Politecnico di Torino

Master Thesis

Unsupervised Video Anomaly Detection using 3D CNNs and Novelty Detection techniques

Master of Science in Physics of Complex Systems

Thesis Advisors
Stiven Kulla
Prof. Andrea Bottino

Candidates
Adrián Khalil López Raven
Eugenio Marinelli

April 7, 2021

Acknowledgements

We thank here the people without whose contribution this work wouldn't have been possible. First, a special thanks to Stiven Kulla, our supervisor, for the huge support and for withstanding the pressure in our darkest moments, being able to come out strong and still provide us guidance. We would also like to thank the people at Addfor for providing us both the technical equipment necessary for our work, and the warmth of a community in these dire times. Thanks also to Professor Bottino for the support and pertinent comments throughout our work.

The following are the personal acknowledgements specific to each author:

Eugenio:

I would like to thanks my parents who strongly supported me throughout this master, both morally and economically. Secondly I would like to thank the many people i've met and exchanged thoughts, that are responsible for my growth in these years. Among them a special thank to the person that by far has influenced me the most, with whom I shared laughs and disillusions, my patner in this thesis, Adrian.

Adrián:

First I wish to thank my parents, Óscar and Elizabeth, for continually being endless sources of wisdom and love, for being two examples of formidable moral and psycological sophistication whose example I hope to live up to throughout my life (btw, thanks also for the money to stay here).

Secondly, I thank Stefano and Matteo, not only for the endearing and loving friendships, but for helping me unwind the knots of my mind in the darkest moments throughout the journey of this work.

And a final thank you must go to my partner Eugenio for whom I have the utmost of admiration. Your presence has been a never ending source of enrichment for thought and life.

Abstract

In this work we propose and test a method to perform video anomaly detection exploiting the power of deep convolutional neural nets to extract strong low dimensional representations to be used in conjunction with novelty detection algorithms.

Given the outstanding performances obtained in image and human action recognition using convolutional neural networks trained on large datasets, in our work we used a 3D convolutional neural network pretrained on the Kinetics 700 and UCF101 datasets as a feature extractor, to map snippets of surveillance videos into a lower dimensional embedding space. We then tested whether snippets of anomalous events are mapped into statistical outliers in the embedding space, which can be detected using novelty detection algorithms.

Using our technique on the UCF Crime dataset we got a final AUC score of 74.8 % which is comparable to the 75.4% obtained by the creators of the dataset, Sultani et al. [1] demonstrating the validity of our approach.

Acronyms

List of the most used acronyms in the following work:

- NN: Neural Network
- CNN: Convolutional Neural Network
- FCNN: Fully Connected Neural Network
- TL: Triplet Loss
- AE: AutoEncoder
- GAN: Generative Adversarial Neural Network
- SCL/ SupConLoss: Supervised Contrastive Loss [2]
- CEL: Cross Entropy Loss
- AUC: Area under the Roc Curve
- LOF: Local Outlier Factor

Contents

1 Introduction

In the last decade with the advent of deep convolutional neural networks (CNNs) the field of computer vision has been completely revolutionized. Thanks to their high flexibility, they have been massively used in almost every task that concerns image or video processing. A way to interpret the function CNNs play is that they work as highly sophisticated and automatized feature extraction systems; they allow to embed high dimensional data into a lower dimensional dimensional space (see fig 1.1) that still contains semantic information useful for the task at hand.

Throughout this work we propose and test a new approach to tackle the problem of video anomaly detection leveraging CNNs' aforementioned capacity to extract meaningful features and pairing it with novelty detection techniques.

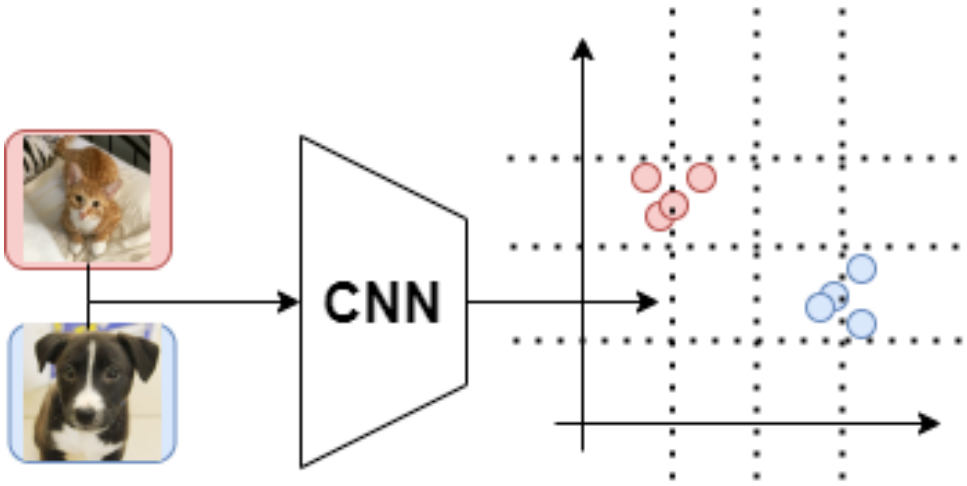


Figure 1.1: A pictorial representation of a CNN embedding images into a 2 dimensional space.

Video anomaly detection is the task of individuating within a stream of footage, segments of video that are considered anomalous. More specifically in this work we tackle the task of temporal anomaly detection which consists in finding what frames are anomalous; this in contrast to spatial anomaly detection which aims at finding where inside the frame the anomaly happens.

To tackle this problem the first step in our approach is to have a feature extractor that maps

the potentially anomalous videos in an embedding space that contains information about the human actions contained in them. This is accomplished by using as a feature extractor a 3D CNN trained on human action recognition. Here, we assume that anomalous videos contain specific actions clearly distinct from those found in normal videos, and that the feature extractor maps different actions into different regions of the embedding space. In that way by projecting the videos into this space, the anomalous ones should end up being isolated from the rest.

The proposed approach works then as follows: First, we assume to have a long stream of normal footage, i.e. with no anomalies in it. This stream is divided into snippets which are mapped into an embedding space via the 3D CNN which has been previously trained on an action recognition dataset. These embeddings will form then the "Normal Clusters" in this space which, thanks to the pretraining on action recognition, should contain information about the normal actions/activities that take place in the videos. Having defined these normal clusters, on inference time one maps the video snippets to be evaluated into the embedding space and through novelty detection techniques (in our case using the Local Outlier Factor algorithm), one determines whether the new embeddings belong to any of the normal clusters or not, classifying as anomalous those that don't. For this reason we consider ours an unsupervised method, in the sense that we don't explicitly tell our model what an anomaly "looks like", instead we tell it what normal "looks like" and then an anomaly is defined as that which doesn't look normal. This means that we provide a normal distribution in embedding space and an anomaly is then defined as a statistical outlier from this distribution.

The anomaly detection system itself consists of the following parts:

1. Feature Extractor: This is used to map the surveillance videos into the lower dimensional embedding space. In our case we used a 3D CNN trained on an action recognition dataset.

2. Novelty Detection Algorithm: With it, first the system learns what normality means during its training in which it sees the embeddings of Normal videos. Then we make our trained algorithm predict the anomaly score of normal and anomalous embeddings.

1.1 Relation with previous works

Our method draws inspiration mainly from the success of the reidentification techniques found in [3] where CNN where used to map photos of people in a hidden embedding space. A cluster is associated to each person. Then, when a new unseen photo is given to the model, the system finds the nearest cluster to the new embedding, associating the respective person to the embedding.

Our idea is analogous since our method is also based on the idea that a 3D CNN pretrained on action recognition should be able to cluster videos with similar actions into regions of the embedding space, and the anomalies woud then be outliers to the so formed clusters.

In the field of deep video anomaly detection there are also two approaches that bear some resemblance to our own:

The Auto Encoder (AE) approach: This method, explored in works like [4] and [5], makes use of the AE architecture in its usual way; it uses a neural net encoder to compresses an input into a latent embedding space and then decompresses it using a decoder neural net. To use as an anomaly detection system they repurpose the reconstruction error to be an anomaly score. These kinds of systems are trained by making them reconstruct normal video footage of the setting.

The GAN approach: The philosophy of this method is similar to the AEs and is explored in works like [6]. It also consists of an encoder that compresses the frames of the video into a latent space, and of a decoder that afterwards tries to reconstruct it. The encoder/decoder structure works as the generator of the GAN. The discriminator then sees both the original and the reconstructed frame and has to learn to recognize the original one. The generator and the discriminator are then pitted against one another in regular GAN fashion. This system is also trained by reconstructing normal video footage.

The similarity between our approach and the two just mentioned is that they all encode information from the video into a latent embedding space. Some differences that we find with respect to these approaches are the following: First, we don't reconstruct the frames from the embedding space instead we do the anomaly detection directly in it. We suspect this to be an advantage since reconstructing the frame is a complex task to perform and might suppose and unnecessary burden for the anomaly detection task. Second, their neural nets are trained to reconstruct normal video footage of the setting in which to do anomaly detection, while ours is trained on Kinetics700. This could be both an advantage and a disadvantage. The disadvantage would be that there is a domain gap between the videos used to train our feature extractor and the videos on which the anomaly detection is performed, while the advantage is that our net learns to extract more abstract and meaningful features given that it is trained on a harder and more sophisticated task (classifying human actions). A third advantage is that the neural net of our feature extractor doesn't need training on the surveillance videos. Training effectively a neural net can be time consuming and computationally expensive and their approaches need to do so for every new setting on which they're implemented while our net is only trained once.

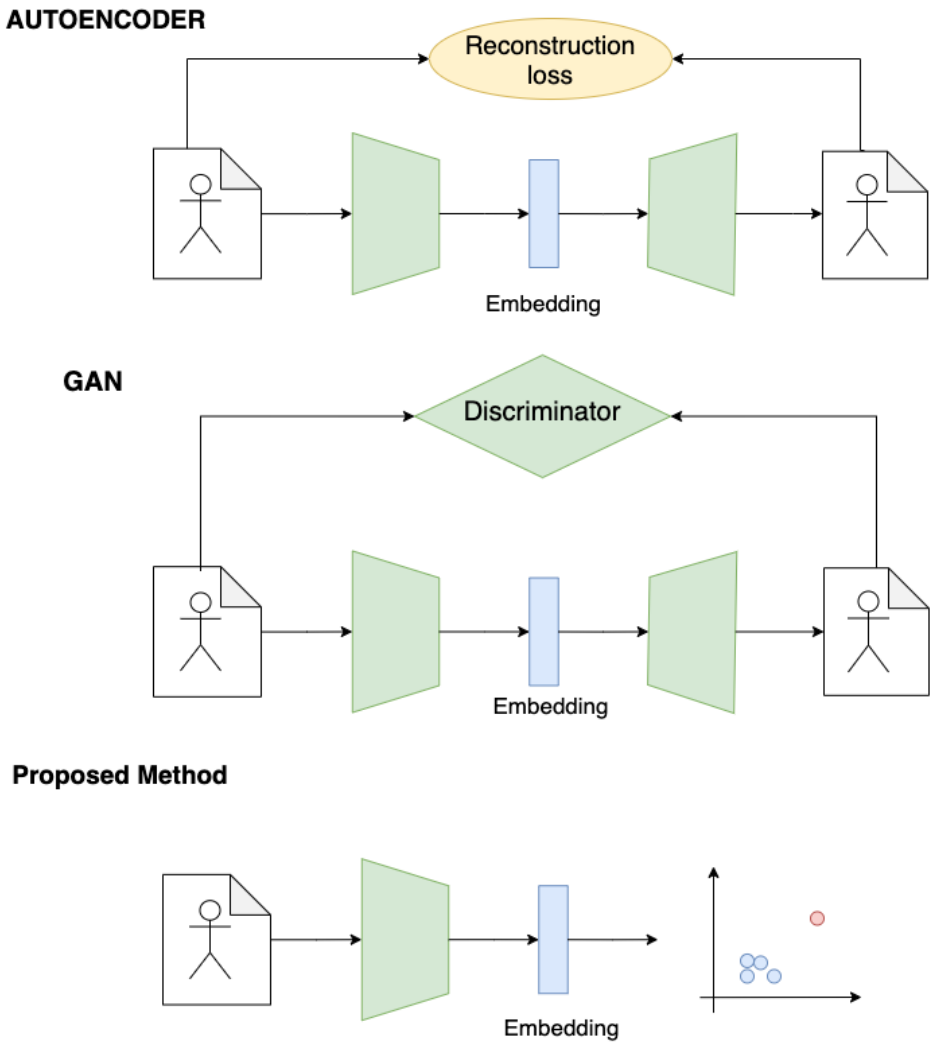


Figure 1.2: Illustration of the resemblances and differences between our proposed method and previous attempts with AutoEncoders and GANs.

1.2 Structure of the work

The structure of this work is the following:

1. **Theoretical Framework**: In this chapter we talk about theoretical concepts that we used for our work, dividing it in three parts. We start by dissecting the feature extractor. In section 2.1.1 we give a brief introduction to neural networks while in 2.1.2 and 2.1.3 we explain in more detail the kind more fit for our task, the convolutional neural networks, known as CNNs, and the reason why they proved to be so successful in computer vision tasks, highlighting also (sec 2.1.4) the contribution of large datasets and data augmentations for this deep learning revolution.

   In Sec 2.2 we explain the theoretical details of the strategies we used to tune the feature extractor and improve its resulting embeddings. We also provide brief theoretical explanations of the different losses used for this task and of the silhouette score, the metric used to measure the quality of the embeddings.

   In the third part we explain in detail the algorithm that we used to do anomaly detection. Since an important part of our work is focused on choosing the dimension of the embedding space, in sec 2.3.1 we discuss the curse of dimensionality, that is an evocative name that refers to some counter intuitive phenomena that prevent an effective anomaly detection in very high dimensional spaces.

   We then proceed to explain in depth the Local Outlier Factor in sec 2.3.2, the algorithm that we used to do the anomaly detection.

2. **Experimental Part**: In this chapter we gather and discuss all the experiments that we performed to validate our model. In sec 3.1 we compared the performance of two metric learning losses mentioned in the Theoretical Framework, triplet and contrastive, using the silhouette score as a performance measure. These tests were performed on image datasets that are less computationally expensive.

   We then went on to build and test the feature extractor for our anomaly detection system. The feature extractor has two components, the backbone (a 3D CNN) and the projection head which consists of a single layer fully connected neural net that takes the output of the backbone and projects it into the final embedding space of our system. To choose the backbone we reviewed different pretrained architectures available on the internet, finally opting to use a 50 layer Resnet(2+1)D. We tested several projection heads, changing specifically two aspects: changing the output embedding dimension (sec 3.3.3) and changing the loss (sec 3.3.4) used to tune them. The performance comparison of the different models was done using the silhouette score obtained in the tuning on the action recognition dataset, and not yet

on an actual anomaly detection task.

With the feature extractor ready, in this last section we put our system to the test in an anomaly detection task. Here, in sec 3.3.1, we first introduce the dataset that we used for this purpose, UCF Crime, and explain the particular way in which we used it.

In these experiments again we compared performances changing the feature extractor along the same domains tested during its tuning: changing embedding dimension and loss function. This time though, the comparison is done comparing their performance on the anomaly detection task, more specifically, we compare the AUC scores obtained with the different feature extractors.

In sec 3.3.3 we compared the performance of the feature extractor in the anomaly detection task changing the dimension of the embedding space. Through this experiment we discovered strong evidence for the dependence of the anomaly detection performance on the dimension of the embedding space, so care must be taken in choosing this hyper parameter.

We also studied how the different losses influenced the anomaly detection performance in sec 3.3.4 but these results were nevertheless inconclusive. In the end we found a good performing anomaly detection system using a randomly initialized projection head, this suggests there were problems in the tuning of the projection head (we suspect associated to the UCF101 dataset) but we got results comparable to others in the industry suggesting the system has potential left exploit.

3. **Conclusions** We end this work by summarizing the most important results found, pointing out the direction for further research and improvement.

2 Theoretical Framework

This chapter is devoted to the theoretical explanation of the concepts that we used throughout the work. The chapter is divided into three sections:

1. Feature Extraction: In this section we explain explain some of the different technologies underlying the neural net we used as our feature extractor, a Resnet50(2+1)D.
2. Creating good embeddings: In this chapter we explain how we trained our feature extractor in order to guarantee a good performance when used in conjunction with the anomaly detection algorithm, and we explain the theoretical details of various losses used during said training.
3. Anomaly Detection: In this section we will talk about what anomaly detection algorithms are, and in particular a subclass of these, novelty detection algorithms, which ended up being the most fitting to address our task. Also, we introduce the specific algorithm that we used in this work: the Local Outlier Factor.

2.1 Feature extraction

In most machine learning problems, the data under study live in very high dimensional spaces. Even in our case a short 10 second 128x128 video is represented as an array with a whopping 14.745.600 entries (128 x 128 x 30fps*10s x 3). Applying classical machine learning techniques (like anomaly detection algorithms) directly in such high dimensional spaces is a hopeless endeavor, and so methods to reduce the dimensionality while still preserving meaningful information are necessary. In the early days this lead practitioners to handcraft features out of the raw unstructured data.

In the field of computer vision though, a technology completely revolutionized this process: Convolutional Neural Nets. Through their convolutional layers, CNNs not only completely automated the feature extraction process, but they also did so in a way that allowed to break all performance benchmarks in the computer vision tasks of the time.

The feature extractor of our anomaly detection system belongs to this family of networks.

It's a 50 layers deep Resenet(2+1)D followed by a fully connected projection head, and the next sections are devoted to explaining the different technologies that underlie this architecture, starting from it's simplest predecessor: the fully connected neural net.

2.1.1 Neural Networks

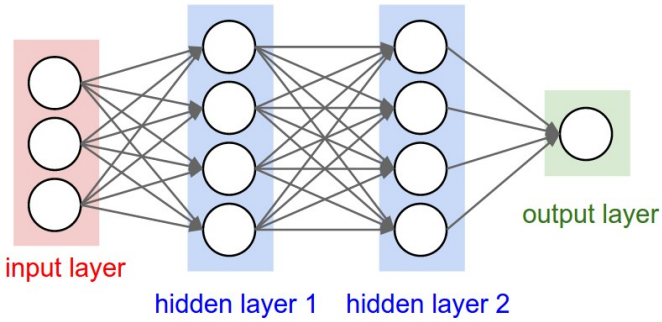


Figure 2.1: Pictorial representation of a fully connected neural net. (Image taken from Stanford's CS231n course on convolutional neural nets)

Traditional feed forward neural networks can be thought of as a sequence of layers of neurons (see figure 3.1) where the first layer is the raw input data (in our case it would be the pixel values of a video) and subsequent layers, called hidden layers, extract ever more abstract information of the data-object until one reaches the final layer, called the output layer.

Internally, each neuron in the net has a value associated to it, called an activation, which depends exclusively on the activation values of the neurons of the preceding layer. More specifically, the activations of the neurons in the l-th layer, $x^{(l)}$, are a linear combination of the values of the preceding layer, $x^{(l-1)}$, passed through a non linear function σ (usually a ReLU function):

$$x^{(l)} = \sigma(W^{(l)}x^{(l-1)} + b^{(l)}) \tag{1}$$

where $W^{(l)}$, the weights matrix, and $b^{(l)}$, the bias vector, are parameters whose values are to be tuned depending during the training stage of the neural net.

During said training stage some training data is passed through the NN and the output is passed to a loss function. This function associates a loss or cost to this final representation of the data, and the neural network "learns" to minimize said loss by tweaking the weight and bias parameters using gradient descent or some other gradient based optimization algorithm.

The gradients can be calculated efficiently by using the backpropagation algorithm. Backpropagation is a dynamic programming algorithm introduced by Rumelhart, Hinton and

Williams in [7] which computes the gradient of the loss with respect to each of the NNs parameters in a recursive way down the computational graph of the network.

Early theoretical results like the "universal approximation theorem" [8] seemed to hint this type of systems could have great potential. This theorem states that just with a 2-layer neural network, by making the hidden layer arbitrarily "tall" (increasing its number of neurons), one could always find parameter values such that the NN could approximate any non-linear function. In reality though, the theorem doesn't prescribe how to find said values, which makes the result not useful in practice.

So in the end *tall* didn't make perfect, it turns out though, increasing *depth* did do the trick. Fully connected nets though, posses a bottleneck that prevents them from being too deep: they have too many learnable parameters. As mentioned before, in a fully connected neural network each activation neuron can attend to every neuron from of the preceding layer. To do this it requires the W matrix which maps a d-dimensional vector into another d'-dimensional vector, meaning W has dxd' parameters, so if you have a high dimensional data such as an image or a video, deep fully connected networks quickly become computationally unfeasible to train.

To solve this problem people came up with Convolutional Neural Networks (CNNs) that by making use of good assumptions about the structure of images, they require far fewer parameters to train.

2.1.2 2D Convolutions

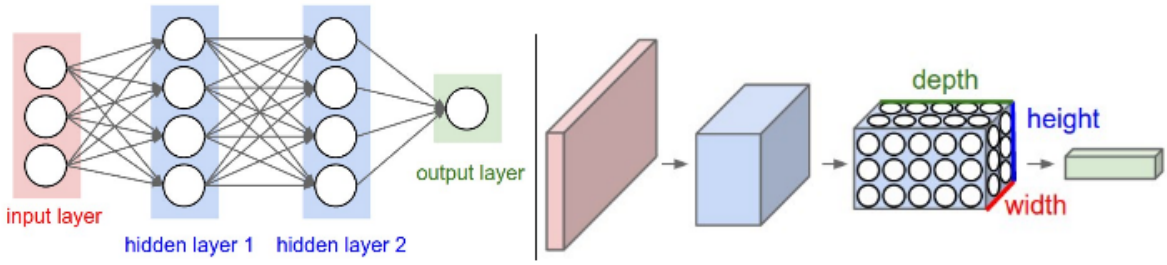


Figure 2.2: Pictorial representation of a fully connected neural net (left) and a convolutional neural net (right). (Image taken from Stanford's CS231n course on convolutional neural nets)

2D CNNs are a type of neural networks in which the activations of neurons in the next layer are not influenced by every neuron in the precedent layer but only on a subset of these.

Their workhorse, the convolutional layer is similar to their fully connected counterpart in that it relates a set of input neurons to a set of output neurons through an affine transformation followed by a non linear operation.

In CNNs this process is mediated through what its called a filter, that is a tensor of learnable

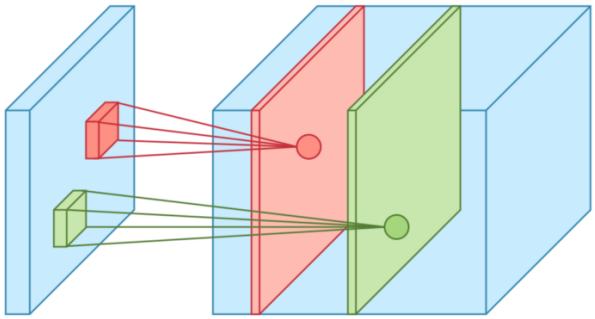


Figure 2.3: Different filters look for various patterns in images creating different feature maps

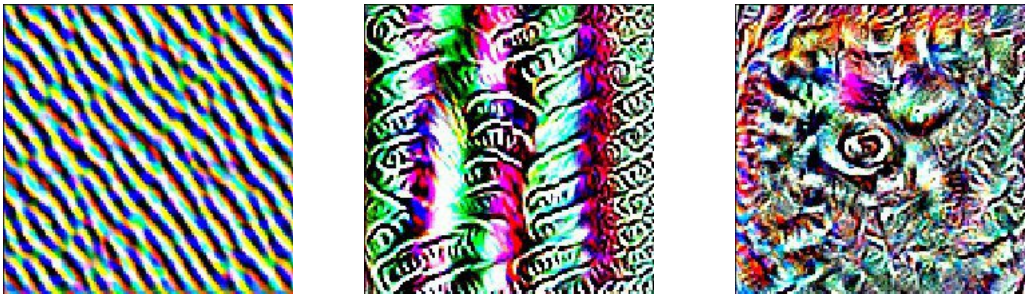
parameters, which slides over the two dimensions (hight and width) of an image performing a convolution, that is a weighted sum between the values of a tensor image and the weights of the filter. The resulting tensor of activations is called a feature map and it expresses, with high values, the areas where the filter overlaps the most with the original images. So if the filter represent a particular kind of edge, meaning that the high values neurons in the filter form an edge, the resulting feature map from the convolution on a image will highlight the particular zones where that edge is present in the image. A convolutional layer is composed of different filters that creates different feature maps (fig 2.3)

This filter-based approach in the processing of data has some peculiarities that makes this kind of deep models very well suited for computer vision tasks:

1. Locality: In convolutional neural nets, the fact that feature maps of consecutive layers are connected through this convolution operation means that neurons are only allowed to look at a small patch of the neurons (sometimes referred to as the receptive field) from the preceding layer. This, in contrast with the fully connected case, in which the neurons are connected to all of the neurons from the preceding layer. This property finds correspondance with the fact that in an image the correlation between pixels is extremely local. In most computer vision applications, for example, we would like a model that knows to detect an eye independently of the content of far-away pixels.

2. Translational invariance: Since the weights of the filters are the same during the convolution across the image, the detection of patterns does not depend on where that particular pattern is situated in the image. This gives a lot of flexibility to the model since usually different viewing angles and positions do not change the semantic of the image. If our model aim to detect the presence of cats in an image, the position of the cat in it is irrelevant.

CNNs are hierarchically structured in a series of consecutive convolutional layers with different purposes. In each layer the feature maps expressed by the filters in the previous layer are convolved by another set of learnable filters and this process continues till the last convolutional layer that signs the end of the backbone of the convolutional neural network.



(a) layer 7 (b) layer 74 (c) layer 168

Figure 2.4: Visualization of trained filters of a CNN Resnet 50 on different layers from [9]. The deeper the filter is the more complex and abstract patterns it can capture: in early layers (fig. 2.4a) filters look for simple lines and thus they can captures more texture features. Going through the middle of the layers (fig. 2.4b) the filters look for more complex patterns and even some flower-like objects can be recognized in the lasts layer (fig. 2.4c)

One convolutional layer of a CNN is made of several of filters. During trainig these filters specialize in the detection of one specific feature of the image. Along the networks the specialization of the filters follow a hierarchical order: filters in the firsts layers learn to detect simple patterns like lines and edge and various textures while later filters detect more abstract and complex patterns like eyes, teeth, doors depending on the dataset they're trained on. This is also due to the fact that deeper layers have a larger view on the original image than the shallower ones: if the first filters look at a 3×3 portion of nearby pixels of the image, in the second layer, by sliding another 3×3 filter on the feature maps of the first layer, it will look at a 9×9 portion of the original image making it possible for those filters to learn more global features of the image.

2.1.3 3D Convolutions

The same idea for 2D images can also be applied in video to extract motion features The first attempt in this area was made by Ji et al. [10], where they trained a 7 layer 3D convolutional network.

Their convolutional layer consists of a filter that is a 3 dimensional tensor (Time, Hight and Width). While in 2D CNN filters scan the image in on a plane, the filter in 3D CNN scans the time dimension as well. This means that the filter not only will recognize spacial features like edges and lines, but also features linked to actions, making the 3D CNN a very powerful tool for action recognition.

At that time memory and data constraints strongly limited the depth of these kind of architectures, but after the breakthroughs brought by the AlexNet [11] and ResNets [12] and more vast datasets like Kinetics, researchers have tried to apply deeper neural networks to the task of human action recognition. With the hope of replicating the success of convnets in image recognition one obvious path to follow is to inflate existing 2D models as in [13] to

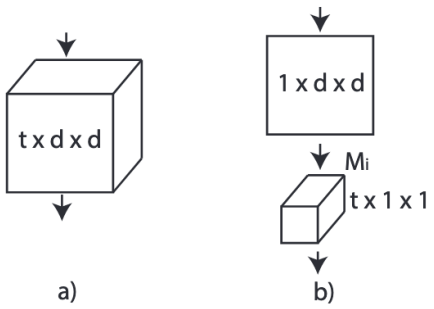


Figure 2.5: Image taken form [15]
a): standard 3D convolutional layer composed by a single 3D filter
b): (2+1)D convolutional layer decompose the convolution into 2 parts, in which the first perform the convolution along space dimensions and the latter along the time one

create 3D resnet [14] and that brought advancements in this area.

In another work by Du Tran, Yann LeCun et al. [15] they tried to factorize the time convolution and the spacial one in what they called "(2+1)D" convolution. This means that the convolution over the video frames is done in two steps: First, a filter of dimensions $1 \times \text{height} \times \text{width}$ perform the spacial convolutions over the frames, then a non-linear function creates the feature maps that are processed by a second filter of dimensions $\text{time} \times 1 \times 1$ and passed again to a non linear function like ReLu (see fig. 2.5)

They noticed that by factorizing the 3D convolution into two separate and successive operations one can double the number of non linearities making the model capable of representing more complex functions and making it also easier to optimize.

The Resnet(2+1)D that we use in this work is based on this type of convolutional layer and reaches SoA results on classification on Kinetics 700, but we will discuss more about this in sec. 3.2.1.

2.1.4 Large Datasets and Data Augmentation

CNN revealed themselves to be very powerful and versatile tools for computer vision tasks. Nevertheless, as has become current theme in deep learning, their potential was only truly expressed when huge datasets were created. In tasks involving images (like image classification and object detection) a landmark happened in 2009 with the creation of Imagenet which when released had more then 10 million images and 10.000 categories and has been continually expanding, counting nowadays with more then 14 million annotated images of 21.000 different categories.

Training on big datasets like Imagenet addressed two major problems in machine learning: overfitting and generalizability.

Overfitting happens when the accuracy of a model trained on a dataset drops consistently when new unseen data is shown. This failure often happens when the model is overly complex (i.e. has too many parameters as it's common in deep learning) and few training data is available. The problem of generalization is related to the one of overfitting, but while overfitting refers to the performance on unseen *data*, the generalizability of the model refers to the its capability to have good performance in different *tasks* in which it has not been directly trained. To solve both problems a good model should look for the right patterns in the data that generalize well enough to different settings and tasks, and training on large datasets was found to be crucial for achieving this [16].

Needless to say, Imagenet revoluzionized the field of image classification, object detection and image segmentation, and efforts were made to replicate the same success in the task of human action recognition following the same steps [17]. In 2017 kinetics 400 was announced and it was formed by more then 16.000 different 10 second videos divided into 400 different human action classes. In its latest update kinetics counted with 700 different classes of actions and 650.000 different videos [18].

As larger datasets were forming, deeper data-hungry CNN models started to appear which required even more training data, but manually increasing the size of datasets is a labor intensive task, and so new techniques were devised to "artificially" increase the available training data.

These techniques are called "data augmentations" and they consist of slight modifications of the data in a way that changes its appearance but preserves its semantic content. Thanks to these tricks the model can see many more samples than there's present in the dataset, leading to longer epochs of training and, at the same time, strongly limiting the risk of overfitting. In image classification it's worth mentioning three data augmention that stood out for their importance in the improvement of the performance: Random Crop of the image, Flipping and Rotations [2] [19].

In this thesis we worked with videos and we made heavy use of data augmentations techniques. We noticed it was risky to use random crop in videos since we could possibly cut important parts of movement, so we resolved to use others types of data augmentations. Together with flipping and 10 degrees random rotations, we also used the time dimension to augment the data, by preforming:

1. Time crops: Video samples of the training set are in general too long (timewise) to be fed directly to the neural net and so a random portion of the video is chosen each time it is loaded into a batch.

2. Randomly fixing the frame rate: the frame rate is chosen randomly, making the videos faster or slower.

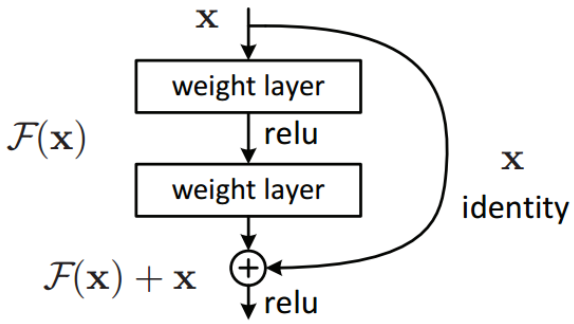


Figure 2.6: Residual connection between layers of a neural network. Image taken from [12]

2.1.5 Resnets

The architecture that we used in this work comes from the Resnet family, that in the last few years became one of the standards of the industry.

The most popular convolutional models today are Resnets. Introduced for the first time by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in 2015 [12], they won several Imagenet competitions and showed that training very deep models was possible.

Before Resnets, very deep models were very hard to train because of the problem of vanishing gradients. What used to happen is that as one backpropagated further back through the net, gradients would become smaller, to the point where deepest layers weren't able to modify their parameters and so the early layers could not learn. In order to solve this problem Kaiming He et al. proposed to insert in the model a "gradient highway", that is, shortcuts to route more efficiently the gradients to the early layers.

The main component of the Resnet is the "residual block", which is a sequence of convolutional layers (called the residual layers) where the output is summed to their input, in what it's called a "skip connection". If we consider the neural network as a computational graph whose gradient flows from the last layer to the early layers during backpropagation, the optimization is made easier since each gradient can use these the skip connection to reach deeper layers.

The concept of the residual block is expressed with equation 2:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{2}$$

where $\mathcal{F}$ is the function representing the residual neural network and $\mathbf{x}$ and $\mathbf{y}$ are the input and the output of the block.

The skip connection made possible to have unprecedentedly deep convolutional networks that brought state of the art results in a wide variety of computer vision tasks.

For resnet architectures with 50 layers or more, the standard resnet block is replaced by a stack of 3 convolutional layer: the first layer is composed of 1×1 filters that simply reduce the number of feature maps, the second layer is formed by 3×3 filters and the last 1×1 filters restore to the input to its original number of feature maps. The advantage of this block is that it increases the depth of the block without increasing its time complexity.

2.1.6 The Architecture of our Feature Extractor

The feature extractor that we used for our task of anomaly detection on video is a Resnet 50 (2+1)D which consists of 16 bottleneck residual blocks stacked one after the other each containing one (2+1)D convolutional layer, plus a final fully connected layer which we denominate the projection head.

2.2 Creating good embeddings

In this work we intend to make use of convolutional neural networks as feature extractors that embed very high dimensional video data into a lower dimensional space that contains enough information so that an anomaly detection algorithm can recognize embeddings associated to anomalous videos.

But what information should the feature extractor distill in order to isolate the anomalies? In our work we assume said information should be pertaining to human actions , and so we trained our feature extractor on human action recognition tasks in order for the extracted embeddings to contain this information.

If one wishes for the feature extractor to generalize well, however, than one should train train on a huge dataset, and since in this case the datasets would consist of videos which use a lot of memory, the training would require considerable amounts of computing power. To give a sense of the amount, we estimated that using the 2 GPUs available to us to train a classifier on the kinetics 700 dataset would require 200 days for 100 epochs of training. Luckily, researchers and companies that have the required computing power have made their trained models available on the internet so that others can exploit and have access to powerful CNNs without having to go through the cost of training them.

These pretrained models are usually used as a backbone of a larger neural net which has a couple further fully connected neural nets which allow to adapt the entire neural net to the task at hand. In our case, the available backbones had to high of a final embedding dimension (we delve more deeply into problems related to dimensionality in Section 2.3.1) and so the further fully connected layer was necessary to project into a lower dimensional space. The task of training this projection head we will call tuning throughout our work.

2.2.1 Tuning the projection head

We want for the head of our feature extractor to further project the output of the backbone into a lower dimensional space without losing information pertaining to human actions. For this purpose we trained it in another human action classification task. For this, the standard loss to use is the Cross Entropy Loss which we explain in detail in section 2.2.1.1.

A classification task is usually not the best for training a feature extractor that will be used in conjunction with clustering techniques. In our case this would be because anomaly detection algorithms use specifically the notion of *distance* to asses whether a sample is anomalous or not, since samples which are *far* from the normal ones are considered *anomalous*. Feature extractors trained on classification tasks on the other hand, learn to partition/divide the embedding space into semantically meaningful portions (to be able to associate a label to

each portion) but they don't specifically encode the information into the *distance* i.e. they encode the information in the topology, not in the metric. So for the anomaly detection algorithm to find anomalous videos, it is necessary that our feature extractor guarantees that videos which are semantically similar are also close in embedding space:

$$\text{Video1 and Video2 are similar} \iff \text{Embs1 and Embs2 are close} \tag{3}$$

There exists various losses designed to enforce this correlation between distance and semantic similarities, which are called metric learning losses. We will explain the two losses of this kind, Triplet and Contrastive, in sections 2.2.1.2 and 2.2.1.3 respectively.

Finally, as a performance metric of the property stated in eq 3 we used the silhouette score, which we explain in section 2.2.1.4

2.2.1.1 Cross Entropy Loss

To train a model for classification, Cross Entropy is the "go to" loss. The Cross Entropy Loss (CEL) assumes that, when you give a sample to your neural net, the final output are scores that tell you how strongly the neural net believes that your sample belongs to a class. The CEL then associates a high loss when the "belief" doesn't match the actual class of the sample. Calling the scores s_i and the actual class y, the loss is calculated as follows:

$$L_{C.E.}(\{s_i\}, y) = -s_y + \log\left(\sum_{i \in classes} e^{s_i}\right) \tag{4}$$

though to understand it's meaning it must be cast into a more illuminating form:

$$\begin{aligned} L_{C.E.} &= -\log\left(\frac{e^{s_y}}{\sum_{i \in classes} e^{s_i}}\right) \\ &= -\sum_{j \in classes} \delta_{j,y} \log\left(\frac{e^{s_j}}{\sum_{i \in classes} e^{s_i}}\right) \\ &= -\sum_{j \in classes} p_j \log(q_j) \end{aligned} \tag{5}$$

In this expression q and p can be interpreted as probability distributions:

- $q_i = \frac{e^{s_i}}{\sum_{j \in classes} e^{s_j}}$; is the probability that the sample belongs to class i according to the neural net, and is computed by applying a softmax over the activations, s_i, of the last

layer.

- $p_i = \delta_{j,y_i}$ represents the probability distribution we hope the neural net would output, i.e. the target probability distribution of the sample belonging to class i.

and equation (5) is called, in information theoretic circles, the cross entropy between p an q.

Given a random variable $I \sim p$, the information obtained by drawing the value i, written as $\mathcal{I}_p(i)$, is defined as:

$$\mathcal{I}_p(i) = -\log(p_i) \tag{6}$$

The cross entropy then measures the average information on a distribution q of a random variable drawn from a probability distribution p and it's defined as:

$$H(p, q) = -\sum_i p_i \log(q_i) \tag{7}$$

This definition has some traits that makes it a suitable candidate for a classification loss: The function is smooth on the q_i s, and has high values when the probability distribution of the predicted label q_i is different from the desired probability distribution p_i. It can be proven that the cross entropy serves as a sort of distance function between the 2 distributions which assumes its lowest value when $q_i = p_i$, $\forall i$.

So minimizing the cross entropy in (5) means we're making the probability distribution q_i to be the same as the desired probability distribution p_i.

2.2.1.2 Triplet Loss

The Triplet Loss is a loss that has been used quite successfully in the task of person re-identification.

The Triplet loss assigns a value to a triplet of samples which are denoted as:

- The anchor: a sort of reference sample against which the others are compared.
- The positive: a sample that belongs to the same category as the anchor.
- The negative: a sample that belongs to a different category than the anchor's.

The aim of the TL then, is to pull together the anchor and the positive, and to push away the anchor from the negative. More formally, it associates the following value to the triplet of samples:

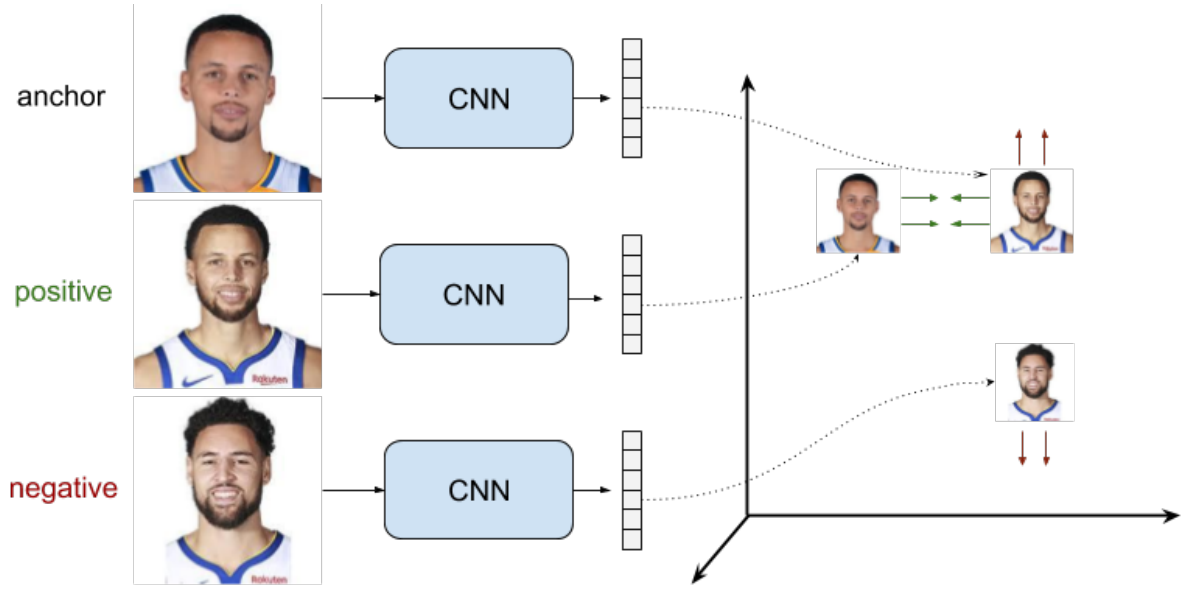


Figure 2.7: Diagramatic representation of the use of the Triplet Loss in person re-identification.

$$L(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0) \tag{8}$$

Where A is the embedding of the anchor, P is the embedding of the positive, N is the embedding of the negative, d is a distance/similarity function in the embedding space and α is a tunable hyperparameter. From expression (8) we can see that in order to minimize the loss, on must minimize distance between the anchor and the positive while maximizing the distance between the anchor and the negative.

For the TL to be effective though, some care must be taken in choosing the triplets. If for a given anchor one simply chooses a random positive/negative, if by bad chance the positive is already close and the negative already far, the loss value will be zero and it won't contribute to the learning. It turns out this happens more often that one might expect and so to address this, people resorted to hardmining the triplet.

Ideally hardmining would assign to every anchor the furthest away positive and the closest negative, in that way it would maximize the loss of the triplet, in turn increasing the value of the gradient and guarantying that some learning takes place. The problem with this though, is that to build the triplet you would have to go through the entire dataset to find the ideal positive/negative, which is too computationally expensive, and so the most common compromise is to restrict this search to samples within the same batch, in this way you get reasonably good triplets without the cost of searching though the entire dataset for each one.

2.2.1.3 Supervised and Self-Supervised Contrastive Loss

As the triplet loss, the supervised contrastive loss [2] also aims at pulling together clusters of points in the embeddings space that belong to the same class and push apart those that don't.
The main differences with the aforementioned triplet loss is that, instead of considering triplets, all the elements in the batch are compared with the anchor.
The technique draws inspiration from its self-supervised counter part [20] which we also use throughout our project, in which the positive is only the augmented version of the anchor and all the remaining elements of the batch are negatives.

Both types rely on the concept of multiview batch, that is a batch that contains a set of samples and additionally augmented versions (the "views" of the samples) of these, i.e. copies of the samples to which some augmentation transforms has been applied.

Given I the set of indexes in multi-view batch, $P(i)$ the set of the indexes of the positive instances with respect to i, the expression of the supervised contrastive loss function is the following:

$$\mathcal{L} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_p / \tau)}{\sum_{j \in I / \{i\}} \exp(\mathbf{z}_i \cdot \mathbf{z}_j / \tau)} \tag{9}$$

The self-supervised contrastive loss has the same expression but the set $P(i)$, instead of considering all the instances with the same label, contains only the augmented version of i, as shown in fig 2.8.

With respect to the triplet loss, it has the advantage of having the hard mining built into it, since every anchor is compared to all the instances in the batch.

2.2.1.4 Silhouette Score

One metric to measure the quality of our embeddings is the Silhouette Score and we use it throughout our work as a quantitative proxy of the clusterization of our embeddings. The silhouette score, given a reference sample, it tells you how close it is to same-labeled instances in comparison to those with different labels. To do so it calculates the average distance between the reference sample and elements with a different label, and compares this with the average distance to the other samples of the same class. More specifically it calculates the mean interclass distance of the closest neighboring cluster and substracts from it the mean intraclass distance, finally normalizing this number to give a score between 1 and -1. The expression to calculate it is:

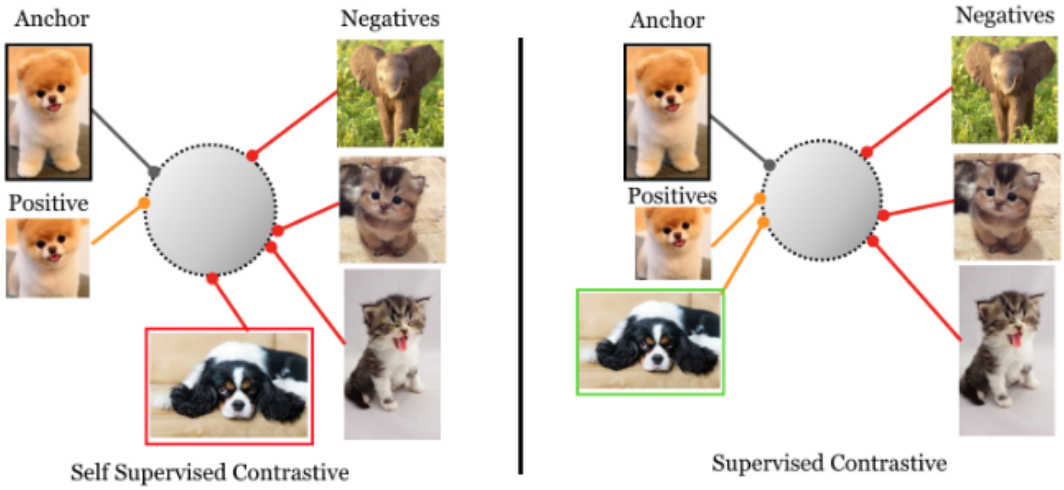


Figure 2.8: Self-Supervised Contrastive Loss vs Supervised Contrastive Loss. The latter takes as input also the labels and try to minimize the cosine distances between the the anchor and the positives, that are the augmented views of the anchor and the same-class instances, while it also maximize the distances with all the others

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{10}$$

where

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \quad \text{and} \quad a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j) \tag{11}$$

2.3 Novelty Detection

Novelty detection algorithms are a class of machine learning algorithms, most used in data mining for tasks like network intrusion detection, that focus on the search of *novelties* that are a type of outliers that occur after the training phase is completed [21].

The task of novelty detection consists of, given a set of normal instances, discriminating whether a new unseen piece of data belongs or not to the same distribution as the normal training instances. In other words, the algorithm tries to determine whether a new piece of data represents an outlier where, using the definition by Knorr and Ng [22]:

"An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism"

Some important algorithms used for novelty detection are:

- One class SVMs [23]:
  One class SVM is a variation from a classical kernel-based Support Vector machines that try to find the smallest hypersphere that include all the training data. During inference a data point lying outside this hypersphere is considered as an outlier.

- Isolation Forest [24]:
  Starting from the assumption that anomalies are isolated and lies on a low density region of the space, this machine learning algorithm associate for each point an anomaly score not using any concept of distance, but instead a value that relates on how difficult is to seperate one specific point from all the others.

  For each data x the isolation forest algorithm constructs an ensemble of tree graphs (also called forest) in which each tree node partitions the space until x is isolated from all the other points. in this frame of reference, outliers that live in low densities part of the space will be easier to isolate, so the average tree length (that measures how many steps on average it is required to separate one point from all the others) will be shorter. The anomaly score for a data point x among n total points, is:

  $$s(x, n) = 2^{-\frac{\mathbf{E}(h(x))}{c(n)}}$$

  where $\mathbf{E}(h(x))$ is the average depth of each binary search tree, and $c(n)$ is a normalization factor that depends on the number of data points n

- Local Outlier Factor [25]:
  Starting from the same assumption of Isolation Forest that anomalies are sparse, living in low density region of the space, this method finds local outliers that exhibit a lower density than their neighbors. This is the approach used throughout our work and it's

explained in more detail in Sec. 2.3.2

For each of these methods, one crucial factor that needs to be controlled for an effective novelty detection is the dimension of the space in which the analysis is performed. The risk of choosing an embedding space too small is that the points would not encode for enough information, while having a too high dimensional space it's possible to incur in the "curse of dimensionality".

2.3.1 Curse of Dimensionality

In this work we reserved special care in selecting the dimension of the embedding space into which map the different video snippets. This because the novelty detection task, which is the essence of the second stage of our approach, is susceptible to the dimensionality of the space in which it's performed since some counter intuitive phenomena arise in high dimensions. In their work [26], Zimek et al. explore these problems which are commonly referred to as the "curse of dimensionality". The first problem mentioned in [26] is the "concentration of distances", that states, citing the main discovery made by Beyer et al. in [27], that given a set of i.i.d. points in a d-dimensional space X_d, the ratio of the variance of their lengths ($||X_d||$) and the length of the mean point vector $\mathbb{E}[||X_d||]$ tends to zero as the dimension d tends to infinity.

$$\lim_{d\to\infty} \left(\frac{||X_d||}{\mathbb{E}[||X_d||]} \right) = 0 \tag{12}$$

The consequence of this lemma is that for any set of points in a high dimensional space the relative contrast between the farthest-point (D_{max}) point and the closest point (D_{min}) vanishes, making the concept of nearest neighbours unstable [27].

$$\lim_{d\to\infty} \left(\frac{||X_d||}{\mathbb{E}[||X_d||]} \right) = 0 \implies \frac{D_{max} - D_{min}}{D_{min}} \to 0 \tag{13}$$

Zimek et al, also show that this result leads to high false positives rates using kNN outlier detection, while the Local Outlier Factor algorithm gives more stable results.

A maybe even more important factor that must be taken into account when dealing with high dimensional data is that irrelevant features can conceal other more relevant attributes, possibly masking outliers. As shown in fig 2.9 (taken from [26]) having a relative high number of noisy dimensions drastically reduces the anomaly score measured using LOF.

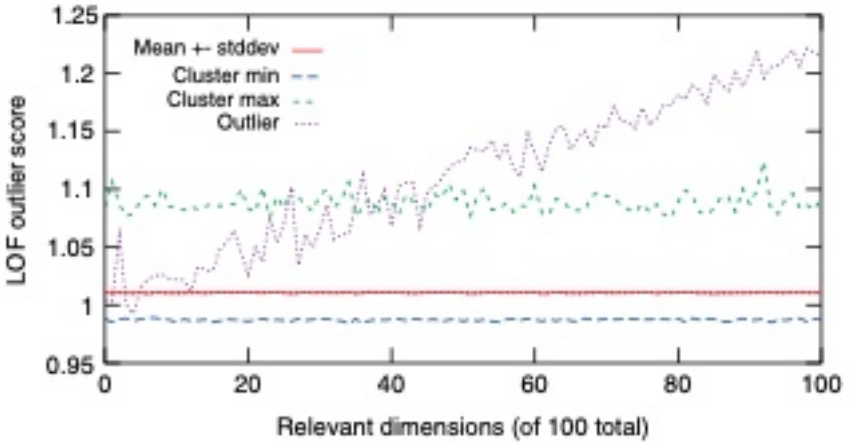


Figure 2.9: Increasing the number of noisy dimensions drastically reduces the LOF outlierness measure of the anomalous point (purple dotted line). Image taken from [26]

2.3.2 Local Outlier Factor

Local Outlier Factor (LOF) is a novelty detection algorithm introduced by M. Bruening in [25]. Based on the assumption that outliers lives in low densities region of the space, LOF associates an anomaly score to a sample by computing how much it is isolated with respect to its neighbours.

To explain better the algorithm let's introduce a couple definitions. First, the k-distance(O) is the distance from O to its k-nearest-neighbour. Then the reachability distance between two points P and O (reach-dist$_k(P, O)$) can be defined as:

$$\text{reach-dist}_k(P, O) = \max\{\text{k-distance(O)}, dist(P, O)\} \tag{14}$$

that is the distance between P and O if such distance is greater than the k-distance of O. It must be enphasized that this is not a"distance" in the proper mathematicals ense of the word since it is not symmetric in it's arguments. However, this definition is useful to have a less noisy measure of distance between points and allows us to introduce the local reachability density (lrd) as the inverse of the average reachability distance of a point P from its neighbours.

$$lrd(P) = \left(\frac{1}{N_{neigh}(P)} \sum_{O \in \text{k-neigh(P)}} \text{reach-dist}_k(P, O) \right)^{-1} \tag{15}$$

where $N_{neigh}(P)$ is the number of neighbors which are within a distance k-distance(P) from P

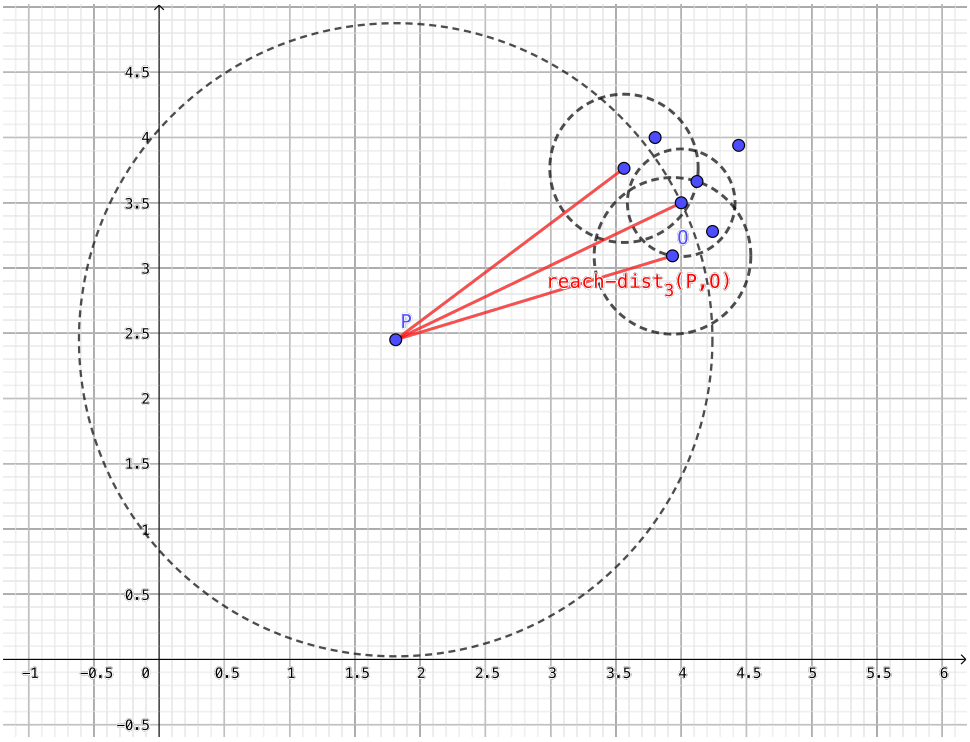


Figure 2.10: Illustration of Local Outlier Factor. P is an outlier since its local density it's lower then its neighbours

(it could be more than k if there were two points or more tied with the same distance).

This can be seen as a measure of densities of points around P: if P is an low density region of the space, meaning its nearest neighbours are far, the average reachability distance will be high, hence its lrd low (see fig 2.10).

The Local Outlier Factor (LOF_k) of a point P compares the local reachability density of that point with its neighbours

$$LOF_k(P) = \frac{\sum_{O \in \text{k-neigh(P)}} \frac{\text{lrd}_k(P)}{\text{lrd}_k(O)}}{N_{neigh(P)}} \tag{16}$$

$$\tag{17}$$

The value of $LOF_k(P)$ tells wether P can be considered an outlier or an inlier:

- If $LOF_k(P) \leq 1$ the point P sees around him the same density of point than its neighbours, and therefore it's called inlier.
- If $LOF_k(P) > 1$ than P is an outlier since the density of points around its neighbours are greater then the density around P

In this work we used the scikit-learn implementation of LOF_k (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html) setting the

number of neighbors hyperparameter $k = 20$.

3 Experiments

The structure of the experiments that we delineated to create and test the anomaly detector is the following:

1. **Comparison of the metric learning losses**: For the purpose of this project we needed a feature extractor that clusterized effectively. Unsure about which loss, Triplet (TL) or Contrastive (SCL), was better for this purpose, we experimented on an image dataset a Resnet 50 on Caltech 101 using both losses and compared their silhouette scores. The model trained with SCL ended up having the better performance and so we used it for the later experiments to train the feature extractor of the anomaly detection system.
2. **Building the feature extractor**: This section is focused on the feature extractor of our anomaly detection system and it's divided into two subsections, one for each of its parts:
   (a) First we do a brief review of the architecture we choose as the backbone of our model, the Resnet(2+1)D, and other architectures available on the internet.
   (b) Then we tuned several projection heads with different embedding dimensions, and compared their silhouette scores. We also trained a projection head using the Cross Entropy loss (CEL) to have a baseline performance to compare to those obtained tuning with SCL.
3. **Anomaly detection on UCF Crimes**: This part is the core of our project, consisting of the anomaly detection experiments. In this section we introduce the dataset used for the task and we performed several analysis using as a performance metric the AUC score. In particular we focused on two lines of experimentation:
   (a) Changing the dimension of the embedding space and checking its effect on the performance of the anomaly detector, the best one being 256.
   (b) Tuning with different losses keeping the dimension of the space fixed at 256 and seeing their influence on the performance of the anomaly detector. Seeing how by using different losses we weren't seeing an improvement on the performance,

we decided to do a model without tuning, meaning, with a randomly initialized projector. This ultimately was our best performing model with an AUC of 0.75, which hints to some issues with the tuning using UCF101, but more importantly it ultimately pointed to the fact that our model works and still holds promise to manifest.

3.1 Choosing the metric learning loss: Triplet vs. Contrastive

In this section we decided to put to the test the two losses mentioned in the theoretical framework in sections 2.2.1.2 and 2.2.1.3. To have our own empirical confirmation of which of the two losses, Triplet (TL) or Contrastive (SCL), worked better for metric learning, we trained a Resnet50 on the Caltech101 twice from scratch, first using TL and then using SCL. We calculated the silhouette score of the validation set throughout both trainings. Additionally we did the same with the Cross Entropy Loss, for comparison reasons. The results are shown in Figure 3.1 from which it can clearly be seen that the best silhoutte score was obtained with the Supervised Contrastive Loss. We therefore didn't use the Triplet Loss in later experiments.

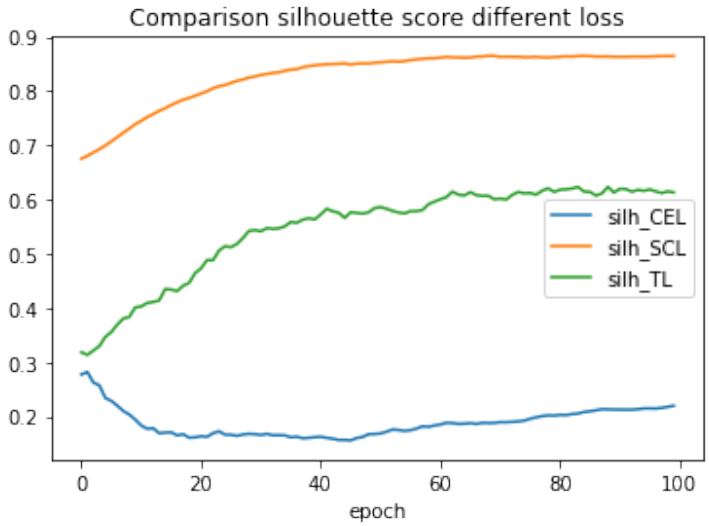


Figure 3.1: Silhouette scores of models trained on images with various losses, Cross Entropy (CEL), Supervised Contrastive Loss (SCL), Triplet Loss (TL)

3.2 Building the Feature Extractor

3.2.1 Choosing the Backbone

As the backbone of our feature extractor we chose the Resnet50(2+1)D-RGB. Next to I3D-RGB, which is one the standards in the industry, it reaches comparable results on the kinetics dataset [15], having a 0.1% worse accuracy. Nevertheless, we chose the Resnet in spite of the minute difference because it was easier to find a pretrained version of it (https://github.com/kenshohara/3D-ResNets-PyTorch).

Amongst the Resnets(2+1)D the one with depth 50 is the one that achieves better performance on kinetics, as reported by [28].

It's also worth mentioning that we don't use optical flow in our model. This mainly for two reasons:

1. The Contrastive Loss performs better on a larger batch size, and given that videos are memory expensive we were already heavily constrained on the size of the batch we could use for training, even without Optical Flow.
2. Calculating the optical flow is time consuming, so using it would've significantly extended training times that were already long.

3.2.2 Tuning the projection head on UCF101

Having a pretrained backbone we proceeded with the tuning of the projection head. For this purpose we trained it for 20 epochs using UCF101, keeping the backbone frozen. UCF101 [29] is an action recognition dataset with 101 classes composed of 13.320 videos. We didn't use Kinetics 700 even though it would seem natural given that the backbone was pretrained on it, because of hardware constraints.

3.2.3 Experimental Setup

During training we used as data augmentations 10°rotations, gaussian noise, flipping, random time crop and random skipping of frames.

To train with the SCL the loss is calculated directly on the output of the projection head. To train with the CEL an extra fully connected layer is added to go from the emebedding space to the class scores. A diagrammatic representation of both setups is shown in fig 3.2.

The silhouette score is computed on UCF101's validation set on the output of the projection head.

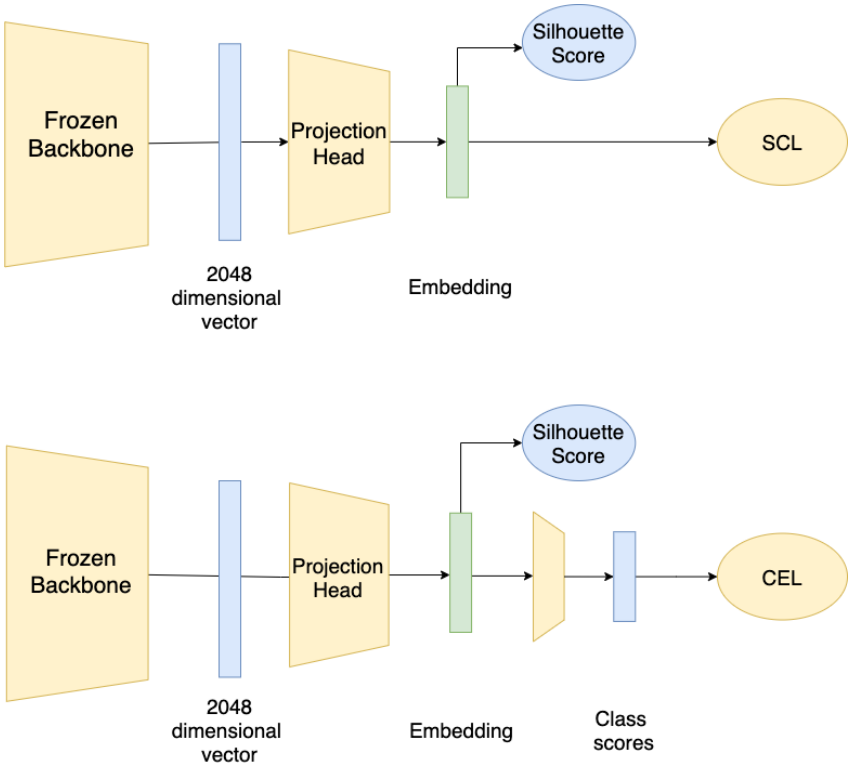


Figure 3.2: Experimental setup used for the different losses.

3.2.3.1 Tuning with different embedding dimensions

Knowing about the different effects that the embedding dimension has on clusterizability, we decided to train several projection heads with different dimensions and see which one reached the best silhouette score. The results are shown in Table 3.1. Interestingly enough, the embedding dimension didn't seem to have any effect on the final silhouette score on UCF101, it only seemed to influence the number of epochs it took to reach the silhouette score (see Figure 3.3). It must be said though, that the embedding dimension did ultimately influence the AUC score of the final anomaly detection system (more on this on the chapter of experiments on UCF crime).

Embedding Dimension	1	2	8	64	128	256	512
Silhouette Score	0.16	0.53	0.54	0.50	0.52	0.53	0.52

Table 3.1: Shilouette scores on UCF101 obtained with projection heads with different embedding dimension.

3.2.3.2 Tuning a projection head also with CEL

We also trained an extra projector with an embedding space dimension of 128 using the CEL. This to have a performance baseline for the projection heads trained with SCL and to confirm that indeed the SCL produced better silhouette scores also training on videos. The final silhouettes scores obtained are shown in Table 3.2 and in fig 3.4

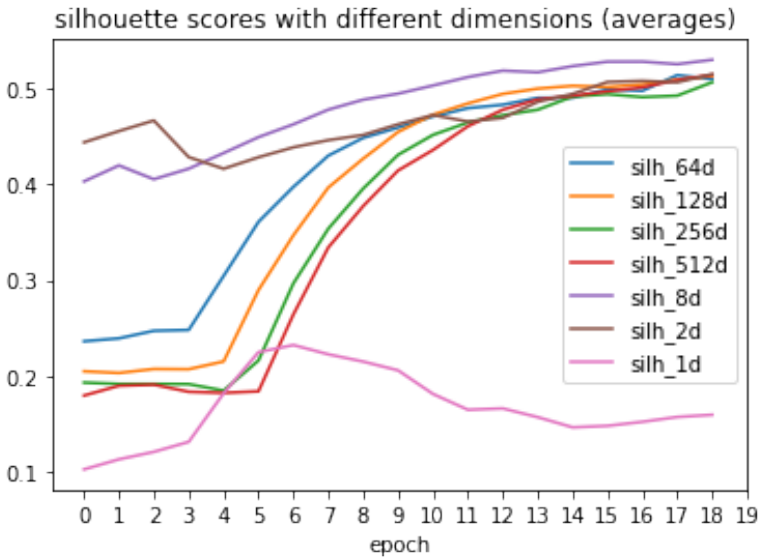


Figure 3.3: Silhouette score during training of the different projectors.

Loss	Cross Entropy Loss	Supervised Contrastive Loss
Silhouette Score	0.32	0.53

Table 3.2: Silhouette scores on UFC101 using Cross Entropy Loss vs Supervised Contrastive Loss.

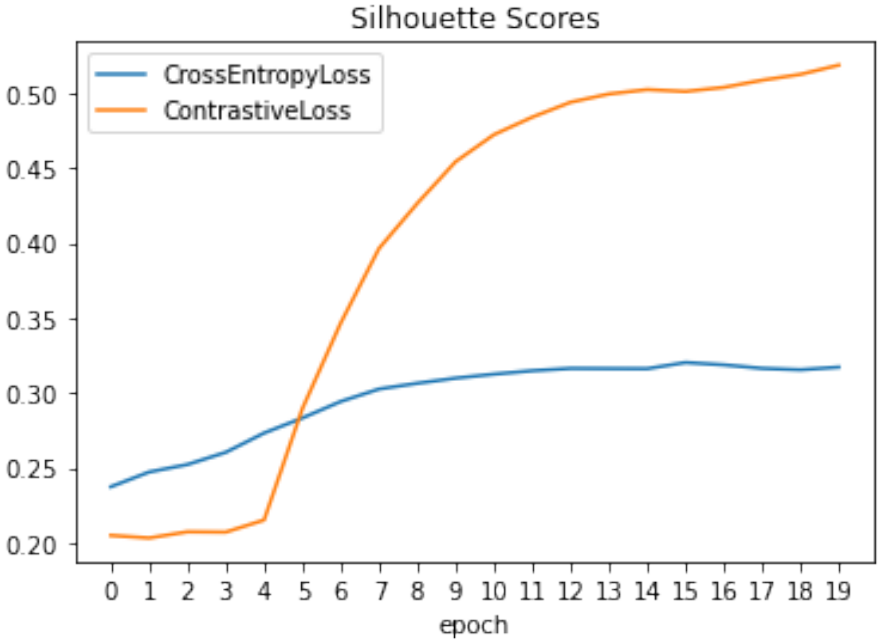


Figure 3.4: Silhouette scores during training on UFC101 using Cross Entropy Loss vs Supervised Contrastive Loss.

3.3 Anomaly detection on UCF Crime

Finally in this section we perform an actual anomaly detection task. Up until this point the anomaly detection system consists of these two parts:

1. **A Feature Extractor**: The Resnet(2+1)D pretrained on the kinetics 700 dataset, followed by the projection head (the last fully connected layer) which was tuned using UCF101.
2. **A Novelty Detection algorithm**: Local Outlier Factor [25] to do novelty detection in the embeddings obtained with the feature extractor.

The first will be used to convert video snippets into lower-dimensional embeddings.

Having the anomaly detection system, we needed a dataset on which to test it; we decided to perform our tests on the UCF Crime dataset, which will be presented in the following section.

3.3.1 UCF Crime dataset

Figure 3.5: Sample of anomalous frames from UCF Crime. a- Abuse, b- Arson, c- Explosion, d- Fight, e- Road Accident, f- Shooting. Image taken from [30]

UCF Crime is a famous dataset for anomaly detection. It contains several untrimmed videos from surveillance recordings that are divided into 13 categories: Normal Videos (that contain footages of normal situations in various settings), Abuse, Arrest, Arson, Assault, Burglary, Explosion, Road Accidents, Robbery, Shooting, Shop lifting, Stealing, Vandalism [1].

3.3.1.1 How we used the dataset

The hypothesis we wished to test is the following: Having our system seen footage of normal events in a given setting, can it detect anomalies in that same setting?

This question lead us to use the UCF Crime dataset in a way that was not the originally intended by the creators of the dataset:

We did not make the obvious choice of using the Normal Videos of the dataset for training the novelty detection system and then the anomalous ones for testing, because that would've meant to train the system in one setting to then test it in another one. Instead we trained it on normal parts of the annotated anomalous videos and used the remaining normal and anomalous parts for inference.

In the designated dataset, only its test set contains the annotations that indicate the start and the end of the anomaly within the video so, in order to test our unsupervised approach in the way just explained, we trained and tested on just this annotated subset, the test set.

3.3.2 Experimental Setup

To test our system for anomaly detection on the UCF Crime dataset we followed these steps:

1. Creation of the embeddings: We divided the anomalous videos into 16 frame long snippets and we labeled them as anomalous if it had more than 3 anomalous frames (20% of the snippet). If it had less, we labeled it as normal.
2. Training of the Novelty Detection Algorithm: Out of all the created embeddings mentioned in the previous step, we trained our novelty detector (Local Outlier Factor) on 80 % of the ones labelled as normal.
3. Testing: The remaining embeddings that were not used for training we used for inference; we used our trained novelty detector to predict whether they were anomalous or not. The ROC curve and the respective AUC score were then calculated to use as performance metrics.

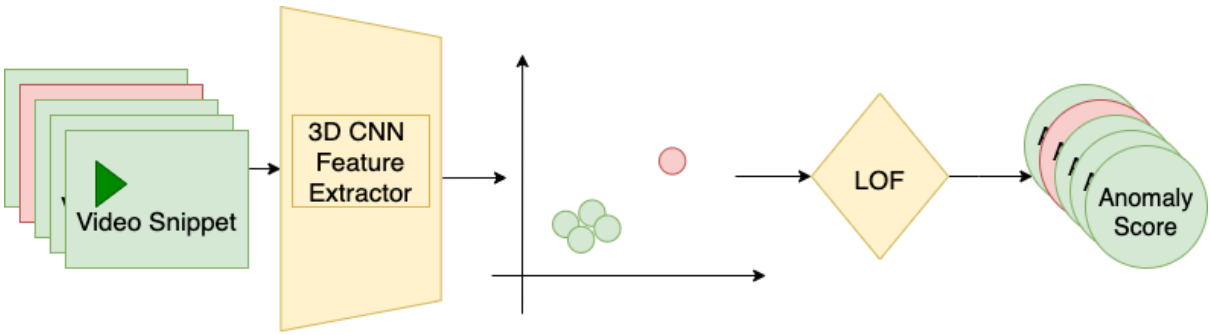


Figure 3.6: Schematic of the anomaly detector. Video from UCF Crime are passed to the 3D CNN which creates the embeddings. The anomaly detection on the embeddings is performed using the Local Outlier Factor algorithm.

3.3.3 Testing the projectors with different embedding sizes

As reported in section 2.3.1 we suspected that the dimension of the space where we projected our videos embedding plays a role in the performance of an anomaly detection system. Yet, in the experiments on UCF101 the dimension of the projection head seemed to have little to no influence in the clusterization of the embeddings (Table 3.1), hence we performed the anomaly detection task using the already tuned heads with different dimensions and compared their respective AUC scores to see if dimension also had no influence in the performance of this task.

Figure 3.7 and Table 3.3 show the result of these experiments and here, dimension indeed influenced the result, having found that for this particular task 256 dimensional embeddings seemed to perform best with respect to lower and higher dimensions, resulting in an AUC score of 0.70.

It's worth mentioning that the 2048 dimensional result is obtained by using only the backbone of the feature extractor without any projection head, because it already had a final embedding dimension of 2.048. This result shows that reducing the dimensionality indeed contributes positively to the performance of our systemr.

Emb Dimension	AUC score
1	0.500
2	0.520
8	0.637
64	0.676
128	0.681
256	0.696
512	0.673
2048 (*)	0.577

Table 3.3: AUC scores using projection heads with different embedding dimension. (*) For dimension 2048 we used just the backbone with no projection head at all.

3.3.4 Testing different losses and the final result

Having found the best performing embedding dimension, in the experiments that follow we kept the dimension fixed in at 256. Here our interests shifted to understanding the impact of the different tuning loss of the projection head had on the final anomaly detection task, wondering if the improvement found in the silhouette score during the tuning translated into an improvement also on the anomaly detection. We decided then to compare the AUC scores of a head tuned with CEL to the one tuned using the SupCon.

We found that the feature extractor tuned using the supervised contrastive loss (SCL) didn't lead to an improvement with respect to the one tuned using the cross entropy loss (CEL)

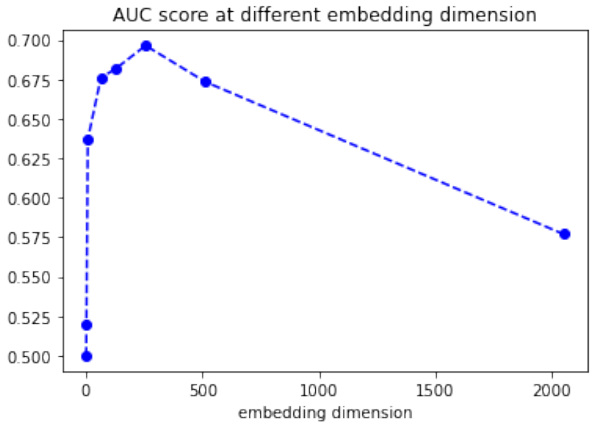


Figure 3.7: AUC score for anomaly detection on ucf crimes with different embedding dimensions

(see table 3.4), even if the former one reached a better silhouette score on UCF101 (as reported in tab 3.2).

This result suggested to us that the improvement of the clusterization brought by the contrastive loss on UCF101 didn't translate into a better clusterization in UCF Crime. We hypothesized that one of the reasons for this could be that the type of videos in the two datasets are too different, i.e. there's a domain gap between the videos used for training the feature extractor and those used for the anomaly detection task. Indeed, human actions datasets are mainly composed of close-up shots of the actions, while UCF Crimes contains mostly distant and wide-angle shots from surveillance cameras.

To test this intuition we let the feature extractor "see" this kind of settings by performing an self-supervised training on the normal training videos of UCF Crime that we hadn't used up until this point.

In order to try to bridge the domain gap between the human action recognition datasets (Kinetics 700 and UCF101) and the surveillance videos of UCF Crime, we performed a training of the feature extractor on UCF Crime's normal videos using the self-supervised contrastive loss [20].

In each forward step 4 different views of the normal video are created. These views are sets of 16 frames taken at different time positions augmented using gaussian blur, random rotation of 10 deg and random horizontal flipping.

The contrastive loss is minimized when the embeddings of different views of the same video are closed together, for each video in the batch (fig 3.8).

The hope in doing this self-supervised training, is that the features used to identify same/different footages could be useful later to improve performance on the anomaly detection task. We calculated then the AUC on UCF Crime using this model and the result is

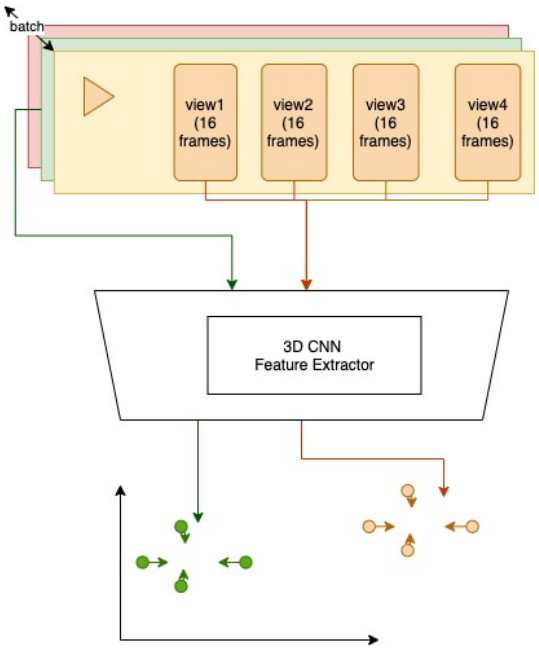


Figure 3.8: Diagram of the proposed self-supervised training for the feature extractor: Each normal video of UCF Crimes is cut into 4 different augmented views of 16 frames each and passed to the 3D CNN. The loss objective is to minimize the distance (expressed as cosine similarity) of same video snippets.

shown in table 3.4 also with the AUC scores previously obtained with the other losses.

Feature extractor training loss	AUC score
Supervised Contrastive	0.693 ± 0.003
Cross Entropy	0.708 ± 0.003
Self-Supervised Contrastive	0.697 ± 0.003

Table 3.4: AUC scores of different training strategies of the feature extractor. For the Supervised Contrastive Loss and the Cross Entropy Loss the feature extractor was trained on UCF101, while with the Self-Supervised Contrastive Loss it was trained on UCF Crime.

We noticed that this model didn't seem to offer any performance boost with respect to those tuned with the other two losses. Nevertheless this self-supervised training gave us a hint of the importance of the pretraining on kinetics:

Given the domain gap it was not guaranteed that our CNN architecture would actually extract motion features or it would just extract background features, but we show in 3.5, that the same model trained on the normal video of UCF Crime had a considerably increase of performance in the AUC score if the backbone was pretrained on kinetics 700 (tab 3.5.

Seeing that the neither the SupCon loss nor the Self-Supervised Contrastive Loss were improving the performance we decided to test the anomaly detector with a randomly initialised projection head without any tuning. This, to have a performance baseline to asses

Feature extractor trained self supervised	AUC score
with pretrainig	0.696 ± 0.004
without pretraining	0.576 ± 0.002

Table 3.5: Comparisons of AUC scores with backbone of feature extractor pretrained and not pretrained, 256 embedding size.

what was the positive contribution of the tuning step on the performance of the anomaly detector.

A model with a random dimensionality reduction wouldn't provide an unreasonable baseline since we know from results like the Johnson-Lindenstrauss lemma [31] that random projections approximately preserve the distance between points in an ensemble, distance between points being the main property necessary for our anomaly detection system to work. So if the information related to actions extracted by the backbone would partly preserved using this random dimensionality reduction.

This experiment lead us to a somewhat surprising result: the model without any tuning outperformed the rest having an **AUC score of** 0.748 ± 0.003.

This result is almost the same as the one reported in the original UCF Crime paper [1], in which, with their proposed method, they reached an AUC score of 0.7544 which strongly suggests our method has potential.

This also means the backbone is indeed managing to extract information useful for the anomaly detection task.

Nevertheless, the lower AUC scores of the tuned projection heads implies that, when projecting the embeddings into a lower dimensional space, the tuned heads are losing more information than the randomly initialised one, so, unlike the backbone they're not extracting information, they're losing it. Given that the main difference between the training of the backbone and the projection heads was the datasets, it's likely that the fact that the tuned heads didn't perform was the dataset used to tune them, UCF101. To fully test this hypothesis though, one should train also the projection head using kinetics 700 and check whether that would give better results. We didn't do such experiments in this work because of time and hardware constraints.

It's also worth emphasizing that while the pretrained backbone seems to be the reason for the good result, using the backbone alone to create the embeddings didn't provide nearly as good a performance , with an AUC score of 0.577 ± 0.003 as reported in table 3.3. So projecting into a lower dimensional space is indeed a key factor in guaranteeing a good performance for our anomaly detection system.

4 Conclusion

At the beginning of this work we set ourselves to answer whether 3D CNNs pretrained on action recognition could work as feature extractors to be used in video anomaly detection, to which we think the answer is on the affirmative. Additionally, we focused our research on improving the performance of the feature extractor in mainly two fronts: changing the loss with which it's trained and changing the dimension of its embedding space.

From our experiments we extracted the following conclusion:

1. Pretraining on action recognition works: Given the good final performance of our system we can conclude that models pretrained on human actions do extract information useful for anomaly detection on surveillance videos. It's worth noting that they're able to do so in spite of there being a domain gap between the videos used for pretraining and those of the anomaly detection task.

2. Dimension matters: Out of the diverse experiments that we conducted, we can definitely conclude that the dimension of the embedding space in which the videos are mapped influences the performance of the anomaly detection task. So, in order to use our system, particular care must be taken in choosing the embedding dimension to get the most out of its performance.

   This conclusion can be most clearly seen by the fact that just adding to the backbone a random dimensionality reduction (Section 3.3.4), the performance jumped from an AUC of 0.57 to an AUC of 0.75, that is, you get a performance increase just by changing the embedding dimension from 2048 to 256.

3. Care must be taken in choosing what dataset is used for the tuning: The fact that our model with a random projection head outperformed the tuned ones implies that the tuned heads are destroying more information in the compression than the random one. As explained in Section 3.3.4, the evidences points to the dataset chosen for the tuning as the most likely cause of this result; possibly a larger, harder dataset is necessary for this stage, but more research should be conducted to arrive to more definite conclusions.

4. The SupCon Loss clusterizes better than both the Triplet and the Cross Entropy but our results on how this translates to the anomaly detection task are inconclusive: In our experiments on Caltech101 and UCF101 we showed that the models trained with the Supervised Contrastive Loss reach higher silhouette scores than those trained with either the Triplet or the Cross Entropy Loss. Additionally, during the tuning stage of the projection heads using UCF101 we consistently got better silhouette scores using the SCL over those obtained with the CEL. Nevertheless, the question remains open about whether this better clusterization during tuning can be translated into better anomaly detection. Our anomaly detection results seemed to suggest that the SCL doesn't bring an improvement over CEL, however, concluding this would be erroneous, since the problems found during the tuning stage compromise those AUC results.

5. The method shows promise: Our anomaly detector reaches a similar AUC to those of other methods found in the literature , and this using only a random projection head . This means that, with an effectively tuned projection head, our model would very likely reach better AUC scores. The obtained AUC score serves to showcase the possibilities of our approach.

4.1 Future Works

There's potential left unexploited in our system given that our best performing model used the randomly initialized tuning head. Therefore is left as a future work to tune also the projection head on kinetics 700 to see what AUC scores could be reached.

Since our feature extractor doesn't require a computationally demanding training on the specific setting to be used, it lends itself to a readily implementation in a variety of settings, making a natural continuation of this work to apply this technique in real world scenarios. This would entail specifying a security camera on which to test it like the camera of a mall or office, gathering footage that would be considered normal for the setting, and finally using the approach to study what kind of events are found to be anomalous.

Bibliography

[1] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos, 2019.

[2] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2020.

[3] Hao Luo, Youzhi Gu, Xingyu Liao, Shenqi Lai, and Wei Jiang. Bag of tricks and A strong baseline for deep person re-identification. *CoRR*, abs/1903.07071, 2019. URL `http://arxiv.org/abs/1903.07071`.

[4] Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning temporal regularity in video sequences, 2016.

[5] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. volume 2454, pages 113–123, 09 2002. ISBN 978-3-540-44123-6. doi: $10.1007/3\text{-}540\text{-}46145\text{-}0\_17$.

[6] Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training, 2018.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: $10.1038/323533a0$. URL `https://doi.org/10.1038/323533a0`.

[8] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. doi: $10.1007/\mathrm{BF}02551274$. URL `https://doi.org/10.1007/BF02551274`.

[9] Victor Dibia. Convnet playground, 2016. URL `https://convnetplayground.fastforwardlabs.com/#/models`.

[10] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1): 221–231, 2013. doi: $10.1109/\mathrm{TPAMI}.2012.59$.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 25:1097–1105, 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018.

[14] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition, 2017.

[15] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition, 2018.

[16] Dave Gershgorn. The data that transformed ai research—and possibly the world. URL https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/.

[17] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.

[18] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. A short note on the kinetics-700-2020 human action dataset, 2020.

[19] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

[20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[21] Arthur Zimek and Peter Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8:e1280, 08 2018. doi: 10.1002/widm.1280.

[22] Edwin Knorr and Raymond Ng. Algorithms for mining distance-based outliers in large datasets. *VLDB*, 06 1998.

[23] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th*

International Conference on Neural Information Processing Systems, NIPS'99, page 582–588, Cambridge, MA, USA, 1999. MIT Press.

[24] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 01 2009. doi: 10.1109/ICDM.2008.17.

[25] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335388. URL https://doi.org/10.1145/335191.335388.

[26] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012. doi: https://doi.org/10.1002/sam.11161. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11161.

[27] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *ICDT 1999. LNCS*, 1540, 12 1997.

[28] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[29] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[30] Ramna Maqsood, Usama Bajwa, Gulshan Saleem, Rana Raza, and Muhammad Anwar. Anomaly recognition from surveillance videos using 3d convolution neural network. *Multimedia Tools and Applications*, 02 2021. doi: 10.1007/s11042-021-10570-3.

[31] William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 01 1984. doi: 10.1090/conm/026/737400.