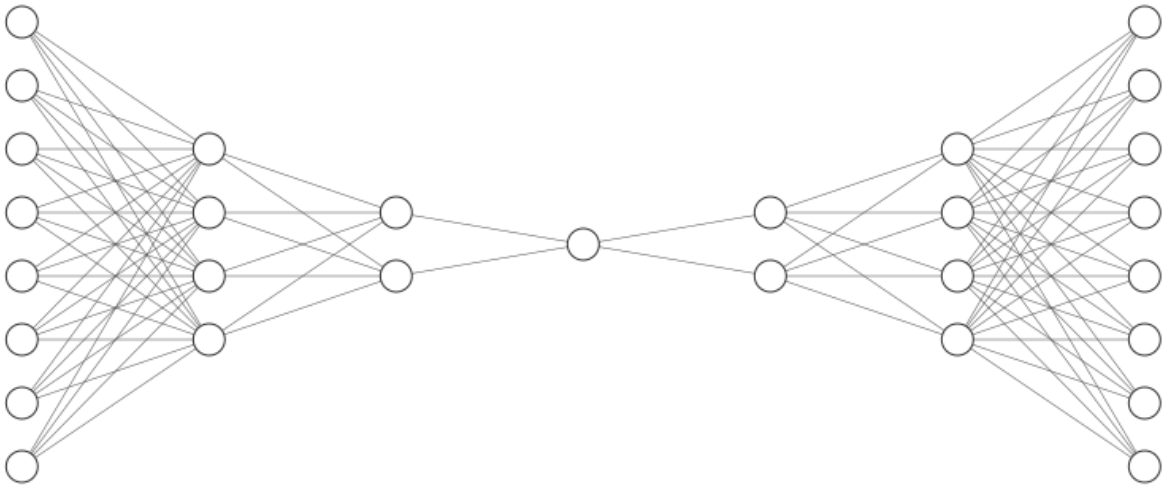
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Convolutional Autoencoder for Anomaly Detection

IST Project-Report

A.Y. 2020/21

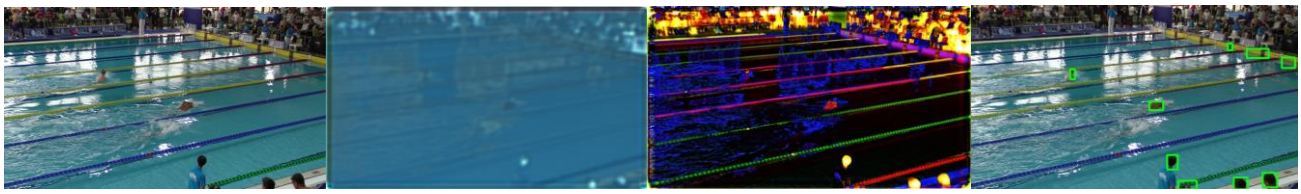
Professors Stefan Duffner, Christophe Garcia, Nicolas Jacquelin



Carlucci Francesco

Introduction

The aim of the project is to detect objects in an image that are unknown to the network, by its reconstruction error. The network is trained on images that don't contain the specific object, so it is able to reconstruct exclusively these images, especially the background texture of the environment, while everything is not in the training dataset is an anomaly to him, so it will poorly reconstruct it. This way we can then detect the objects of interest by analizing its reconstruction loss, the difference between the reconstructed image and the input. We've used a convolutional autoencoder trained on a dataset of images from swimming competitions, its task is to reconstruct the empty pool only so we can detect the swimmers in the test set, that are our anomalies, by calculating an heatmap of the reconstruction error.

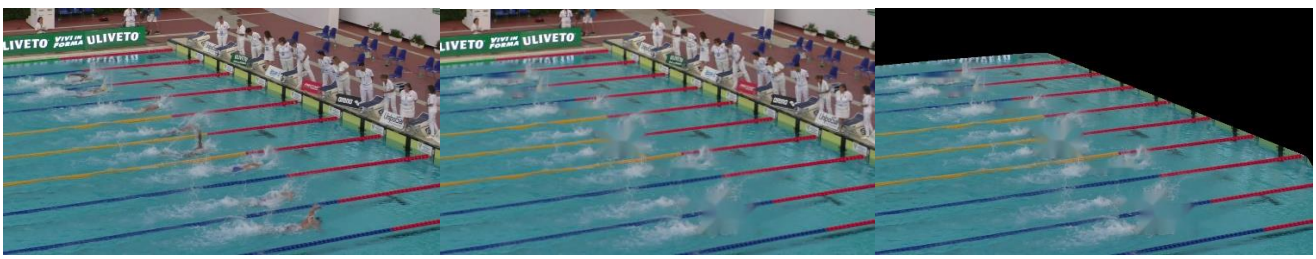


An example of how the mechanism works: 1. Input image; 2. Reconstructed image; 3. Error Heatmap; 4. Segmented Anomalies;

Dataset

The dataset includes 317 images of swimming pools taken from competition video, each with an annotation file of the swimmers positions. The images have dimensions 1600x900 and 308 of them have been used as training set. To make the swimmers an anomaly we had to eliminate them from the training set, so a python scripts has inpainted the swimmers regions based on the annotation in the file. The inpainting procedure fill the removed rectangle with pixels similar to the neighbouring regions. This technique generates artificial data as close as possible to an empty pool or an image of the water texture.

Next preprocessing step needed was the removal of the out of the pool region, where there are spectators and objects the autoencoders didn't have to learn. Another script detected the pool region in the image and produced a binary mask of the pool shape. The script used two main techniques, looking for the blue color of the water and locating the lines of the pool lanes and borders. The problem is complicated by the difference in color and orientation of the pool in images from different videos, resulting in a tradeoff between wrongly included non pool and excluded border pool regions depending on the thresholds and parameters of the script. All the image modifications, included the target modification for the training, have been made with the OpenCV library for python.

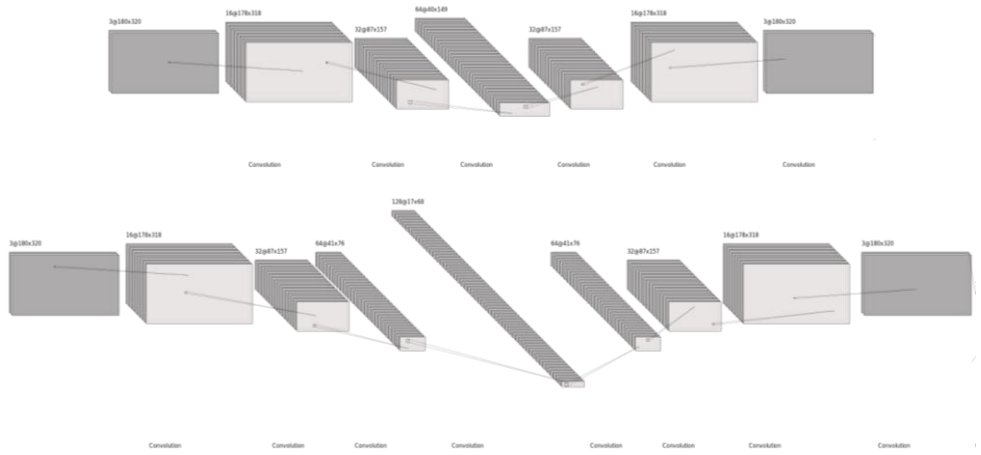


An example of preprocessing: 1. Original image; 2. Inpainted image; 3. Non pool mask applied to the image;

Autoencoder architecture

The autoencoder is fully convolutional, in order to achieve translational invariance. Two architecture have been tested, first one with 3 layers for the encoder and 3 for the decoder, the second one with 4 layers each, with increasing kernels and number of channels. The encoder and the decoder are symmetric. The first architecture was too little compressive, it soon learned to reconstruct everything in the image too precisely, swimmers too. The network must have a quite compressive bottleneck because a not limited enough system would generalize well, adapting to new types of data, while we need a system that reconstruct only the pool.

The convolutional layers are connected by a ReLU activation function. The models are implemented in pytorch and trained using mean squared error loss function and the Adam optimizer. The autoencoders are feed with input images resized to a fifth of the size during training, to speed up computation. To avoid the autoencoder from learning the surroundings of the pool the non-pool region of the output image is set equal to the input pixels, so the loss will be always zero. Slightly different modifications to the non-pool region have been tested, with the goal of limiting the autoencoder reconstruction capabilities. At the end of the training the model is saved along with the optimizer parameter, for testing or continuing the training, and the results of the model on the training set are shown to verify it.



Scheme of the two architecture with kernels 1. {3,5,9} 2. {3,5,7,9}. The second architecture is the one that produced the final results.

Data augmentation

The models have been trained first on the 308 images of the training dataset, then on an augmented dataset of 2009 images that include also images modified by means of random cropping, rotating, flipping vertically and horizontally, random saturation and brigthness changing and normal filtering. The augmenter script generate for each image the corresponding mask too. Another smaller dataset of 1148 images have been tried too but the time needed to train on the bigger dataset is not so large to justify the use of this last one.

Some Data Augmentation techniques used, like brightness and saturation change, random crop, flip and rotation

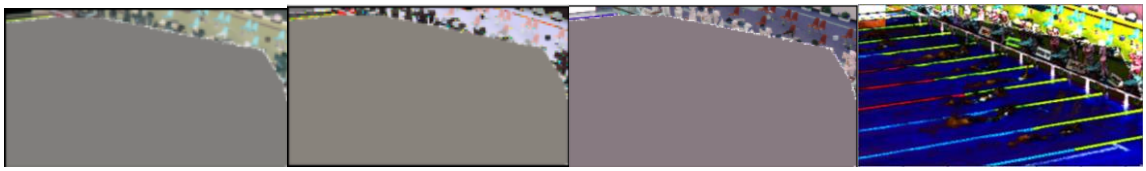
Tests

The models are tested on a test set composed of images from a different competitions, so a video the autoencoder has never seen before, both with and without swimmers. The test images are normalized from [0,1] to a smaller range smaller to reduce the image luminosity and the brighter spots of the image are set to the mean value, the heatmap is computed by pixelwise squared error then elevated to a coefficient and normalized to make more visible the regions with high loss against the negligeable error in the background, that get reduced to black. The input image, the loss heatmap and the reconstructed image are shown while the loss image is saved.

The loss images are then processed by the Detector program that uses methods similar to the pool detection script to search dark red or purple regions corresponding to swimmers. The loss image is cleaned to remove the error caused by pool lanes or light reflections on the water and then the detected bounding boxes are drawn on the original test images.

Improvement attempts

The pdf file Models in [4] collect every model trained during the project. There are the models trained on the non augmented dataset, on 10, 50 and 100 epochs, the ones trained on augmented data, for 5, 10, 20, 50 and 80 epochs, the models with the old architecture and there are also some further tests made with the aim to limit the reconstruction capabilities of the autoencoder, so that the system can be trained to reconstruct better the pool without learning the swimmers, making them easier to identify. The first attempt was a dataset where the pool has been detected in a more conservative way, then new kernels have been tried, to shrink the bottleneck, making both architectures more compressive, but the loss didn't change significantly. The last tests were based on the addition of some percentage of noise to the non pool region of the target image during training, the aim was to add another term the autoencoder have to optimize that tend to maximise the loss for the non pool region, avoiding the network to learn it. Gaussian noise, gaussian noise upper bounded by a margin, an inverted pixel target and a module shift (setting the pixel far from the input through a non random function) have been tried. These noise technique have been tested up to 8 epochs, the problem with a model as simple as this autoencoder is that the training can easily fall in an all 0 or all 1 state, but nevertheless the technique is still valid and should be further studied. In particular, the article [5] thoroughly explore the effect of noise in neural networks training, aiming to explain the degree of pattern learning and memorization of the network.

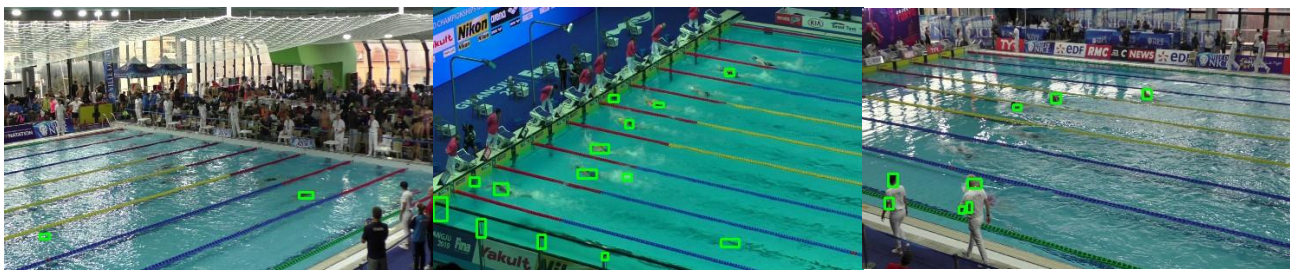


Some of the modification of the target image tested, the gray region is the output of a untrained model, the non pool region is filled with the input image pixels modified: 1.Inverted Pixels 2. Module shift 3.Noise bounded by a margin 4. Resulting loss of the inverted pixel method, note that there is a higher loss in the out of the border region.

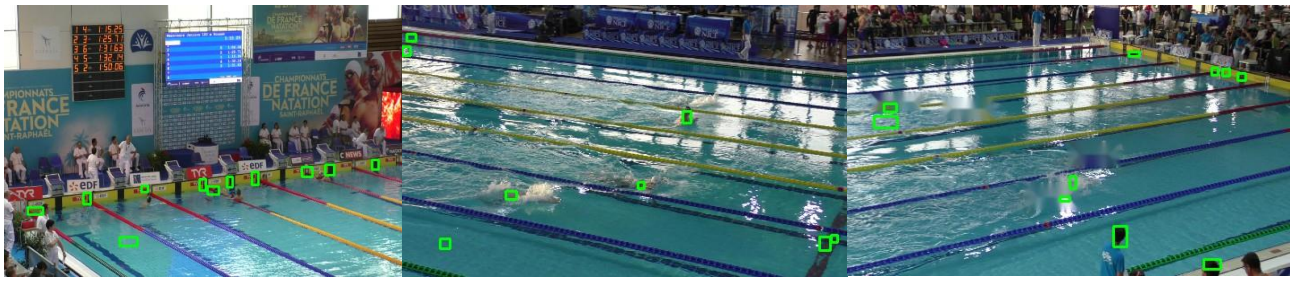
In fact the only way for a CNN to reconstruct random noise would be brute force memorization, in [5] it is proved that, although a network can learn noise only inputs and there is indeed a process of memory during learning on noisy data as the network learns noise too, it starts to do it after some epochs of training, so it learns general patterns first and only then it starts generate more complex hypothesis to explain the noise. They used the critical sample ratio to estimate the complexity of decision regions throughout the training, observing that it increases gradually in all cases (real data, noise input and noise labels) stabilizing later and at a higher value in the presence of noise so the network always goes from simple to more complex hypothesis but need more training to fit the noise. The final accuracy reached by the model is also lower, so noise effectively interfere with the learning of real data. They showed that in the case of noise in the target the phenomenon is stronger and it reach a peak of accuracy on the clean validation set before reaching maximum accuracy on the training set, at the turning point in which it starts learning the noise the accuracy on the clean dataset decrease, so using modified noisy target is effective in limiting the model reconstruction capabilities as it learns general patterns first and then degrade its performance to fit the noise. Noise has an effect on the required model capacity and training time too: a higher capacity is needed to achieve optimal performance at the end and decreasing capacity yield a stronger increase in training time needed to fit noise rather than real data. So noise would allow to train the model longer without it to generalizes and learn anomalies. In addition to early stopping in the training the article experiments with regularization techniques to limit the speed of noise fitting without affecting useful pattern learning, the most effective techniques seems dropout, more than the relatively weak weight decay we used, further tests should be performed using the above modifications and dropout.

Results

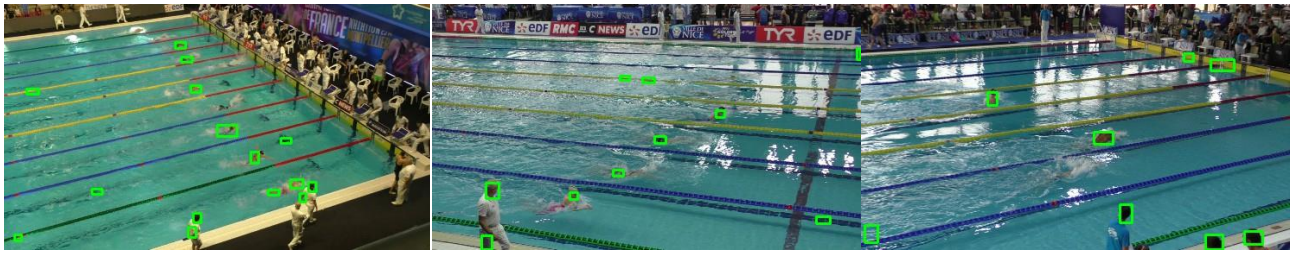
The produced heatmap show that in most of the images of the test set there is an error corresponding to the position of the swimmers, in particular for the part of the body out of the water or completely submerged shapes, but there are also other errors, very similar, due to the pool lanes and water movements. These diffused errors makes harder the automatic detection of swimmers with the OpenCV script, that therefore has still some missed detection and false positives, the latter particularly caused by pool lines, objects on the border or inpainted regions. We can see that the model was not able to build a clear heatmap, because it still generalizes too much, when it learns the texture of the water good enough to have less background error it starts to reconstruct swimmers too so its loss decreases too uniformly. The best results can be seen on the models trained on the augmented dataset on 20 or 50 epochs, at 80 epochs it reconstructs very accurately the image, but starts to learn anomalies too. It's possible to detect the bounding boxes on a model trained on non augmented data for 10 epochs too, properly setting the input size.



Bounding boxes detected on the model trained on the augmented dataset for 50 epochs



Bounding boxes detected on the model trained on the augmeted dataset for 20 epochs



Cases of missed detection and false alarms on the model trained for 20 epochs

Related Works

Looking to previous works similar, in [1] an extremely compressive RBM is used, trained on really small patches of road images to learn only the road texture. It is able to detect anomalous object in the road by the reconstruction loss of each patch as it poorly reconstruct everything that is non road. More refined models would perform better, we can see something similar in [3] where an adversarial autoencoder is used. This approach aim to turn an autoencoder into a generative model by combining it with a GAN discriminator that calculate the adversarial cost from the autoencoder hidden code. Some differences is that [3] performs detection on different type of anomalies, objects in crowded scenes, and doesn't try to locate the anomaly in the image, rather classify anomalous scenes. The article [2] proposes a training technique to limit the autoencoder reconstruction capability, it has proven to be useful in cleaning the heatmaps and reduce the amount of false alarms, but it need a dataset with anomalies and their location, even if small, eliminating some advantages of the method. Using feature maps from a larger pre-trained model as a starting point, like [6], seems the technique that would help the most, the article [6] shows that all the considered models that uses pretrained CNN (like resnet18 or, in this case VGG19), leveraging hierarchical discriminative features, outperforms all the models based on autoencoders or GAN trained directly on raw data. The model used in [6] is composed of a deep pretrained network, VGG19, which does the hierarchical image feature extraction, each one of the 16 layers generate a feature map at a different scale, as its receptive field focus on a different region of the input. All the feature maps are then aggregated, averaged and concatenated to form a multiscale regional map, with each region having 16 different scales. This map is then passed to a convolutional autoencoder trained to reconstruct feature maps not original images, the anomaly score is then calculated on the reconstruction loss, and the anomaly positions are segmented. The autoencoder they use has only 1x1 convolution layers with a latent space dimension that adapts to the specific dataset calculated through PCA. This method named DFR, deep feature reconstruction, achieve very good results on the MVTEC AD dataset composed of multiple specific classes, with low variability and small anomalies, like holes or scratches, designed for industrial anomaly detection, so although the whole approach seems a significative improvement it should be tried on a dataset with more wide situations to prove itself effective for a problem like ours

that is slightly different, aiming to detect large objects or people. The curve they used, PRO-AUC, can be a very good metric to compare different experiments.
Another attempts could be to train a model with a more supervised approach, using automatic annotations based on physical trajectories within frames of a video, building tracklets that the model use to validate the detected bounding boxes.

Conclusions

The results show that the idea on which the project is based is solid, and it works despite the very simple model used and rather limited number of augmented data. This technique can be really useful as it allows to train the network on a dataset without anomalies, with normal events only, in a totally unsupervised way, with no need of manual annotation. Another advantage is that it can be used to detect a wide range of objects in various contexts. The Autoencoder had problem in reconstructing pool lanes and bright spots where the light reflects while it was quite good at learning the background texture of the water, this is in accrod to [5]: the model has learned general patterns first. That suggests that maybe a more compressive architecture trained on smaller images cut in the center of the pool could perform better, as in [1], and is compatible with what has been observed in [6] with regard to multiscale: They observed that low scale, very local, were not significant and very high global scale increased the efficience by only a small margin, with the exception of texture images, like wood, where the local features were already enough to represent everything normal and so allowed better detection of small anomalies. An use case knowledge of the anomaly sizes would be useful in dimensionating models feature maps to watch at exactly the right level. Another way would be to select only images without swimmers, limiting the use of inpainting, and also doing more data augmentation.

About the training modification tested, the noise method can be a good way to limit the autoencoder reconstruction capability, as proved by [5], eventually using the inverted pixels on part of the image where we want an higher loss, using the annotation of the dataset to invert region where there are swimmers and perform some epochs of negative learning. Something similar is described in [2] although applied to a different type of dataset. The detection system need to be improved too, maybe using another neural network system instead of openCV algorithms, or even just setting adaptive parameters in the existing script for different types of images.

Notes

[1] Creusot, C.; Munawar, A. , "Real-time small obstacle detection on highways using compressive RBM road reconstruction" IEEE Intelligent Vehicles Symposium (IV), 2015, pp.162-167, June 2012, doi:10.1109/IVS.2015.7225680

[2] A. Munawar, P. Vinayavekhin and G. De Magistris, "Limiting the reconstruction capability of generative neural network using negative learning," *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017, pp. 1-6, doi: 10.1109/MLSP.2017.8168155.

[3] Dimokranitou, Asimenia. "Adversarial Autoencoders for Anomalous Event Detection in Images." (2017).

[4] The Github repository of the project can be found at: https://github.com/Francesco-Carlucci/IST

[5] Devansh Arpit, Stanisław Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A Closer Look at Memorization in Deep Networks. ICML, 2017

[6] Jie Yang, Yong Shi, and Zhiquan Qi. DFR : Deep feature reconstruction for unsupervised anomaly segmentation, 2020