UNIVERSIDAD POLITÉCNICA
DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

EIT DIGITAL MASTER IN DATA SCIENCE

Use of Autoencoders as a Generalization Method for Anti-Spoofing Classification

Master Thesis

Tomás Ernesto Watman Outon

Madrid, June 2019

This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of EIT Digital Master in Data Science.

Master Thesis
EIT Digital Master in Data Science
Thesis Title: Use of Autoencoders as a Generalization Method for Anti-Spoofing Classification
June 2019

Author: Tomás Ernesto Watman Outon
Mathematical Engineering Graduate
Universidad Complutense de Madrid

Supervisor:

Francisco Javier Segovia Pérez
Professor

Lenguajes y Sistemas Informáticos e
Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain

Title of the thesis: Use of Autoencoders as a Generalization Method for Anti-Spoofing Classification

Author: Tomás Ernesto Watman Outon

Advisor: Francisco Javier Segovia Pérez

Abstract

Computer vision is one of the hot topics nowadays when talking about artificial intelligence, covering a wide range of problems under its domain, from document verification to object detection.

One of these problems consists of detecting spoofing attacks, masquerading as a registered user in a face-recognition system, gaining illegitimate access and advantages, used in face-recognition payments, phone unlocking, workers checking –in…

When training models focused on solving this kind of problems, one of the main concerns is the data we have to deal with due to its small variety, leading to generalization problems.

Some of the main approaches that try to tackle these overfitting issues include dropout techniques, model variations or adding some kind of regularization.

While in general the way of adding regularization is by introducing a regularization parameter in the objective function, in this thesis we will be presenting another equivalent regularization algorithm.

To my parents, for always supporting me during these not-so-long 24 years.

Acknowledgements

This project couldn’t have been done without the members of the AI department of Everis Madrid, who helped me with the decision on how to focus the problem to approach and giving me advice during the process, and Javier Segovia from UPM, for being so patient with me and being always willing to help.

Also, special thanks to X.Tan, Y.Li, J.Liu and L.Jiang, creators of the NUAA Photograph Imposter Database.

Table of Contents

List of Figures

Figure 5.

Image showing a comparison between Autoencoders and PCA

O. Cohen. "PCA vs Autoencoders", 2018. [Online] Available: https://towardsdatascience.com/pca-vs-autoencoders-1ba08362f450 [Accessed: March 2019]

Page 21

Figure 6.

Image showing the architecture of a Denoising Autoencoder

J. Jordan. "Introduction to Autoencoders", 2018. [Online] Available: https://www.jeremyjordan.me/autoencoders/ [Accessed: April 2019]

Page 21

Figure 7.

Image showing the architecture of a Sparse Autoencoder

J. Jordan. "Introduction to Autoencoders", 2018. [Online] Available: https://www.jeremyjordan.me/autoencoders/ [Accessed: April 2019]

Page 22

Figure 8.

Images showing the trajectory evolution of 50 networks with and without pre-training, being the darker the color of the dot, the later the epoch. To plot the trajectories, the authors used tSNE (left) and ISOMAP (right)

Erhan et al. "Why does Unsupervised Pre-Training Help Deep Learning?", 2010. Université de Montréal.

Page 25

Figure 13.

On the left image there are displayed several different Haar features. On the right image, how this features are measured over a picture.

P. Viola, M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features”, 2004. Mitsubishi Electric Research Lab.

Page 29

Figure 14.

Image showing how LBP converts each pixel into a feature vector

L. Sánchez. “Local Binary Patterns Applied to Face Detection and Recognition”, 2010. Universitat Politècnica de Catalunya.

Page 30

Figure 15.

Samples from the NUAA Photo Imposter Database, taken on different sessions (each row represents one session). The left pairs are samples from real people and the right pairs of their pictures

Tan et al. “Face Liveness Detection from a Single Image with Sparse Low Rank Bilinear Discriminative Model”, 2010. Nanjang University of Aeronautics and Astronomics.

Page 31

Figure 16.

Images showing the process of building the NUAA Photo Imposter Database; more specifically, how they tried to make the pictures look more real by moving them around the camera vision and bending the pictures on both the horizontal and the vertical axis

Tan et al. “Face Liveness Detection from a Single Image with Sparse Low Rank Bilinear Discriminative Model”, 2010. Nanjang University of Aeronautics and Astronomics.

Page 32

List of Tables

1. Introduction

Computer vision is a field of study focused on teaching a computer to see or understand images.

It is a subfield of artificial intelligence and machine learning since it involves the use of some general methods developed in those areas to improve the results of more specialized learning algorithms.

To sum up, computer vision attempts to reproduce the capability of human vision.

The goal of computer vision is to extract useful information from images. This has proved a surprisingly challenging task; it has occupied thousands of intelligent and creative minds over the last four decades, and despite this we are still far from being able to build a general-purpose "seeing machine." (Page 16, Computer Vision: Models, Learning, and Inference, 2012.)

This problem may seem like an easy task since it is something humans can do without any efforts. However, it is not as trivial as one could think of.

First of all, even if it is something we are all continuously doing, it is, as anything related to the brain, not yet fully understood.

And also, and more related to other machine learning problems, the data we have to deal with is highly complex: An object could be seen from vary different angles and distances, with different light conditions, shapes, colors… Maybe not even the full object may be seen or it may be deformed, but could still be recognizable.

A powerful visual system must be able to extract information from any of the infinite possible configurations of an image.

But, despite all this difficulties, computer vision is still very used in many different problems such as:

- **Image classification.** To what broad category does an image belong?
- **Object detection.** Where is an object located inside an image?
- **Image segmentation.** To what object does each pixel of the image belong?
- **Computational photography.** Modification of the properties of an image (style transfer, increased resolution…)

For a computer to be able to understand an image, we should see it as an array of pixels, each one of them having a value from 0 to 255. With this information, various algorithms called feature extractors will try to detect features of the image such as edges or contours.

Here is where deep learning stands out, simplifying the process of feature extraction using convolutions.

A convolution is a mathematical operation on two functions to produce a third one that expresses how the shape of one of them is modified by the other.

Deep learning takes advantage of convolutions in what is called convolutional layers.

A convolutional layer is actually a combination of three steps: The initial convolutional process, a non-linear activation function and a pooling step (down sampling).

- In the convolutional part, we have a matrix (named kernel) with some weights assigned sliding along the image. The piece of the image covered by the kernel on each step is the receptive field.
  While the kernel is moving, the convolutional layer multiplies the values of the receptive field with the weights of the kernel and sums them up.
  Every unique location of the kernel over the input image produces a number, and the collection of numbers at the end of the sliding process is called a feature map.
  By obtaining different feature maps, what we are really doing is extracting different features from the image.

- By just taking as an output the result from the previous part, what we are doing is trying to model our problem using a linear function.
  As mentioned before, the problem we are dealing with is not trivial; therefor a linear function may not be the most accurate.
  Also, when working with neural networks, linearity presents other
  Important problems:
    - The derivative of a linear function is constant, not allowing us to fully exploit gradient descent.
    - The combination of linear functions is another linear function, making any deep network equivalent to a single layer.

  For this reasons, it is highly recommended the use of a non-linear activation function applied to the results from the convolutional part.

- A problem with the feature maps is that we want them not to be sensitive to the location of said features.
  To make the model more robust to changes, we shall down sample the output on each layer before passing to the next one.
  Pooling layers do this down sampling by summarizing the presence of features in patches of the feature map.
  There are two prevailing pooling operations:
    - Average pooling. It calculate the average value for each patch on the feature map.
    - Maximum pooling. It calculates the average value for each patch on the feature map.

  In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. (Page 342, Deep Learning, 2016.)

This structure is supported with a biological interpretation. Hubel & Wiesel [14] designed a hierarchical model of the brain composed of simple cells that activate when they detect basic shapes and complex cells that combine the activations of the simple cells by activating to the same shapes but with less sensibility to position.
Simple cells would be represented by the convolutional process while the pooling layer acts as the complex cells.

By adding convolutional layers, we build a convolutional neural network that, together with fully connected layers, can deal with any of the problems mentioned above.

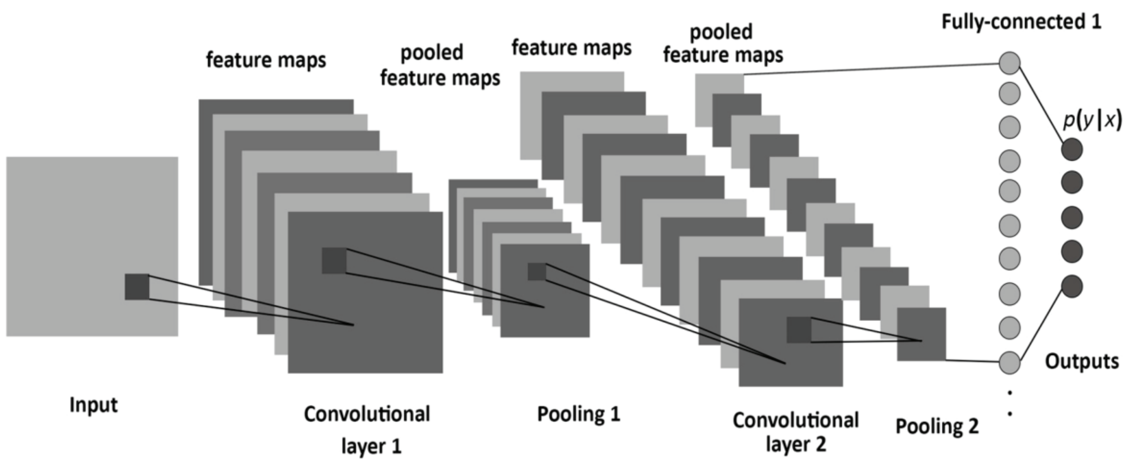


Figure 1

1.1. Image Classification

Image classification involves predicting the class of one object in an image.

This is the most basic problem in the computer vision field but, at the same time, the most studied one since it is where researchers focused to develop new network architectures.

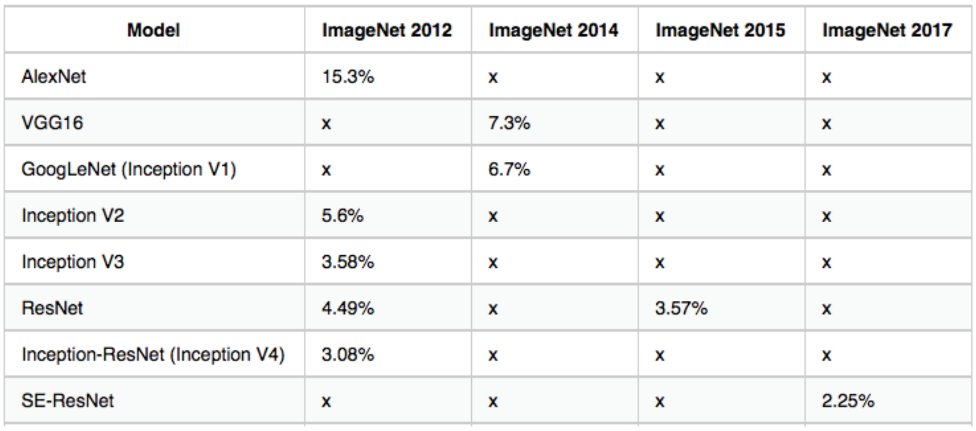
The ImageNet Challenge

The ImageNet dataset has become, together with other important image collections such as MNIST, one of the pillars of computer vision.

It is formed by millions of images, each of them belonging to one of the one thousand possible classes. The challenge consists in predicting the most probable set of five classes that contains the real tag of the input image.

Up until 2012, traditional algorithms such as SIFT were used to tackle the competition. It was then when A. Krizhevsky et al. [22] introduced a deep learning solution, revolutionizing the field and starting a race to build the deepest classifier.

The introduction of nonlinearities (RELU activation function) together with smaller filters to reduce the number of parameters (VGG) [20], the development of *inception modules* (training multiple convolutional layers simultaneously and stack their feature maps linked with a multi-layer perceptron), producing non-linear transformations (GooGleNet) [37], and residual learning, helping with the vanishing gradient problem (ResNet) [19] are some of the improvements made thanks to the research inside this topic.



Model	ImageNet 2012	ImageNet 2014	ImageNet 2015	ImageNet 2017
AlexNet	15.3%	x	x	x
VGG16	x	7.3%	x	x
GoogLeNet (Inception V1)	x	6.7%	x	x
Inception V2	5.6%	x	x	x
Inception V3	3.58%	x	x	x
ResNet	4.49%	x	3.57%	x
Inception-ResNet (Inception V4)	3.08%	x	x	x
SE-ResNet	x	x	x	2.25%

Table 1

1.2. Object Detection

There are some problems where knowing what is inside an image is not enough; we should also extract the exact position of the object inside the picture.

This is the case of tracking objects, counting problems, self-driven cars or face detection.

Multiple algorithms have been created in order to solve this issue, each of them with their own strengths and weaknesses:

- R-CNN (Region-CNN) follows the most basic idea, and is one of the state-of-the-art CNN-based object detection approaches.

  This idea consists on dividing the total image into smaller regions and applying the classification in each of them.
  This regions are created with the use of a fixed-sized sliding window that checks every positions of the image.

  This process not only takes a huge amount of time (depending on the size of the window, the number of needed predictions per image increases drastically) but is also not very accurate due to the size of the window being fixed (variable ratios between the sizes of the object with respect to the distance to the camera and the window may lead to errors).

  Faster R-CNNs are improvements of this method, but instead of providing the possible regions by using a sliding window they train a separate CNN to try to predict where the good regions are.

- YOLO (You Only Look Once), unlike the previous algorithms that don't look at the whole image but parts of it, is an algorithm that predicts the bounding boxes and their class probabilities with a single CNN.

  YOLO works as follows:

  - It splits the input image into an SxS grid
  - On each cell grid, two things are predicted: n bounding boxes and some conditional probabilities.
    - For each bounding box, both the position of it and its probability of containing an object are predicted.
    - For each cell, the probabilities of belonging to different classes given that the box contains an object are also predicted

- By combining the previous predictions and keeping only the most important bounding boxes (using thresholds and NMS), we can detect and predict the different objects present in the input image.

The main advantage of using YOLO is the decrease on the predicting time (around 45 frames per second); however, it may fail with small objects appearing in groups or generalizing for objects with different aspects ratios.

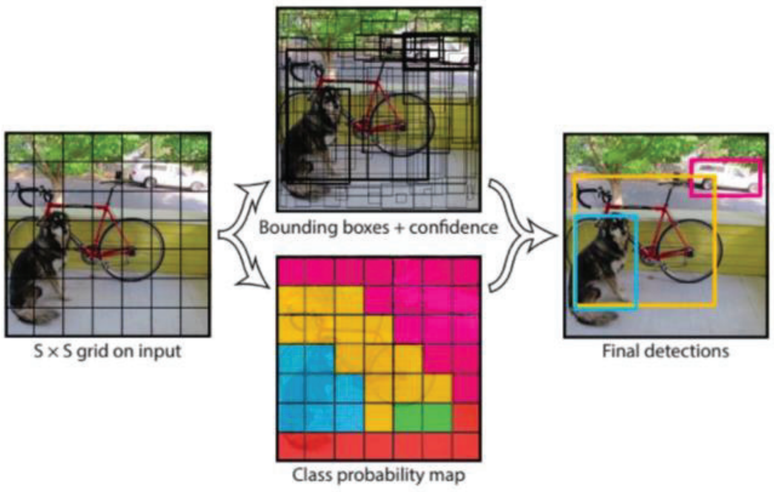


Figure 2

- SSD, as YOLO, predicts the position of the object and its class by just passing the input image through one network.

  In the case of SSD, it makes use of standard architectures for image classification in the first layers followed by an auxiliary structure to produce detections.
  This auxiliary structure is made of progressively decreasing in size convolutional layers, unlike YOLO which uses a single scale feature map.

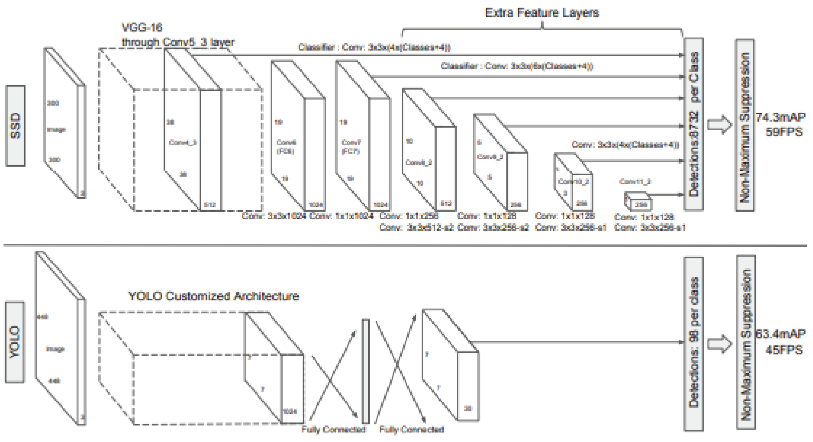


Figure 3

1.3. Autoencoders

Using convolutions is not the only tool a neural network can use to learn features about data.

In this chapter we will make an introduction about autoencoders, which are non-supervised algorithms trying to replicate the input data into the output.

What autoencoders try to learn is a representation of the identity function such that the output $\hat{x}$ is similar to the input x. This networks could be seen as composed by two parts: an encoder, f(x), and a decoder, g(f(x)). It is the first part the one we are interested in.

An autoencoder can be trained as any other feedforward neural network, the only difference is the final objective (different loss function). However, to avoid overfitting and end up learning the identity function, a series of restrictions can be set up, dividing autoencoders into two main categories: undercomplete and regularized.

1.3.1. Undercomplete Autoencoders

One of the restrictions that we can ask the autoencoder to fulfil is setting the dimension of the final layer of the encoder lower than the one from the input.

In this case, the loss function L(x, g(f(x))) tries to minimize the difference between the output and the input.

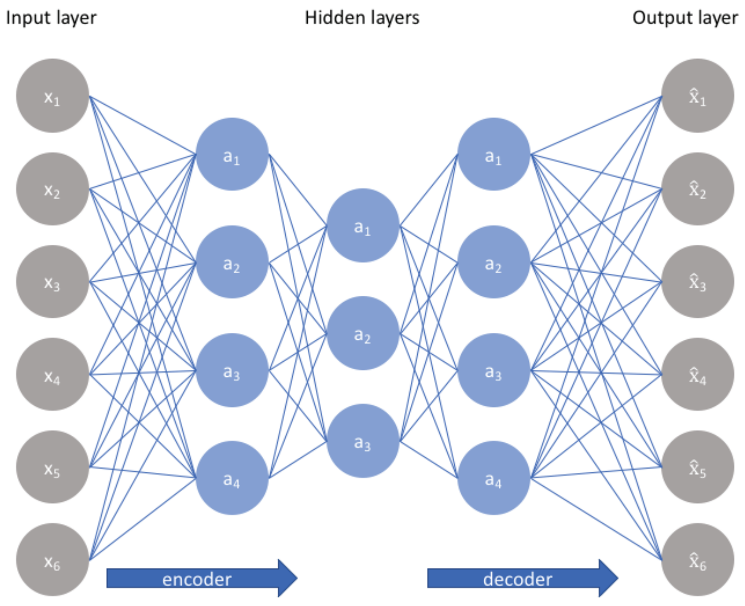


Figure 4

This type of autoencoders learn the most salient features of the distribution of the data.

When we work with a linear encoder and the loss function is calculated using MSE, the autoencoder learns the same subspace as PCA. We can get a more powerful result by using non-linear encoders and decoders.

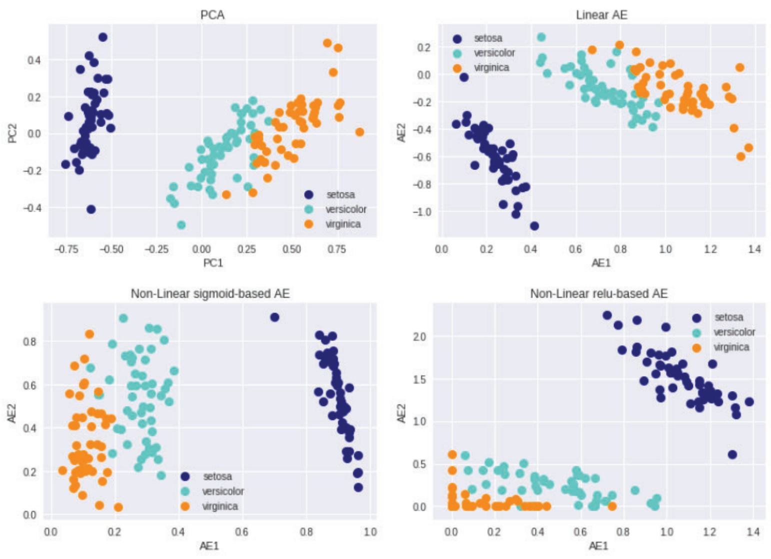


Figure 5

1.3.2. Regularized Autoencoders

Instead of restricting the structure of the network, we can set the restrictions to the loss function we use to train.

One of the most common and used cases of this type of autoencoders are denoising autoencoders.

In this case, what we try to minimize is the loss function L(x, g(f(x + ε))), where ε is some kind of noise we add to the input to try to force the model to eliminate it.

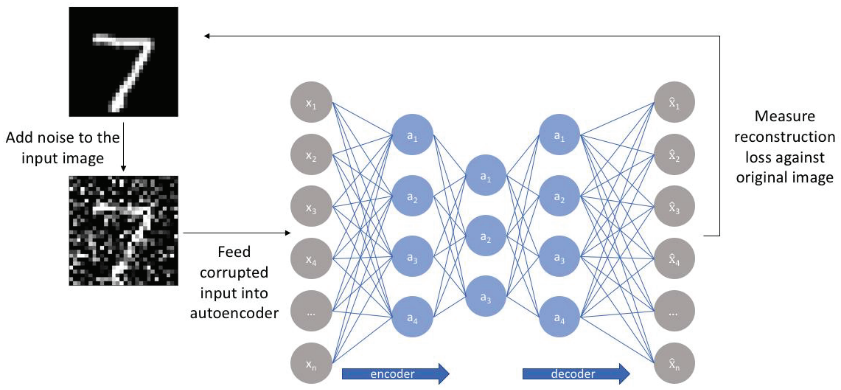


Figure 6

There are cases where we want similar inputs to have similar encodings. Contractive autoencoders take care of this by trying to diminish the derivatives in the hidden layers. This kind of autoencoders has proven better performances than DAE. (Rifai et al. [34])

	MNIST	CIFAR
Contractive Autoencoder	1.14	47.86
DAE (Gaussian noise)	1.18	54.81
DAE (Binary masking noise)	1.57	49.03
Autoencoder with weight decay	1.68	55.03
Autoencoder	1.78	55.47

Table 2

Sparse autoencoders are another type of regularized autoencoders that work similar to undercomplete autoencoders, but instead of making the architecture smaller they add a sparsity term, forcing some of the nodes to be inactive (with an output close to 0).

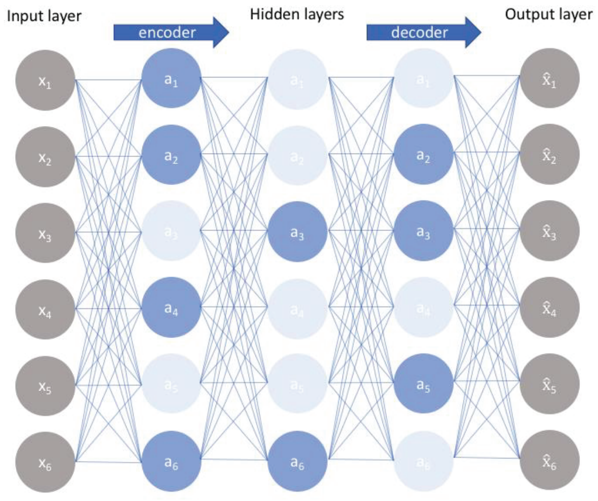


Figure 7

To add this sparsity term into our loss function, we should define some auxiliary functions. We denote:

$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} a_j^{(2)}(x^i)$, as the average activation of hidden unit j given an input x, being $a_j^{(2)}$ the activation of the hidden unit j for each input.

$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$, as the Kullback-Leibler divergence (a statistical function for measuring the difference between two distributions), being s_2 the number of neurons in the hidden layer and ρ the sparsity parameter (usually, a small value close to 0).

1.4. **Anti-Spoofing**

We define spoofing as the act of trying to impersonate another user in order to gain access to some information, spread malware or any other malicious cyber-attack.

While most of the spoofing attacks nowadays are more focused on system protocols and addresses attacks, we will go into detail on facial spoofing methods.

"Facial spoof attack is a process in which a fraudulent user can subvert or attack a face recognition system by masquerading as a registered user and thereby gaining illegitimate access and advantages."(C. Shiranthika [7])

The simplest attack that can be made is displaying an image, easily accessible since facial pictures are widely spread on social networks.

These types of attacks are both flat and static, favoring the improvements in more sophisticated techniques:

- Eye-cut photo attack: Printed photos with holes in the eyes regions in order to simulate blinking and eye movement.
- Warp photo attack: Printed photos that are bent in different directions in order to simulate face movement.
- Video attack: Looped video trying to look more natural than holding photos.
- 3D mask attacks: The only of the attacks mentioned that tries to tackle depth sensors.

Facial recognition is no longer something we see on science fiction movies. We can find facial recognition systems on multiple devices that we use in our daily life such as last generation mobile phones.

We can also find face recognition in other solutions to very diverse problems: employee detection for checking in, new payment methods, recognition in security cameras…

In this document, we will be describing the process of building an anti-spoofing system combining convolutional neural networks together with autoencoders in order to do an unsupervised pre-training of the model as a form of regularization.

2. State of the Art

Out of all the theory we have just spoken about, there are three main points we should focus about: Autoencoders (and successful cases where they have been applied to convolutional networks and more concrete for generalization purposes) and how neural networks have been tried as a solution to the anti-spoofing problem.

2.1. Convolutional Autoencoders

Fully connected autoencoders ignore the structure an image has, same as fully connected networks do in comparison with convolutional networks. What convolutional layers try to do is to localize different features repeating among the input.

In this sense, convolutional autoencoders share a similar architecture as the fully connected ones, except that their weights are shared in the whole input, preserving spatial locality.

Several autoencoders can be stacked one after another to form a deep hierarchy (proven to provide the most successful results in computer vision and object recognition), using them as a pre-trained version for a CNN with identical topology. (Vincent et al. [40])

This method of first pre-training the network and later follow the normal supervised techniques, combining both autoencoders and max pooling layers (D. Scherer, A. Müller, S. Behnke. [10]), makes the training stage a bit harder than usual, but provide some useful filters, outperforming randomly initialized networks.

It has been recognized for some time that generative models are less prone to overfitting than discriminant ones (A. Ng, M. Jordan. [3]) and it can be compared to the application of PCA as a pre-processing step before applying a classifier.

	1K	10K	50K
CNN (%)	7.63	2.21	0.79
CAE (%)	7.23	1.88	0.71

	1K	10K	50K
CNN (%)	55.52	35.23	22.50
CAE (%)	52.30	34.35	21.80

Table 3

2.1.1. Autoencoders Used for Generalization

The main problem deep learning approaches have to deal with, even if they have proved their potential, is training models with a great number of adaptive parameters, obtaining a highly non-convex objective function (possibility of finding many distinct local minima).

Ending in different minimums does not provide equivalent generalization errors, usually reaching regions of the parameter space with poorly generalization.

In Erhan et al. [11] we can find the hypothesis that, when using traditional gradient descent in the training process, the sequence of example used as input defines a trajectory in parameter space, converging in some sense. Small perturbations of that trajectory have more effect early on.

Hence, the parameters could be trapped in particular regions, influenced by the early examples (The author compares this process with the "critical period" observed in Bornstein 1987).

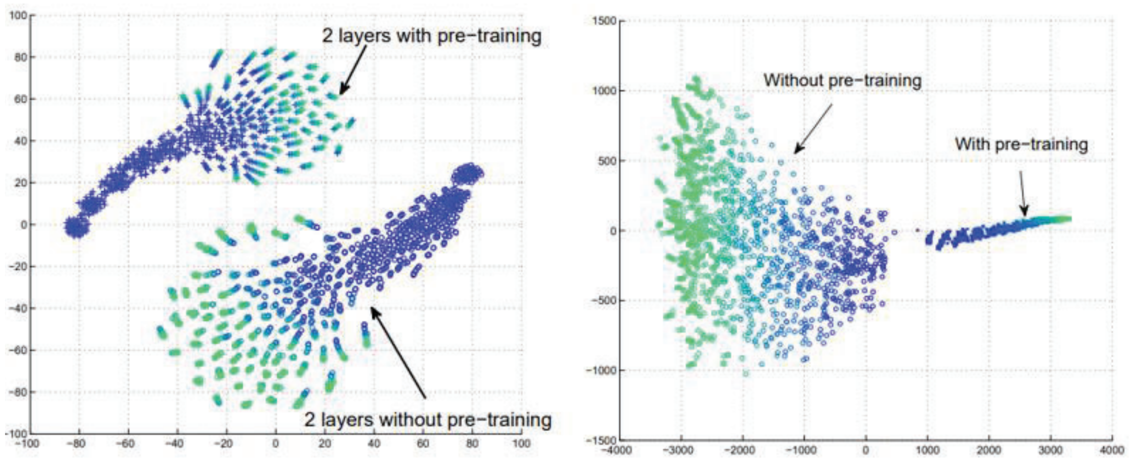


Figure 8

By reviewing the literature, we can find something in common in all the successful methods for training deep architectures: using an unsupervised learning algorithm at the level of a single layer (at the stage k, the k-th layer is trained, with respect to an unsupervised criterion, using as input the output of the previous layer, and while the previous layers are kept fixed). (Hinton 2006, Hinton and Salakhutdinov 2006, Bengio 2007, Ranzato 2007, Vincent 2008, Weston 2008, Ranzato 2008, Lee 2008).

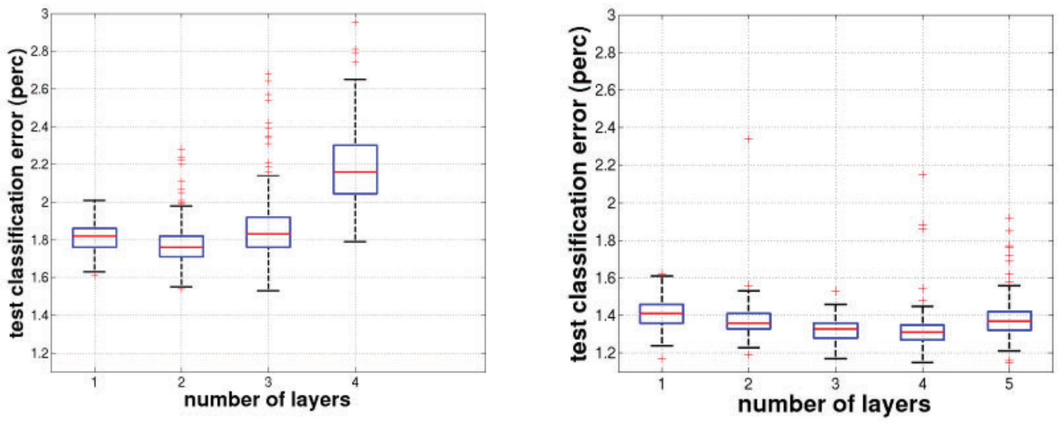


Figure 9

Vincent et al. [40] attempted at doing this by using stacked denoising autoencoders and Ranzato et al. [27][30] by using sparse autoencoders.

2.1.2. Layer-Wise Construction

Training deep neural networks was historically not an easy task due to diverse problems, both architectural and computational.

On one side, because of how the backpropagation algorithm for updating the weights of the network works, weights in hidden layers close to the output layer are updated normally, while weights in hidden layers close to the input layer are updated minimally or not at all. This is referred to as the vanishing gradient problem.

Nowadays, we know many ways to prevent the vanishing gradient problem such as better activation functions, weight initialization or variants of gradient descent. However, greedy layer-wise pretraining was what initially allowed the development of deep networks.

*The deep learning renaissance of 2006 began with the discovery that this greedy learning procedure could be used to find a good initialization for a joint learning procedure over all the layers, and that this approach could be used to successfully train even fully connected architectures. (*Page 528, Deep Learning, 2016.)

This pretraining consists on successively adding a new hidden layer to a model and training it while keeping the weights from the previous ones fixed.

It is called a greedy algorithm because it doesn't search for the global optimal solution, but it divides the problem in multiple more shallow problems, aggregating locally optimal solutions.

There are two different approaches when we want to carry out a pretraining for a supervised problem:

- Each layer learns on the same supervised task, successively adding layers to the model.
- Using the layer-wise construction to build an autoencoder, training it in an unsupervised manner and finally adding a supervised layer at the end of it.

Once all the desired layers are added to the model, it is common to fine tune them all together.

We can find different research about how and why does this algorithm work and how does it help with solving the overall task, such as Bengio et al [6]

	Experiment 2			Experiment 3		
	train.	valid.	test	train.	valid.	test
DBN, unsupervised pre-training	0%	1.2%	1.2%	0%	1.5%	1.5%
Deep net, auto-associator pre-training	0%	1.4%	1.4%	0%	1.4%	1.6%
Deep net, supervised pre-training	0%	1.7%	2.0%	0%	1.8%	1.9%
Deep net, no pre-training	.004%	2.1%	2.4%	.59%	2.1%	2.2%
Shallow net, no pre-training	.004%	1.8%	1.9%	3.6%	4.7%	5.0%

Table 4

Other works (Y. Zhou [41]) have tried to improve this algorithm by combining both the idea of training the model layer-wise to avoid vanishing gradient problems and training the model as a whole to keep the influence from one layer to the previous ones.

Dataset/Algorithm	SDAE-3	SDAE-3-Joint
MNIST	2.28%	**1.55%**
MNIST-basic	4.17%	**3.13%**
MNIST-rot	21.57%	**16.06%**
MNIST-bg-img	**28.91%**	29.44%
MNIST-bg-rand	13.50%	**12.57%**
MNIST-bg-img-rot	**59.43%**	61.17%
rect	6.29%	**4.95%**
rect-img	25.12%	**23.90%**
convex	28.53%	**25.23%**

Table 5

2.2. Anti-Spoofing

When applying deep neural networks to try to solve the anti-spoofing problem, we can find various papers focusing on different characteristics that may set a difference between real and spoofing.

Li et al. [24] attempted at applying Fourier spectra for measuring biometrics related to face texture.

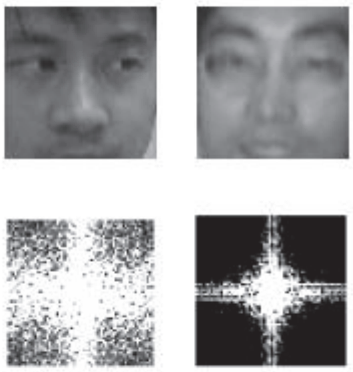


Figure 10

Freitas et al. [12] and Zang et al. [43] also tried to tackle the problem by trying to identify textural features (LBP-TOP and Multispectral Reflectance Distributions, respectively).

Other papers go instead to motion based approaches, from eye-blinking (Sun et al. [35]), to lip movement (Kollreider et al. [21]) or even verifying the fitness between video and audio signals (G. Chetty, M. Wagner [13]) .

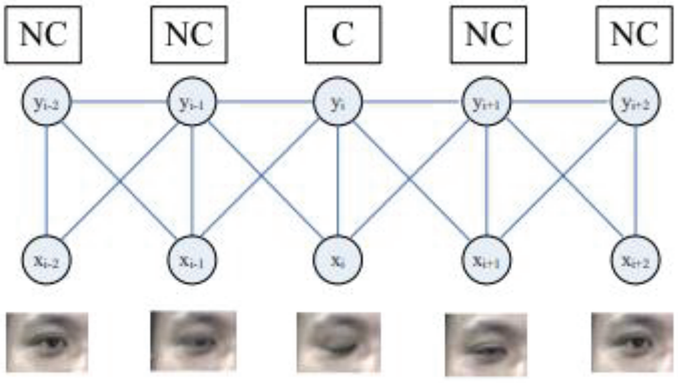


Figure 11

As opposed to all the previously mentioned binary approaches (real vs fake), other works such as Liu et al. [26] added auxiliary supervision and proposed a deep model that uses the supervision from both the spatial and temporal auxiliary information.

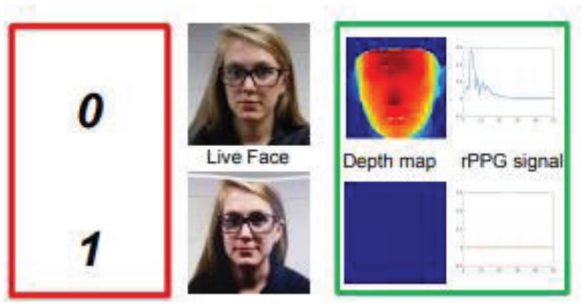


Figure 12

2.2.1. Face Detection with OpenCV

OpenCV provides us with two face detection classifiers: Haar and LBP.

Haar:

The Haar Classifier is a method designed by P. Viola and M. Jones [29] that attempts at extracting Haar features from each image, discarding irrelevant ones using AdaBoost.

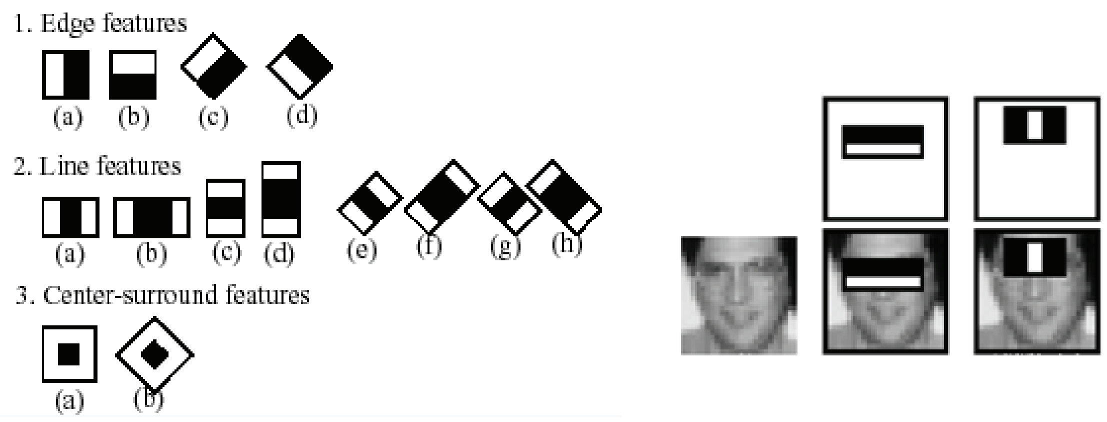


Figure 13

LBP:

The LBP Classifier (T. Ahonen, M. Pietikäinen [39]) tries to form a feature vector by dividing each image into blocks and applying a window over each block.

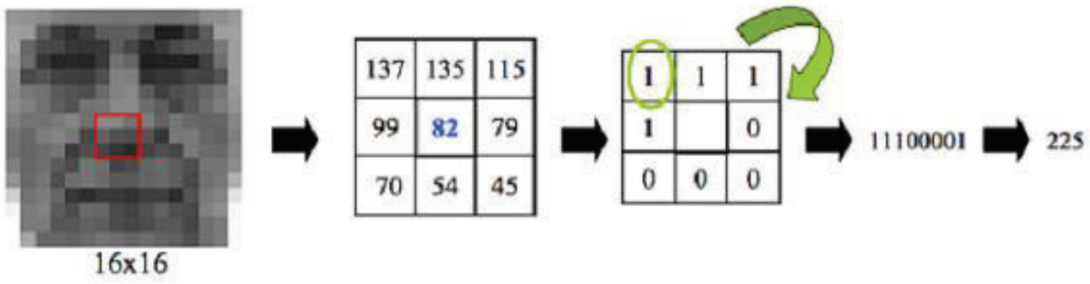


Figure 14

For each window, it compares its central pixel value with its surroundings, assigning a 1 if they are greater or 0 if they are lower.

Finally, starting from the top left corner and in a clockwise direction, it forms a binary number which is converted into an integer.

3. Project development

In this section we will be describing the data used in the experiment, how we handled it and the methods used to develop our experiment.

The architecture and training of the networks was done using Keras in the Google Colab environment, to be able to make use of the GPU and speed up the process.

3.1. Dataset

The dataset used in this project is the NUAA Photograph Imposter Database (taken from Tan et al. [38]). We can choose from three different image formats (no pre-processing, with already applied face detection or already normalized), from which we picked the first one.

It consists of 12614 images (5105 shots of real people and 7509 shots of pictures of that same people) of 15 subjects under different conditions.

To assure this variations, a series of videos were taken in different sessions using a webcam, were participants were asked to look frontally to the camera, trying to resemble a picture as much as possible (neutral expressions, no blinks or head movements…). From this videos, a set of frames was hand-picked to form the final dataset.

For the picture cases, photographs were taken using a high definition camera and printed on photographic paper. This paper was then moved around the vision of the webcam and folded in different directions.

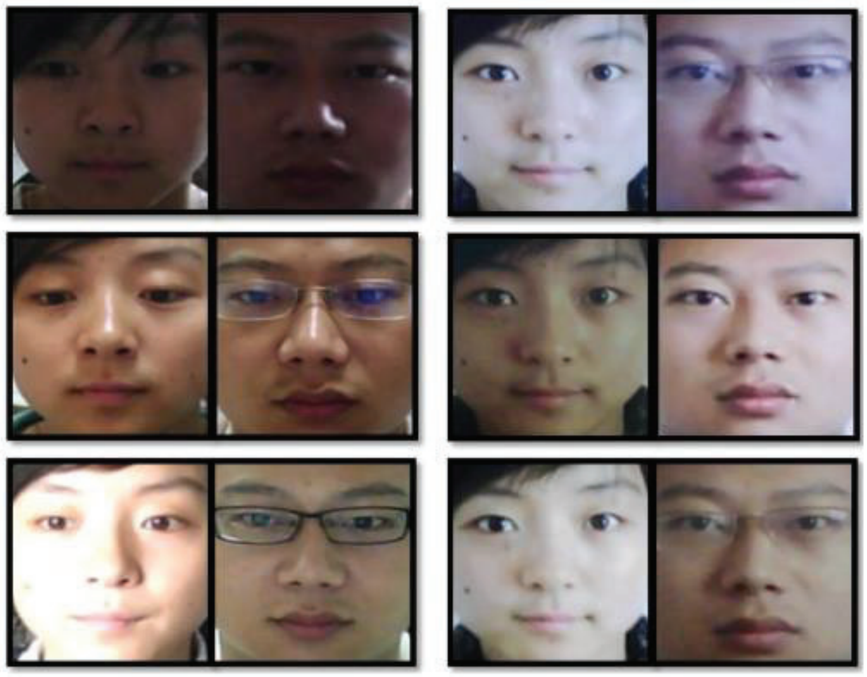


Figure 15

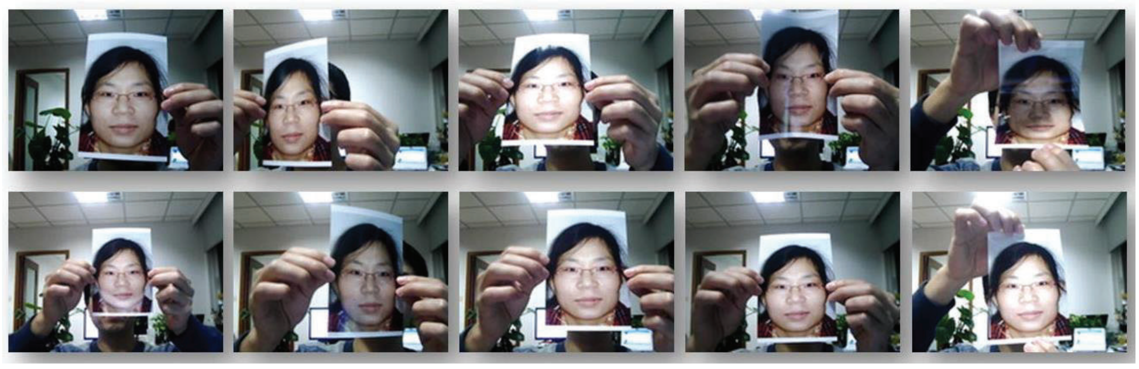


Figure 16

The creators of the dataset distributed the pictures of the different sessions along a training set and a testing set, making sure that there is no overlapping between them (the same subject in a specific session can't be in both sets at the same time).

The sizes of this sets are: 8878 for the test set (5743 imposter and 3135 real people) and 3045 for the train set (1872 imposter and 1173 real people).

Since there is no validation set and, usually, the distribution of data is 60-20-20 for train-test-validation, we decided to redistribute the already established sets as follows: take the train set as the validation and divide the test set, around 70% for the new training set and the remaining for the test set (still conserving the non-overlapping property).

Once the data has been selected and reorganized, we can do some cleaning at the same time as we start the face detection phase.

For this purpose, we use the already existing face detector *haarcascade_frontalface_default.xml* provide in the OpenCV library and we run it over the dataset, taking away some cases were the detection wasn't successful.

While we are manually checking the results from the OpenCV detector, we also discard those pictures that we don't consider of having a good enough quality.

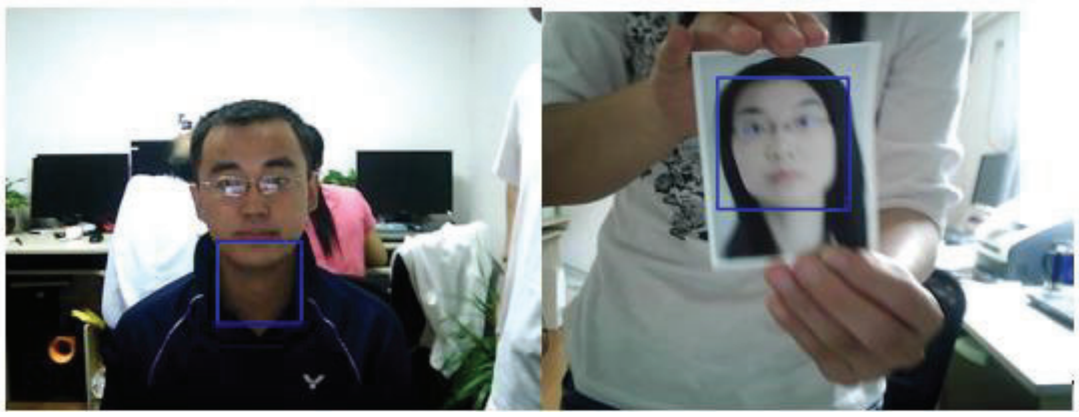


Figure 17

Finally, the faces are resized into a 256x256 square before using them as an input to the network.

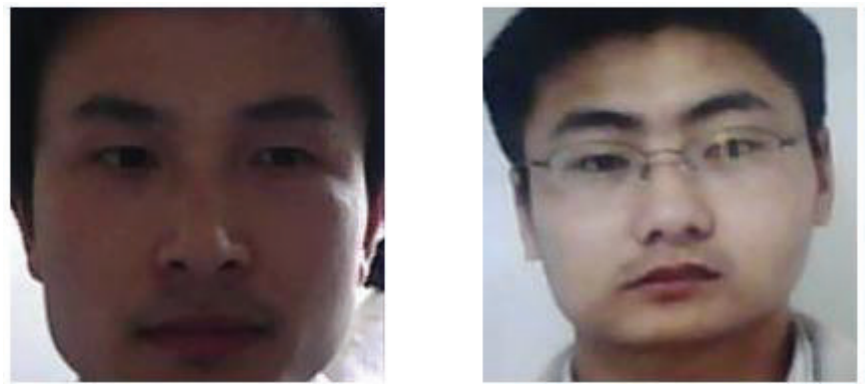


Figure 18

	Train	Validation	Test
Real	2193	1173	942
Spoofing	3926	1844	1577
TOTAL	**6119**	**3017**	**2519**

Table 6

3.2. CNN

Once the data is prepared, we can start working on our anti-spoofing neural network.

First, we will work on a regular CNN, with no pre-training or initialization weights. The architecture of such network is a combination of convolutional and max pooling layers, ending with a flattening and a dense layer. A summary, taken from the Keras API, can be found in the Annexes.

An early stopping is set, comparing the training and the validation loss with a patience of 20 epochs, to know when to stop training.

This is the first result we get:

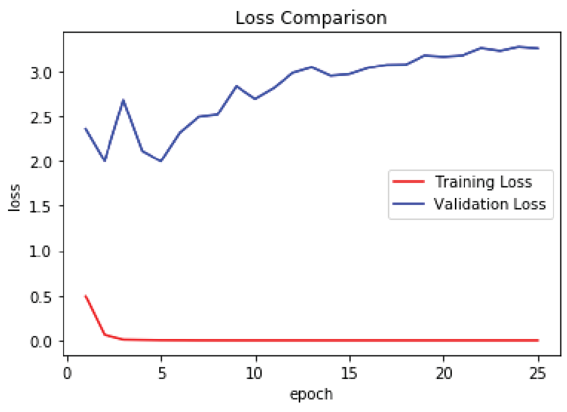


Figure 19

We can clearly see how our network is overfitting, since the training loss approaches 0 really fast while the validation loss starts increasing in the early epochs.

The main methods to avoid or at least try to reduce the overfitting are:

- Adding more data. Not possible in our case, since we are working with an already existing dataset and not one we built ourselves.
- Use another kind of architecture or reduce the complexity of it. We can't change the type of the architecture since it is the use case we are interested in for our experiment. In the case of the complexity, other tries were made as a pre-step to find a good architecture and the best one out of them was picked.
- Use data augmentation. This is applicable to our case. We will try doing data augmentation on the training set by doing vertical flips and varying the shear, the range and the brightness of the images.

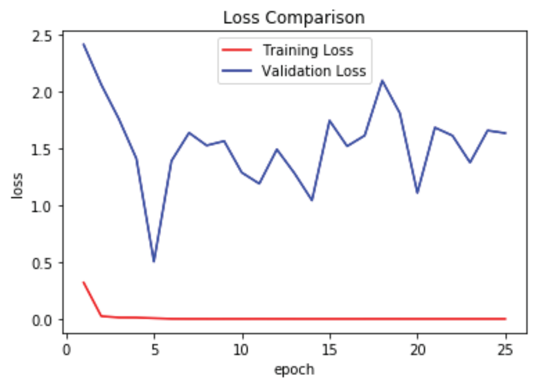


Figure 20

In this case, while we are still overfitting, the validation loss doesn't abruptly start increasing, suggesting that we may be going the right way.

Finally, we will try dropout as a regularization technique.

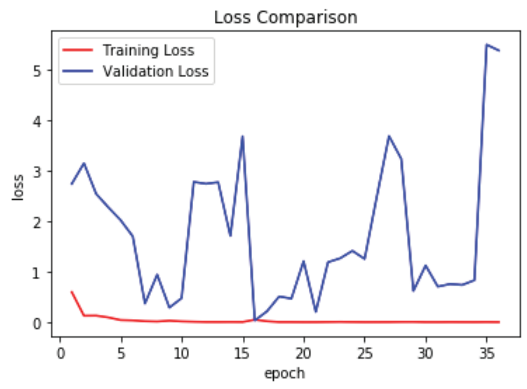


Figure 21

The gap between both losses is lower than in previous cases, becoming almost unrecognizable at some point, but while the training loss follows a steady decline, the validation loss is more irregular.

Since we were not able to reach satisfactory results, we will try to add a pre-training of the weights and check if this works as a better regularization method.

As the pre-training step we will implement an AE.

3.3. Autoencoder

To build an undercomplete AE, we should take the convolutional part of our previous net (with no flattening and dense layer) and add a symmetrical network to it, changing the pooling layers for upsampling ones.

The architecture of this network can also be found in the Annexes.

As it can be seen in the summary provided by Keras, this model counts with almost 19 million parameters, what leads to memory problems if we want to train them at the same time.

To avoid such problems, we will build the AE using layer-wise construction.

To do this, we will build 4 small AE (their architectures can also be found in the Annexes), train them separately and transfer their learnt weights into the original CNN.

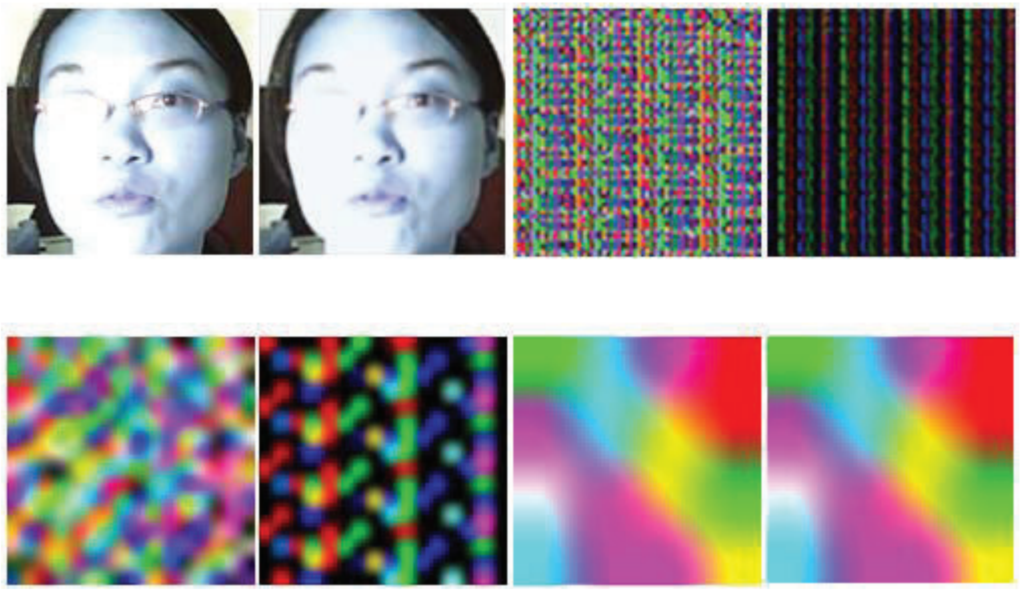


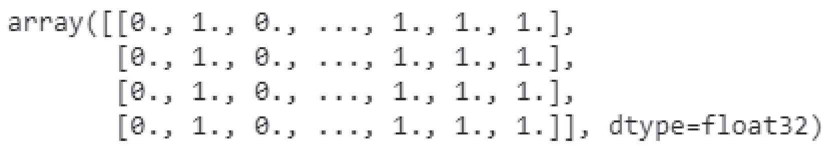
Figure 22

We can observe different behaviors as we go deeper into the network:

- The first autoencoder takes a face as an input and outputs a slightly lower quality version of it.

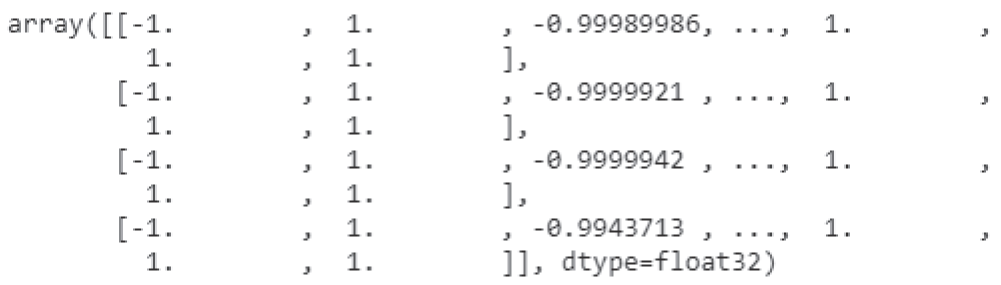
- The second and third autoencoders take as an input the final code of the previous encoder and output some kind of a denoised version of it.

- Finally, the fourth and last autoencoder take as an input the final code of the third autoencoder and returns almost the same result (at least graphically; if we do a numerical comparison we can observe how input and output are not exactly the same).



```
array([[0., 1., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 1., 1., 1.]], dtype=float32)
```

-



```
array([[-1.        ,  1.        , -0.99989986, ...,  1.        ,
         1.        ,  1.        ],
       [-1.        ,  1.        , -0.9999921 , ...,  1.        ,
         1.        ,  1.        ],
       [-1.        ,  1.        , -0.9999942 , ...,  1.        ,
         1.        ,  1.        ],
       [-1.        ,  1.        , -0.9943713 , ...,  1.        ,
         1.        ,  1.        ]], dtype=float32)
```

-

- *Figure 23*

We can also see how different they act by comparing their loss evolution during training:

- The first autoencoder presents a steady progression, corroborating the theory about the first layers being the ones that learn the most out of the data.

- The second and third autoencoders start showing a fixed loss while they still improve their training.

- The fourth autoencoder presents a fixed loss function, as expected since we have reached a deep level of the network.

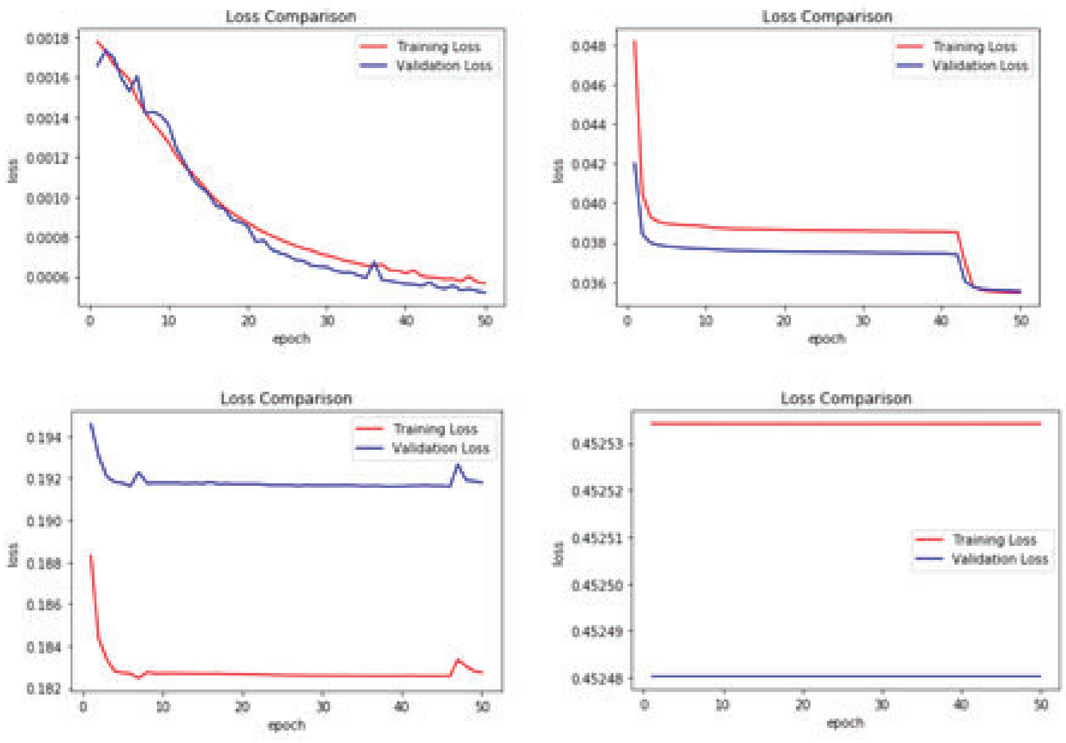


Figure 24

Finally, we transfer the weights learnt on the autoencoders, add a dense layer to do the classification and retrain the CNN.

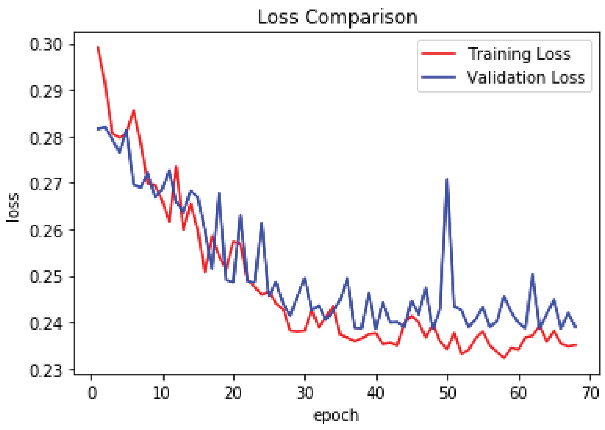


Figure 25

4. Results

As we can see on the last Figure, we have improved the results obtained from the others where we applied other regularization techniques.

- The model no longer presents overfitting.
- Both training and validation losses show the same tendency.
- While the validation loss is a bit more irregular than the training loss, it is something explainable from the use of mini-batches and in no case comparable to our previous results.
- The training loss is a bit higher than the ones we obtained previously; however, it is preferable losing some accuracy in the training set if, in exchange, our model performs better on unseen data.

It is interesting to notice that in none of the autoencoders we find any trace of overfitting, suggesting that the combination of local non-overfitting solutions approximates to a good global non-overfitting solution.

When calculating the test accuracy, we get approximately a 65%. However, the goal of this research was not to build a high-accuracy anti-spoofing system but one that can generalize well on unseen data.

Also, even if we tried to build a test set as different as possible from the training set, the data may be too similar to be able to consider them independent.

5. Conclusions and future development

Along this essay we have presented the field of computer vision and some of the main problems that are studied in it: image classification and object detection.

Also, we have introduced the concept of the autoencoders, how they work and how can they be helpful for generalization purposes. Even if nowadays the most common practice is applying transfer learning and using some of the already existing deep networks, this approach could be useful as a weight initialization for those cases were we want to build a network of our own.

We have analysed and put into practice the layer-wise construction method and done some research on how other data scientists have tried to prevent spoofing attacks.

In conclusion, in our opinion, autoencoders shouldn’t be used on their own due to the low accuracy obtained by using them.

However, the elimination of overfitting that couldn’t be achieved with other regularization techniques suggests that it may be useful to use them as a previous step for some other algorithms.

For future developments, other kind of autoencoders could be constructed and compared between each other and with the one built here.

Also, other datasets could be obtained to test how well does it generalize in those cases.

It would have been nice if we would have been able to compare the layer-wise construction with a deep autoencoder. However, a computer with GPU would be needed for that since Google Colab has a limit of RAM and GPU that can be used.

The final CNN model has been saved so an anti-spoofing system can be implemented.

Bibliography

[1] A. Bonner. "The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks and Image Classification", 2019. [Online] Available: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb [Accessed: March 2019]

[2] A. Ng. "Sparse Autoencoder. Lecture notes", 2011. [Online] Available: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf [Accessed: June 2019]

[3] A. Ng, M. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, Advances in Neural Information Processing Systems 14 (NIPS'01), pp. 841–848, 2002.

[4] A. Ouaknine. "Review of Deep Learning Algorithms for Image Classification", 2018. [Online] Available: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-image-classification-5fdbca4a05e2 [Accessed: March 2019]

[5] A. Yu. "How To Teach A Computer To See With Convolutional Neural Networks", 2018. [Online] Available: https://towardsdatascience.com/how-to-teach-a-computer-to-see-with-convolutional-neural-networks-96c120827cd1 [Accessed: March 2019]

[6] Bengio et al. "Greedy Layer-Wise Training of Deep Networks", 2006. Université de Montréal.

[7] C. Shiranthika. "Face Spoof Detection", 2019. [Online] Available: https://medium.com/datadriveninvestor/face-spoof-detection-e0d08fb246ea [Accessed: April 2019]

[8] S. J.D. Prince. Computer Vision: Models, Learning and Inference. Cambridge University Press, 2012

[9] Goodfellow et al. Deep Learning. MIT Press, 2016

[10] D. Scherer, A. Müller, S. Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition", 2010. University of Bonn, Autonomous Intelligent Systems Group.

[11] Erhan et al. "Why does Unsupervised Pre-Training Help Deep Learning?", 2010. Université de Montréal.

[12] Freitas et al. "LBP – TOP Based Countermeasure Against Face Spoofing Attacks", 2012. University of Campinas.

[13] G. Chetty, M. Wagner. “Audio-visual Multimodal Fusion for Biometric Person Authentication and Liveness Verification”, 2005. University of Canberra

[14] H. Hubel, T. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”, 1962. The Physiological Society

[15] I. Itzcovich. “CPU Real Time Face Detection using Deep Learning”, 2018. [Online] Available: https://towardsdatascience.com/faced-cpu-real-time-face-detection-using-deep-learning-1488681c1602 [Accessed: March 2019]

[16] J. Brownlee. “How to Use Greedy Layer-Wise Pretraining in Deep Learning Neural Networks”, 2019. [Online] Available: https://machinelearningmastery.com/greedy-layer-wise-pretraining-tutorial/ [Accessed: June 2019]

[17] J. Jordan. “Introduction to Autoencoders”, 2018. [Online] Available: https://www.jeremyjordan.me/autoencoders/ [Accessed: April 2019]

[18] J. Masci. “Stacked Convolutional Autoencoders for Hierarchical Feature Extraction”, 2011. Istituto Dalle Molle di Studi sull’Intelligenza Artificiale

[19] K. He et al. “Deep Residual Learning for Image Recognition”, 2015

[20] K. Simonyan, A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”, 2014

[21] Kollreider et al. “Real-Time Face Detection and Motion Analysis with Application in Liveness Assessment”, 2007

[22] Krizhevsky et al. “ImageNet Classification with Deep Convolutional Neural Networks”, 2012. Neural Information Processing Systems Conference.

[23] L. Sánchez. “Local Binary Patterns Applied to Face Detection and Recognition”, 2010. Universitat Politècnica de Catalunya.

[24] Li et al. “Live Face Detection Based on the Analysis of Fourier Spectra”, 2004. Chinese Academy of Science, Beijing

[25] Liu et al. “SSD: Single Shot MultiBox Detector”, 2016

[26] Liu et al. “Learning Deep Models for Face Anti-Spoofing: Binary or Auxiliary Supervision”, 2018. Michigan State University.

[27] M. Ranzato, Y. Bourear, Y. LeCun. “Sparse Feature Learning for Deep Belief Networks”, 2008. New York University

[28] O. Cohen. “PCA vs Autoencoders”, 2018. [Online] Available: https://towardsdatascience.com/pca-vs-autoencoders-1ba08362f450 [Accessed: March 2019]

[29] P. Viola, M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features”, 2004. Mitsubishi Electric Research Lab.

[30] Ranzato et al. “Efficient Learning of Sparse Representations with an Energy-Based Model”, 2007. New York University

[31] Ranzato et al. “Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition”, 2007. New York University

[32] Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”, 2016. IEEE Conference on Computer Vision and Pattern

[33] Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2016

[34] Rifai et al. “Contractive Auto-Econders: Explicit Invariance During Feature Extraction”, 2011. Université de Montréal

[35] Sun et al. “Blinking-Basd Live Face Detection Using Conditional Random Fields”, 2007. International Conference on Advances in Biometrics

[36] S. Remanan. “Beginner’s Guide to Object Detection Algorithms”, 2019. [Online] Available: https://towardsdatascience.com/beginners-guide-to-object-detection-algorithms-6620fb31c375 [Accessed: May 2019]

[37] Szegedy et al. “Going Deeper with Convolutions”, 2014. IEEE Conference on Computer Vision

[38] Tan et al. “Face Liveness Detection from a Single Image with Sparse Low Rank Bilinear Discriminative Model”, 2010. Nanjang University of Aeronautics and Astronomics.

[39] T. Ahonen, M. Pietikäinen. “Face Description with Local Binary Patterns: Application to Face Recognition”, 2006. Oulu University

[40] Vincent et al. “Extracting and Composing Robust Features with Denoising Autoencoders”, 2008. Université de Montréal

[41] Yingbo Zhou. “Learning Deep Autoencoders without Layer-Wise Training”, 2014

[42] Yuxuang et al. “Using Supervised Pretraining to Improve Generalization of Neural Networks on Binary Classification Problems”, 2018. University of Auckland

[43] Zang et al. “Face Liveness Detection by Learning Multispectral Reflectance Distributions”, 2011. Institute of Automation, Chinese Academy of Science

Annexes

Architecture of the CNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 8)       224
_________________________________________________________________
activation_1 (Activation)    (None, 256, 256, 8)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256, 256, 8)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
activation_2 (Activation)    (None, 128, 128, 16)      0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 128, 16)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
activation_3 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
activation_5 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
activation_6 (Activation)    (None, 8, 8, 256)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 256)         0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
```

```
_________________________________________________________________
activation_7 (Activation)    (None, 4, 4, 512)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 4, 4, 512)         0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2, 2, 1024)        4719616
_________________________________________________________________
activation_8 (Activation)    (None, 2, 2, 1024)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 2, 2, 1024)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 1, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
activation_9 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_9 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2050
_________________________________________________________________
activation_10 (Activation)   (None, 2)                 0
=================================================================
Total params: 6,295,378
Trainable params: 6,295,378
Non-trainable params: 0
```

Architecture of the AE

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_65 (Conv2D)           (None, 256, 256, 8)       224
_________________________________________________________________
activation_36 (Activation)   (None, 256, 256, 8)       0
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 128, 128, 16)      1168
_________________________________________________________________
activation_37 (Activation)   (None, 128, 128, 16)      0
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 64, 64, 32)        4640
_________________________________________________________________
activation_38 (Activation)   (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
```

```
________________________________________________________________________
activation_39 (Activation)    (None, 32, 32, 64)        0
________________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 16, 16, 64)        0
________________________________________________________________________
conv2d_69 (Conv2D)            (None, 16, 16, 128)       73856
________________________________________________________________________
activation_40 (Activation)    (None, 16, 16, 128)       0
________________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 8, 8, 128)         0
________________________________________________________________________
conv2d_70 (Conv2D)            (None, 8, 8, 256)         295168
________________________________________________________________________
activation_41 (Activation)    (None, 8, 8, 256)         0
________________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 4, 4, 256)         0
________________________________________________________________________
conv2d_71 (Conv2D)            (None, 4, 4, 512)         1180160
________________________________________________________________________
activation_42 (Activation)    (None, 4, 4, 512)         0
________________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 2, 2, 512)         0
________________________________________________________________________
conv2d_72 (Conv2D)            (None, 2, 2, 1024)        4719616
________________________________________________________________________
activation_43 (Activation)    (None, 2, 2, 1024)        0
________________________________________________________________________
max_pooling2d_48 (MaxPooling (None, 1, 1, 1024)        0
________________________________________________________________________
up_sampling2d_25 (UpSampling (None, 2, 2, 1024)        0
________________________________________________________________________
conv2d_73 (Conv2D)            (None, 2, 2, 1024)        9438208
________________________________________________________________________
activation_44 (Activation)    (None, 2, 2, 1024)        0
________________________________________________________________________
up_sampling2d_26 (UpSampling (None, 4, 4, 1024)        0
________________________________________________________________________
conv2d_74 (Conv2D)            (None, 4, 4, 256)         2359552
________________________________________________________________________
activation_45 (Activation)    (None, 4, 4, 256)         0
________________________________________________________________________
up_sampling2d_27 (UpSampling (None, 8, 8, 256)         0
________________________________________________________________________
conv2d_75 (Conv2D)            (None, 8, 8, 256)         590080
________________________________________________________________________
activation_46 (Activation)    (None, 8, 8, 256)         0
________________________________________________________________________
up_sampling2d_28 (UpSampling (None, 16, 16, 256)       0
________________________________________________________________________
conv2d_76 (Conv2D)            (None, 16, 16, 64)        147520
________________________________________________________________________
activation_47 (Activation)    (None, 16, 16, 64)        0
________________________________________________________________________
up_sampling2d_29 (UpSampling (None, 32, 32, 64)        0
________________________________________________________________________
conv2d_77 (Conv2D)            (None, 32, 32, 64)        36928
________________________________________________________________________
```

```
_________________________________________________________________
activation_48 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
up_sampling2d_30 (UpSampling (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_78 (Conv2D)           (None, 64, 64, 16)        9232
_________________________________________________________________
activation_49 (Activation)    (None, 64, 64, 16)        0
_________________________________________________________________
up_sampling2d_31 (UpSampling (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_79 (Conv2D)           (None, 128, 128, 16)      2320
_________________________________________________________________
activation_50 (Activation)    (None, 128, 128, 16)      0
_________________________________________________________________
up_sampling2d_32 (UpSampling (None, 256, 256, 16)      0
_________________________________________________________________
conv2d_80 (Conv2D)           (None, 256, 256, 3)       435
_________________________________________________________________
activation_51 (Activation)    (None, 256, 256, 3)       0
=================================================================
Total params: 18,877,603
```

Architecture of Layer-Wise AE 1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 256, 256, 8)       224
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 128, 128, 16)      1168
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 64, 64, 16)        0
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 128, 128, 16)      2320
_________________________________________________________________
up_sampling2d_10 (UpSampling (None, 256, 256, 16)      0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 256, 256, 3)       435
=================================================================
Total params: 4,147
```

Architecture of Layer-Wise AE 2

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_6 (InputLayer)         (None, 64, 64, 16)        0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 64, 64, 32)        4640      
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 32, 32, 64)        18496     
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 64)        0         
_________________________________________________________________
up_sampling2d_11 (UpSampling (None, 32, 32, 64)        0         
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 32, 32, 64)        36928     
_________________________________________________________________
up_sampling2d_12 (UpSampling (None, 64, 64, 64)        0         
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 64, 64, 16)        9232      
=================================================================
Total params: 69,296
```

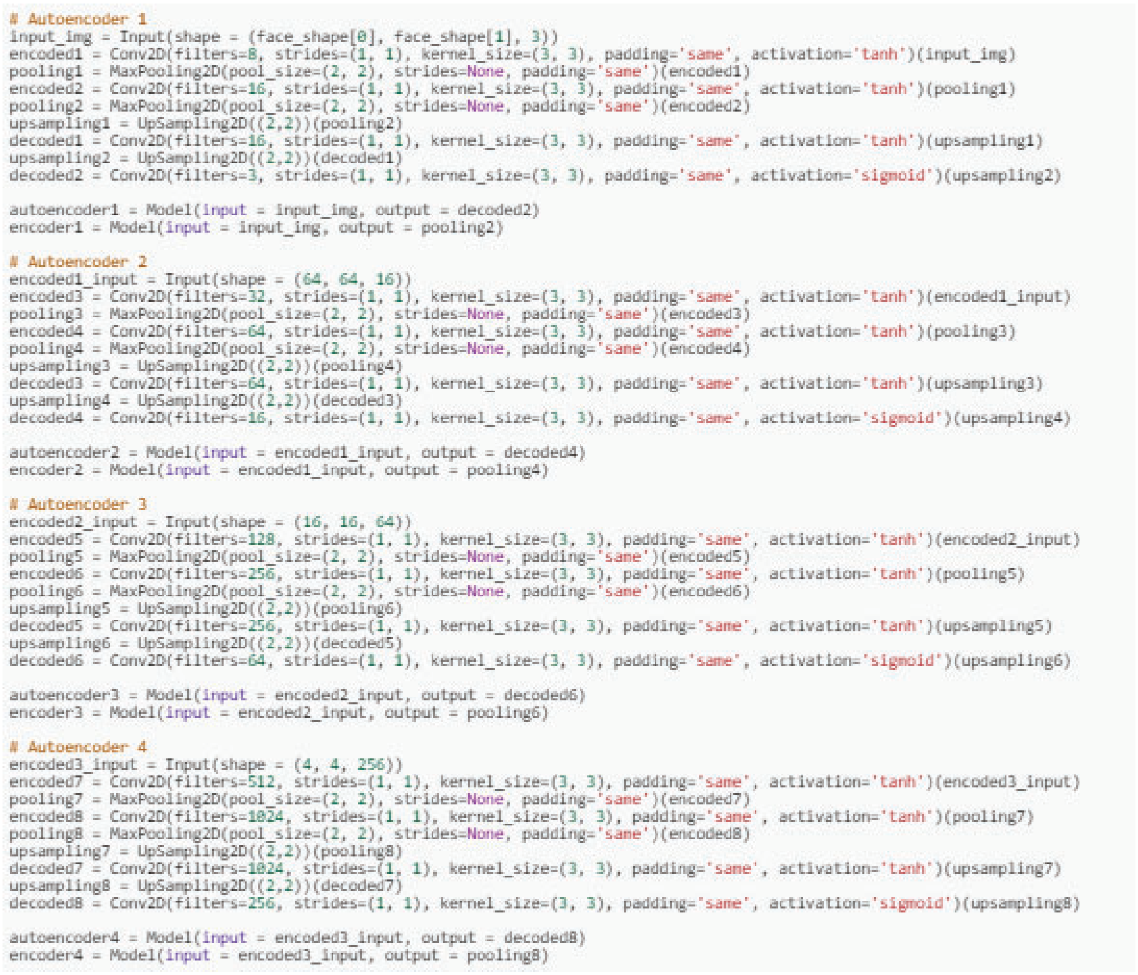
Architecture of Layer-Wise AE 3

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_7 (InputLayer)         (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 16, 16, 128)       73856     
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 8, 8, 128)         0         
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 8, 8, 256)         295168    
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 4, 4, 256)         0         
_________________________________________________________________
up_sampling2d_13 (UpSampling (None, 8, 8, 256)         0         
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 8, 8, 256)         590080    
_________________________________________________________________
up_sampling2d_14 (UpSampling (None, 16, 16, 256)       0         
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 16, 16, 64)        147520    
=================================================================
Total params: 1,106,624
```

Architecture of Layer-Wise AE 4

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_8 (InputLayer)         (None, 4, 4, 256)         0         
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 4, 4, 512)         1180160   
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 2, 2, 512)         0         
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 2, 2, 1024)        4719616   
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 1, 1, 1024)        0         
_________________________________________________________________
up_sampling2d_15 (UpSampling (None, 2, 2, 1024)        0         
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 2, 2, 1024)        9438208   
_________________________________________________________________
up_sampling2d_16 (UpSampling (None, 4, 4, 1024)        0         
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 4, 4, 256)         2359552   
=================================================================
Total params: 17,697,536
```

Keras Code for Building a Layer-Wise Deep Autoencoder



```
# Autoencoder 1
input_img = Input(shape = (face_shape[0], face_shape[1], 3))
encoded1 = Conv2D(filters=8, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(input_img)
pooling1 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded1)
encoded2 = Conv2D(filters=16, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(pooling1)
pooling2 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded2)
upsampling1 = UpSampling2D((2,2))(pooling2)
decoded1 = Conv2D(filters=16, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(upsampling1)
upsampling2 = UpSampling2D((2,2))(decoded1)
decoded2 = Conv2D(filters=3, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='sigmoid')(upsampling2)

autoencoder1 = Model(input = input_img, output = decoded2)
encoder1 = Model(input = input_img, output = pooling2)

# Autoencoder 2
encoded1_input = Input(shape = (64, 64, 16))
encoded3 = Conv2D(filters=32, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(encoded1_input)
pooling3 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded3)
encoded4 = Conv2D(filters=64, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(pooling3)
pooling4 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded4)
upsampling3 = UpSampling2D((2,2))(pooling4)
decoded3 = Conv2D(filters=64, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(upsampling3)
upsampling4 = UpSampling2D((2,2))(decoded3)
decoded4 = Conv2D(filters=16, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='sigmoid')(upsampling4)

autoencoder2 = Model(input = encoded1_input, output = decoded4)
encoder2 = Model(input = encoded1_input, output = pooling4)

# Autoencoder 3
encoded2_input = Input(shape = (16, 16, 64))
encoded5 = Conv2D(filters=128, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(encoded2_input)
pooling5 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded5)
encoded6 = Conv2D(filters=256, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(pooling5)
pooling6 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded6)
upsampling5 = UpSampling2D((2,2))(pooling6)
decoded5 = Conv2D(filters=256, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(upsampling5)
upsampling6 = UpSampling2D((2,2))(decoded5)
decoded6 = Conv2D(filters=64, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='sigmoid')(upsampling6)

autoencoder3 = Model(input = encoded2_input, output = decoded6)
encoder3 = Model(input = encoded2_input, output = pooling6)

# Autoencoder 4
encoded3_input = Input(shape = (4, 4, 256))
encoded7 = Conv2D(filters=512, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(encoded3_input)
pooling7 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded7)
encoded8 = Conv2D(filters=1024, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(pooling7)
pooling8 = MaxPooling2D(pool_size=(2, 2), strides=None, padding='same')(encoded8)
upsampling7 = UpSampling2D((2,2))(pooling8)
decoded7 = Conv2D(filters=1024, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='tanh')(upsampling7)
upsampling8 = UpSampling2D((2,2))(decoded7)
decoded8 = Conv2D(filters=256, strides=(1, 1), kernel_size=(3, 3), padding='same', activation='sigmoid')(upsampling8)

autoencoder4 = Model(input = encoded3_input, output = decoded8)
encoder4 = Model(input = encoded3_input, output = pooling8)
```

Glossary

CNN:	Convolutional Neural Network
AE:	Autoencoder
NMP:	Non-Maximum Suppression
MSE:	Mean Squared Error
PCA:	Principal Component Analysis
SDAE:	Stacked Denoising Autoencoder
CAE:	Convolutional Autoencoder