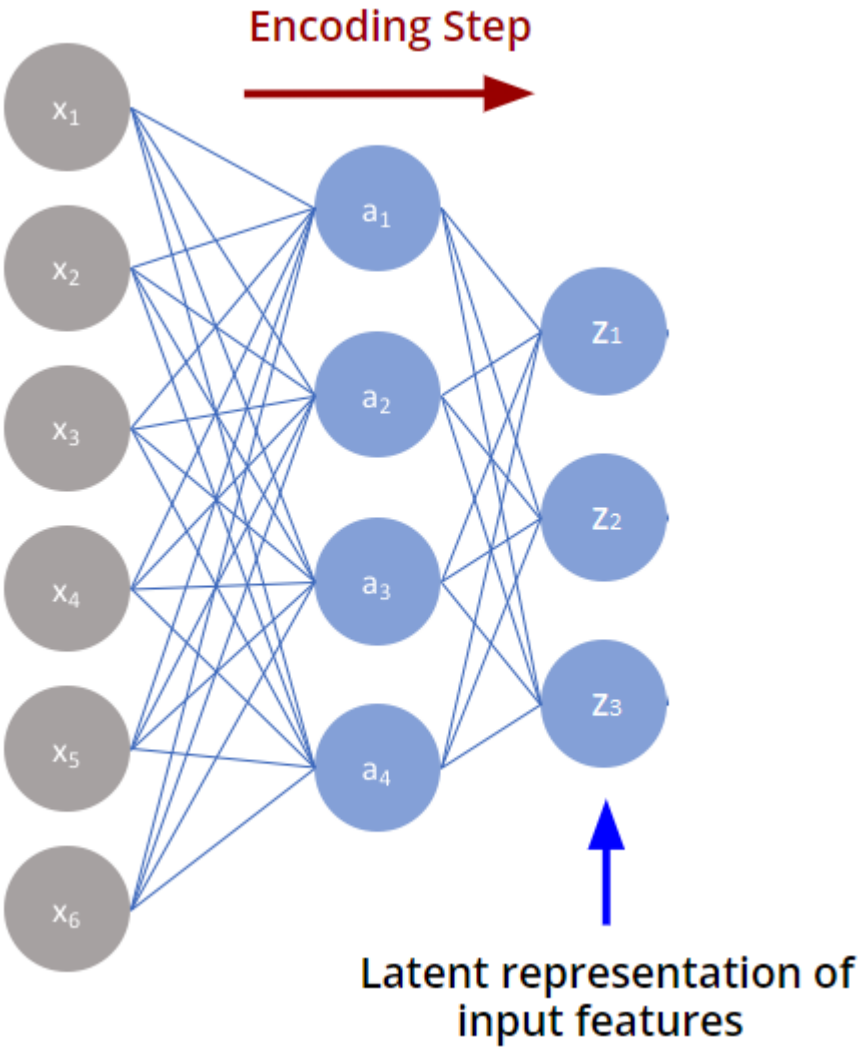
Source: https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d

An encoder-decoder model generally consists of three parts: the encoder, the latent space representation, and the decoder. The purpose of the encoder network is to transform the input data into a latent space representation that is often a vector z; the decoder network then produces the output by decoding z. During training, the encoder and the decoder are trained together to minimize the empirical risk. example.

The Encoder

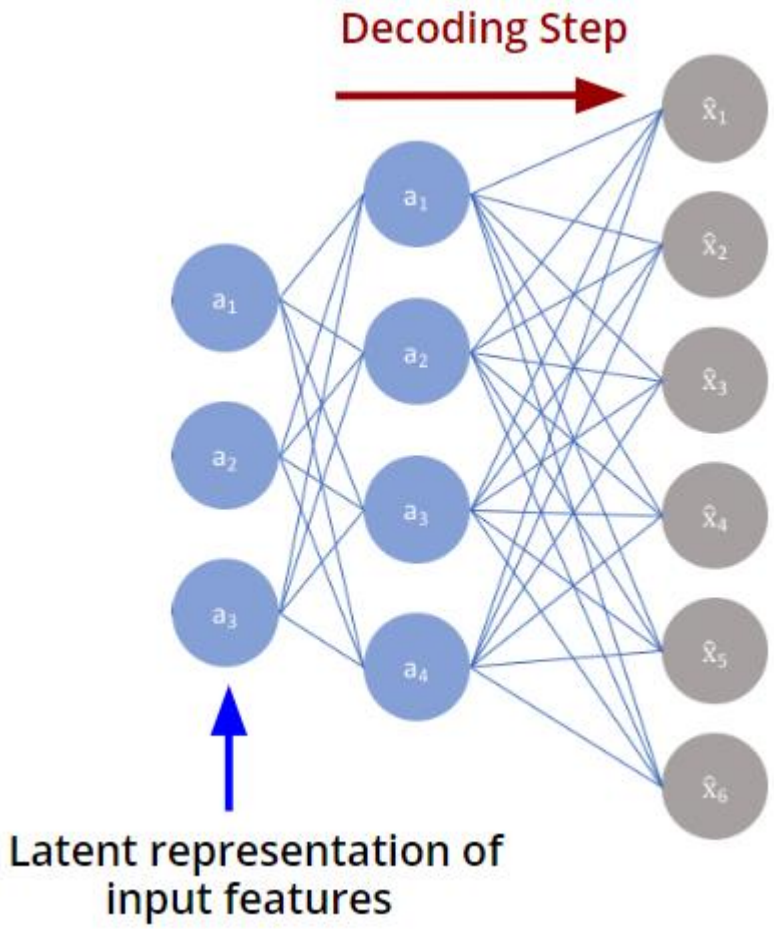
The encoder refers to the first half of the autoencoder, where the number of hidden neurons decreases as the network gets deeper.

The decreasing number of neurons forces an "encoding" or compression of the input features x into a shorter representation that can be found in the middle-hidden layer. Let us call this representation of x as hidden vector z.



The Decoder

The decoder represents the part of the autoencoder where the number of neurons in the hidden layers increases again. The shorter, latent representation of input features that are encoded into the middle hidden layer, are used by the decoder to reconstruct the original input features *x*. Let us call the reconstructed input as *x_hat*.



Why Do We Apply Dimensionality Reduction to Find Outliers?

Don't we lose some information, including the outliers, if we reduce the dimensionality? The answer is once the main patterns are identified, the outliers are revealed. Many distance-based techniques (e.g. KNNs) suffer the curse of dimensionality when they compute distances of every data point in the full feature space. High dimensionality must be reduced. Interestingly, during the process of dimensionality reduction outliers are identified. We can say outlier detection is a by-product of dimension reduction.

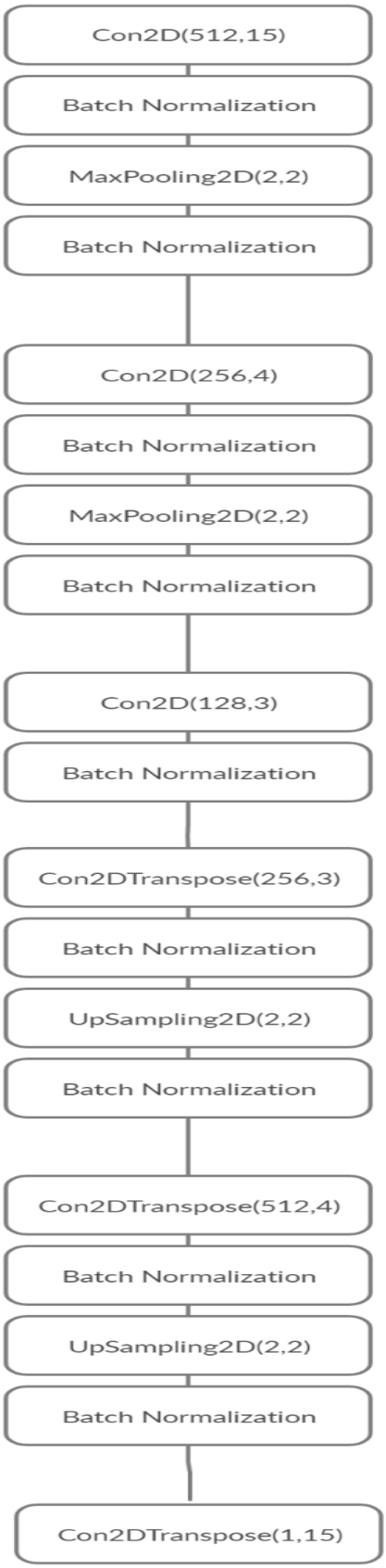
Network Design and Model Training:

We perform below steps in designing and training our network.

- We first download the UCSD dataset from the below link and extract it : http://www.svcl.ucsd.edu/projects/anomaly/dataset.html

```
tar = tarfile.open("UCSD_Anomaly_Dataset.tar.gz")
tar.extractall()
tar.close()
```

- Then we resize our images into shape of (160,160,1) and put them into an array and normalize it by dividing by 255.
- Then we design our network in the below format.



convolutional autoencoder Architecture

Here is Our Model Details :

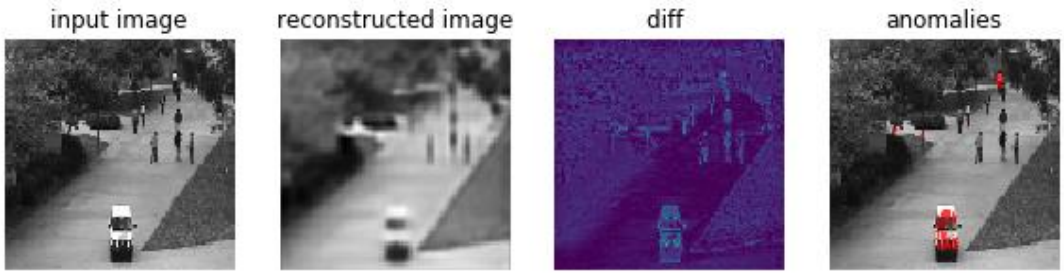
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 160, 160, 1)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 40, 40, 512)       115712
_________________________________________________________________
batch_normalization_1 (Batch (None, 40, 40, 512)       2048
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 512)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 20, 20, 512)       2048
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 256)       2097408
_________________________________________________________________
batch_normalization_3 (Batch (None, 20, 20, 256)       1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 256)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 256)       1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 128)       295040
_________________________________________________________________
batch_normalization_5 (Batch (None, 10, 10, 128)       512
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 10, 10, 256)       295168
_________________________________________________________________
batch_normalization_6 (Batch (None, 10, 10, 256)       1024
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 20, 20, 256)       0
_________________________________________________________________
batch_normalization_7 (Batch (None, 20, 20, 256)       1024
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 20, 20, 512)       2097664
_________________________________________________________________
batch_normalization_8 (Batch (None, 20, 20, 512)       2048
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 40, 40, 512)       0
_________________________________________________________________
batch_normalization_9 (Batch (None, 40, 40, 512)       2048
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 160, 160, 1)       115201
=================================================================
Total params: 5,028,993
Trainable params: 5,022,593
Non-trainable params: 6,400
_________________________________
```

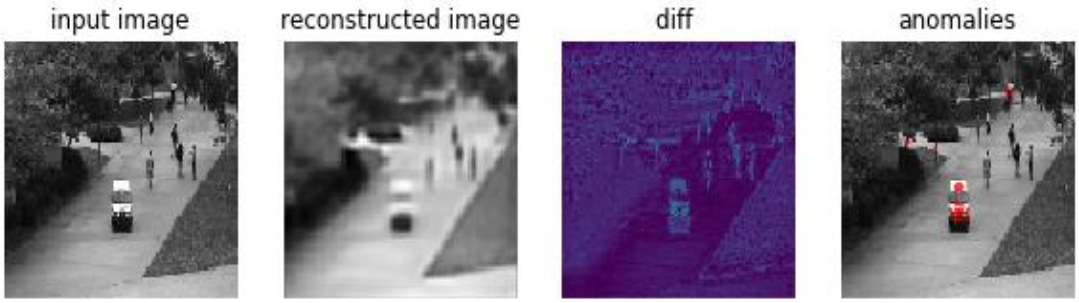
Implementation Details:

- The above network is designed using functional API.
- Then we train our network with features as our image and label as same image.
- Our network's aim it to reproduce our original image, in-turn it will learn all pedestrian's activities.
  (Our training data only contain pedestrian images without any anomalies like cars or trucks).
- Then we save our model.
- Then we test our trained model on the test images given UCSD dataset which contain anomalies like cars and trucks.
- To detect anomalies, we take the difference between the original image and recovered image and apply a convolution filter consisting of all ones with a dimension of 4*4. We have used a threshold value. In case of an image without any anomaly, difference between original and reconstructed image will be very less and after applying filter we see all pixel values close to zero. In case of anomaly, our network fails to reconstruct the vehicle portion, hence difference between original and reconstructed image will be high, and when we apply filter (all ones with a dimension of 4*4) we end getting a patch whose pixel values are close to 1.
- So, we highlight the portion where pixel value crosses threshold, in our case pixel with car object gets highlighted.
- We mark the detected anomalies with red marks in the test images.

Results and Evaluation:

After training and hyperparameter we check the results on test data and got below results on test images.





From the above figure, we can see that there is a car on the walkways which is the anomaly.

Instructions to run the code:

A. Download the data from the link provided in data file (inside data folder). Extract it and load the test data from the code given in Project_final.py file.
B. Run code with following headings present in Project_final.py file:
    1. loading the model
    2. Function to Plot anomaly
    3. Loading Test Data Set
    4. Run this code to view original, reconstructed,diff and anomalies

References:

[1] Baihong Jin Yingshui Tan, An Encoder-Decoder Based Approach for Anomaly Detection with Application in Additive Manufacturing, NY: arXiv.org,2019

[2] M. Haselmann, D.P. Gruber, P. Tabatabai, Anomaly Detection using Deep Learning based Image Completion, NY: arXiv.org, 2018

[3] Saikiran Bulusu, Bhavya Kailkhura, ANOMALOUS INSTANCE DETECTION IN DEEP LEARNING: A SURVEY, NY: arXiv.org, 2020

[4] https://github.com/NRauschmayr/Anomaly_Detection

[5]https://stackabuse.com/autoencoders-for-image-reconstruction-in-python-and-keras/

[6] https://www.deeplearning-academy.com/p/ai-wiki-anomaly-detection

[7] https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6

[8] https://medium.com/analytics-vidhya/unsupervised-learning-and-convolutional-autoencoder-for-image-anomaly-detection-b783706eb59e