UNIVERSITAT DE BARCELONA

Bachelor's Thesis

Double bachelor's degree in Mathematics and Computer Science

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Out of distribution image detection

Author: Raül Villalba Rodríguez

Directors:	**Dr. Santi Seguí**
	Dr. Xavier Guitart
Developed in:	**Departament de matemàtiques**
	i informàtica (Universitat de Barcelona)

Barcelona, **21st June 2020**

Contents

Resum

En els darrers anys, de la mà de l'evolució de la tecnologia i la intel·ligència artificial, el camp de la medicina ha viscut un canvi de paradigma, i altres ciencies s'han introduït directement a la medicina. En el cas d'aquest projecte, es treballa amb un conjunt d'imatges, obtingudes mitjançant una pildora endoscòpica, la qual el pacient s'empassa, i va transmetent imatges dels intestins a un dispositiu extern. Aquest conjunt d'imatges ja ha estat tractat, però un element important en la prevenció de malalties és l'existència d'hemorragies, sang o altres elements als intestins. L'objectiu d'aquest projecte és el d'automatitzar la cerca d'elements extranys en les imatges preses per aquestes càmeres inalambriques, per així estalviar feina als doctors. Per tal de fer això, s'aplica un algorisme de detecció d'imatges de fora de la distribució. En altres paraules, mitjançant l'entrenament d'una xarxa neuronal, es determina si una imatge pertany al mateix tipus d'imatges amb les quals s'ha entrenat la xarxa, o no. Això es fa mitjançant una modificació de la funció de classificació original de la xarxa, aplicant un llindar, el qual determina si la imatge pertany o no a la distribució d'entrenament, a més a més d'un tractament previ de les imatges. En aquesta memòria, es fa una descripció del funcionament i l'estructura de les xarxes neuronals, fent un incís en les xarxes neuronals convolucionals, les quals són les més utilitzades per al tractament d'imatges, i per tant són les que s'utilitzen en aquest projecte, es descriu el mètode de classificació implementat, així com es descriu la seva configuració i els resultats obtinguts amb la implementació donada.

2010 Mathematics Subject Classification. 68T20

Resumen

En los últimos años, de la mano de la evolución de la tecnología y la inteligencia artificial, el campo de la medicina ha vivido un cambio de paradigma, y otras ciencias se han introducido directement a la medicina. En el caso de este proyecto, se trabaja con un conjunto de imágenes, obtenidas mediante una píldora endoscópica, la cual el paciente traga, y transmite imágenes de los intestinos a un dispositivo externo. Este conjunto de imágenes ya ha sido tratado, pero un elemento importante en la prevención de enfermedades es la existencia de hemorragias, sangre u otros elementos en los intestinos. El objetivo de este proyecto es el de automatizar la búsqueda de elementos extraños en las imágenes tomadas por estas cámaras inalámbricas, para así ahorrar trabajo a los doctores. Para hacer esto, se aplica un algoritmo de detección de imágenes fuera de la distribución. En otras palabras, mediante el entrenamiento de una red neuronal, se determina si una imagen pertenece al mismo tipo de imágenes con las que se ha entrenado la red, o no. Esto se hace mediante una modificación de la función de clasificación original de la red, aplicando un umbral, el cual determina si la imagen pertenece o no a la distribución de entrenamiento, además de un tratamiento previo de las imágenes. En esta memoria, se hace una descripción del funcionamiento y la estructura de las redes neuronales, haciendo un inciso en las redes neuronales convolucionales, las cuales son las más utilizadas para el tratamiento de imágenes, y por lo tanto son las que se utilizan en este proyecto, se describe el método de clasificación implementado, así como se describe su configuración y los resultados obtenidos con la implementación dada.

2010 Mathematics Subject Classification. 68T20

Abstract

In recent years, with the evolution of technology and artificial intelligence, the field of medicine has undergone a paradigm shift, and other sciences have been introduced directly into medicine. In the case of this project, we work with a set of images, obtained by a wireless capsule endoscopy, which the patient swallows, and transmits images of the intestines to an external device. This set of images has already been treated, but an important element in disease prevention is the existence of bleeding, blood or other elements in the intestines. The aim of this project is to automate the search for strange elements in the images taken by these wireless cameras, in order to save work for doctors. To do this, an out-of-distribution image detection algorithm is applied. In other words, by training a neural network, it is determined whether an image belongs to the same type of images with which the network has been trained, or not. This is done by a modification of the original classification function of the network, applying a threshold, which determines whether the image belongs to the training distribution or not, in addition to a pre-processing of the images. In this report, a description of the functioning and structure of the neural networks is made, making a section on the convolutional neural networks, which are the most used for image treatment, and therefore are the ones used in this project. The classification method implemented is described, as well as its configuration and the results obtained with the given implementation.

2010 Mathematics Subject Classification. 68T20

Chapter 1

Motivation and planification

Capsule endoscopy [1] is a procedure that uses a tiny wireless camera to take pictures of the patient digestive tract. A capsule endoscopy camera sits inside a vitamin-size capsule the patient swallow. As the capsule travels through the digestive tract, the camera takes thousands of pictures that are transmitted to an external device.

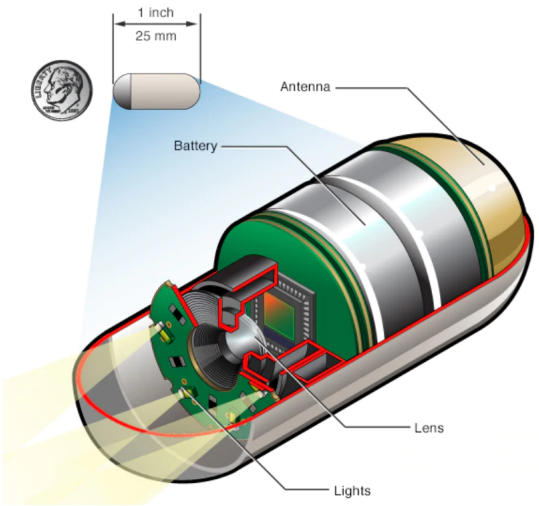

Figure 1.1: Wireless capsule endoscopy composition example

The research group on Data Science of the University of Barcelona works with a data set of images taken with a wireless capsule endoscopy.

The data set (GLF-SB2) has images previously classified into seven categories:

1. Bubbles
2. Clear blob
3. Dilated
4. Turbid
5. Undefined
6. Wall
7. Wrinkles

1.1. Objectives

The aim of this project is to create a new category (other). A supervised model is trained with a limited labeled data set. In the medical field, data sets may be limited, due to economical or legal concerns. A model trained with a certain data set is only able to classify images from the same sample. When a different image is evaluated, the prediction is random. For this reason, is interesting to be able to detect whether an image belongs to the training distribution or not. In the problem faced on this project, it is important to detect images different from the training distribution, since a "new" image could denote the appearance of a new pathology, which must be considered as crucial information.

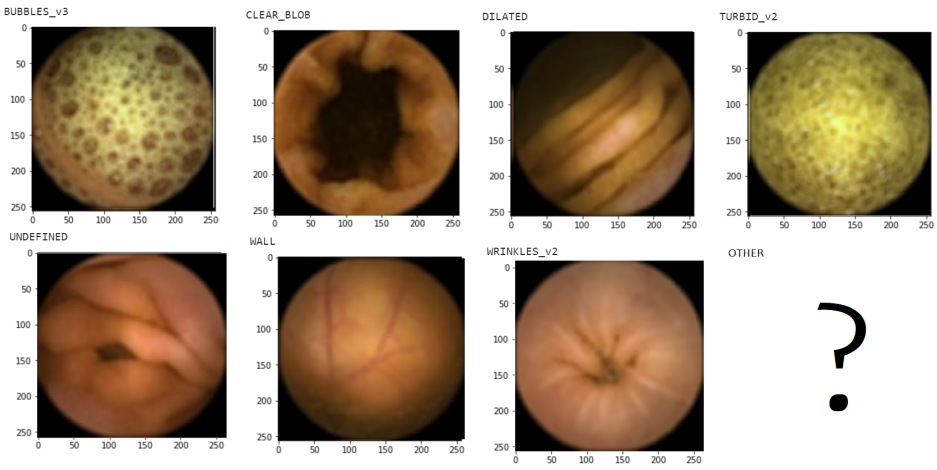

Figure 1.2: Dataset categories

1.2. Planification

The initial planning consisted of using some method of classification to achieve this objective. The first task was to study which possible classification methods or strategies would be the most appropiate for the project. The student began a process of basic training[2], in which he studied from the elementary level to a fairly high level in the field of neural networks.

After a few weeks of research, the goal shifted to detect out-of-distribution images (Initially the aim was to detect images with blood).

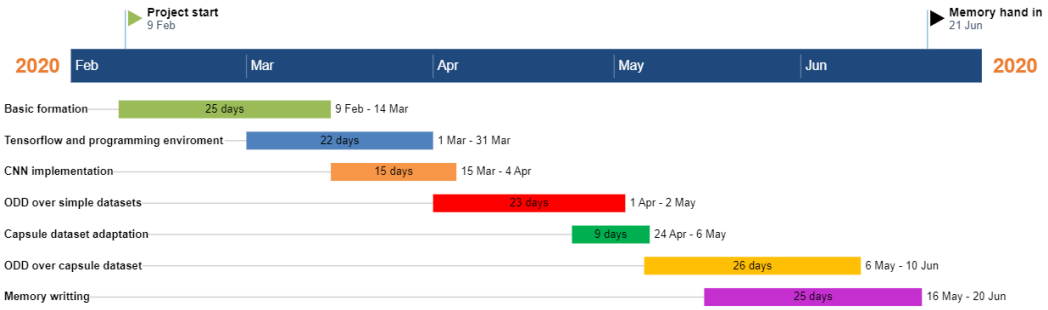

Figure 1.3: Gantt chart

The work was done first on a simple dataset (MNIST[3]),to later work with the project dataset, more complex, so the workload was also divided into these two steps.

1.3. Out of distribution

1.3.1. How does it work?

In order to explain how it works, one definition is needed:

Definition 1.1. *A data set **distribution** is data obtained from the same source. Images of the same object, obtained from different sources, are images from different distributions. For example: pictures and drawings of an object, even if they represent the same object, cannot be considered to be from the same distribution.*

In out-of-distribution detection, the objective is to define a model, train it with some data and then evaluate the model with data from the same distribution and data from different distributions, and detect whether an image belongs to the same distribution that has been used to train the model, or not.

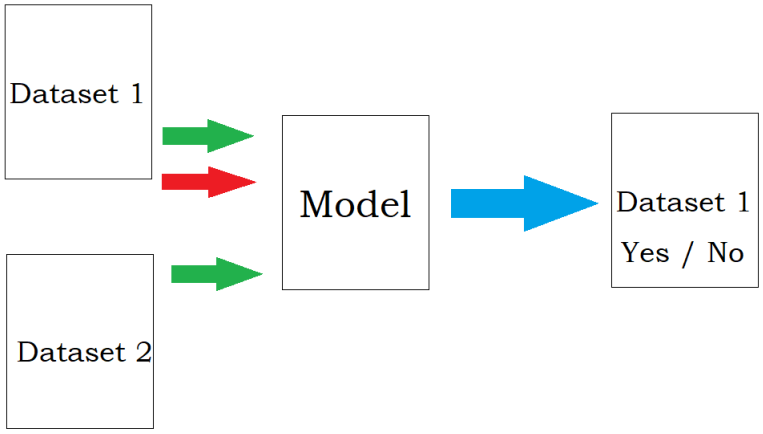

Figure 1.4: ODD detection scheme. Red arrow: training. Green arrow: evaluation. Blue arrow: output

Chapter 2

Artificial neural networks

2.1. Definitions[4]

Definition 2.1. *An **artificial neuron** is a mathematical function conceived as a model of biological neurons, a neural network. Artificial neurons are elementary units in the artificial neural network.*

Definition 2.2. *An **artificial neural network** (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is composed of artificial neurons.*

Definition 2.3. *A neural network **layer** is a collection of neurons operating together at a specific depth within a neural network. Three different types of layers can be distinguished: Input layer, which contains the raw data, the hidden layers, and the output layer, which is the simplest, usually consisting of a single output.*

Definition 2.4. *The ANN **parameters** are the coefficients of the model, which are chosen by the model itself. This means that the parameters of the model are learnable (during training the model learns which parameters minimize error).*

Definition 2.5. *ANN **hyperparameters** are elements that affect model performance, but they have to be pre-set. During **parameter tuning** scientists choose the best hyperparameters.*

2.2. Why is it called "neural" network?

It is because of the similarities it has with the biological neurons. As shown in figures 2.1 and 2.2, it is possible to define the following relationship between a biological neuron, and the mathematical model (artificial neuron):

1. Dentrites - inputs
2. Nucleus - transfer function
3. Axon - activation

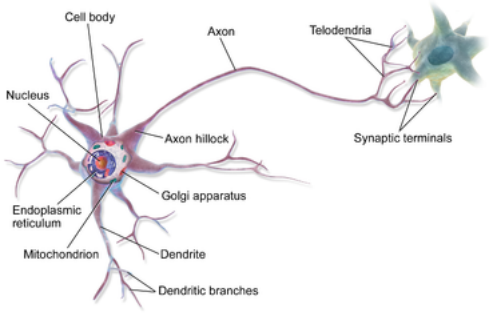

Figure 2.1: Animal neuron

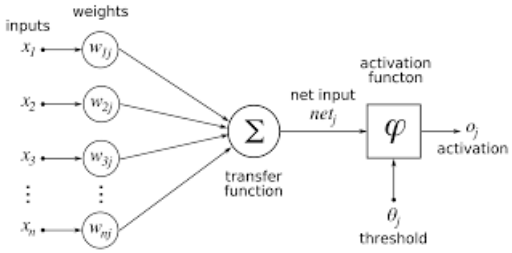

Figure 2.2: Artificial neuron

Once a neuron is described, it is possible to describe the network. Each neuron receives inputs from other neurons, i.e. the output of one neuron is the input of a related neuron and so on.

The relationships described above are of form meaning. In a more functional sense, it is possible to describe the inputs/activations as the electric signals sent and received by the biological neurons.

2.3. Mathematical model

Figure 2.2 represents an artificial neuron. This section will describe each of its components.

1. Inputs: Neuron inputs are basic pieces of information which are used to generate an output.
2. Weights: The weights are multiplied by the inputs. Their function is to decide whether each input should be stimulated or inhibited. The weights are **trainable** elements.

3. Transfer function: The transfer function collects all the products into a single input for the activation function.

4. Activation function: The activation function computes the activation of the neuron (output). There are some common activation functions in ANN.

2.3.1. Activation functions

Binary step function:

A binary step function is a threshold-based activation function. If the input value is less than a threshold value, it returns a fixed value, as well as if the input value is above the threshold. This means that the neuron can only send two different values, which makes the binary step function not useful in a wide range of problems.

$$f(x) = \begin{cases} a & \text{if } x \leq \lambda \\ b & \text{if } x > \lambda \end{cases} \tag{2.1}$$

Usually $a = 0$, $b = 1$, and threshold $\lambda = 0$

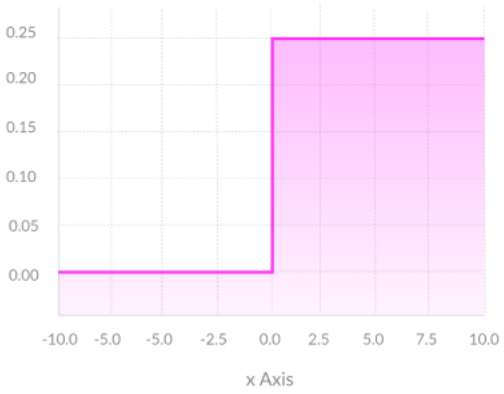

Figure 2.3: Binary step function for a = 0, b = 0.25 and λ = 0

Linear activation function:

A linear activation function takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. It is better than the binary step function in the sense that it can send different values, but this function has two major problems: It is not possible to apply gradient descent [Section 2.6.2]

since its derivative is constant, and as its name indicates, it is a linear function, so its use implies that the last layer of the network is a linear function of the first layer, causing all the layers of the network to collapse into a single layer.

$$f(x) = c \cdot x \tag{2.2}$$

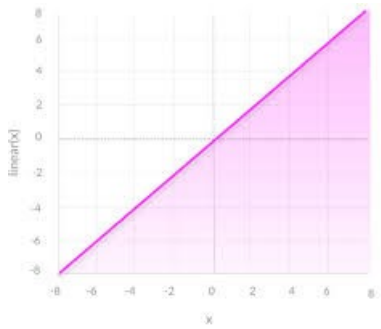

Figure 2.4: Linear activation function for c = 1

Modern models of neural networks use non-linear activation functions. They allow the model to create complex mappings between network inputs and outputs, which are essential for learning and modeling complex data.
Non-linear functions address the problems of a linear activation function:

1. They allow backpropagation and gradient descent because its derivative function is related to the inputs.
2. They allow multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex datasets with high levels of accuracy.

The following activation functions are some examples of non-linear activation functions:

Sigmoid

The output of the sigmoid function is bounded by (0,1). An important characteristic of this function is it has a smooth gradient, which makes the outputs similar for similar inputs. Another characteristic is this function makes very clear predictions, as for inputs below -2 its output is very close to 0, and for inputs above 2 its output is very close to 1.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

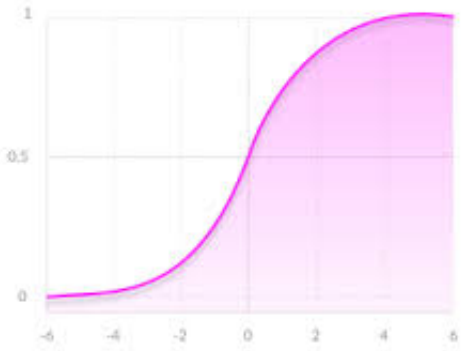

Figure 2.5: Sigmoid activation function

By the definition of derivative of the exponential function, the sigmoid derivative (equation 2.4) can be easily computed by reusing the values obtained by the computation of the validation.

$$f'(x) = \frac{e^x}{(e^x + 1)^2} = f(x)(1 - f(x)) \tag{2.4}$$

Hyperbolic tangent

Hyperbolic tangent function has some similarities with sigmoid, but it also has some important differences: This function maps its inputs into a range (-1,1) (it is zero centred) which also makes its slope to be higher.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.5}$$

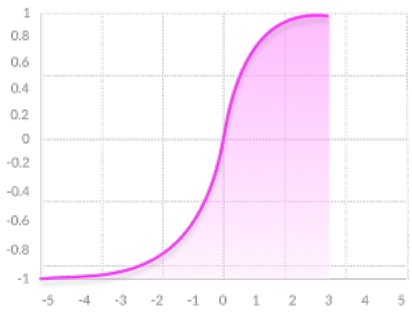

Figure 2.6: Hyperbolic tangent activation function

Its derivative can also be computed by reusing previous computations:

$$f'(x) = 1 - f(x)^2 \tag{2.6}$$

ReLU (Rectified Linear Unit)

The ReLU represents an almost linear function and therefore retains the properties of linear models that made them easy to optimize with gradient descent method.

$$f(x) = max(0, x) \tag{2.7}$$

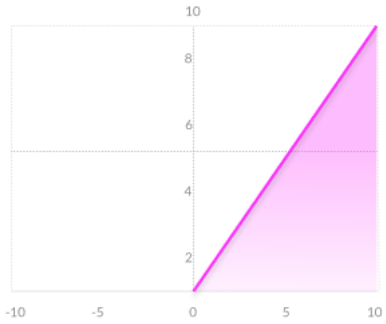

Figure 2.7: ReLu activation function

2.4. Types of ANN

Depending on which is the purpose of the ANN, there are some different types of network, more efficient for certain goals. Among some other types and specializations of them, it is possible to distinguish three main types of networks: Feedforward neural network, Recurrent neural network and Modular neural network. Each of these comes in many variants, depending on the type and number of layers, strategies or connections between layers. In feedforward neural netwotks, information moves only from the input layer directly through the hidden layers to the output layer.

A Recurrent Neural Network (RNN) is a type of neural network that contains loops, allowing information to be stored within the network. In summary, RNN use their reasoning from previous experiences to inform the upcoming events.

RNN are the most common networks used in speech and handwriting recognition. A modular neural network consists on a series of independent neural networks moderated by some intermediary. Each independent neural network serves as a module and operates with separate inputs to perform some sub-task of the task the network expects to perform.

The type of network used for image recognition and classification (and this project) are the Convolutional neural networks (CNN), which are a variation of feedforward neural networks. The name "convolutional" neural networks indicates that the network employs a mathematical operation called convolution. Convolution is a specialized type of linear operation. Convolutional networks are simply neural networks that use convolution in at least one of their layers.

2.5. CNN architecture

A convolutional neural network consists of an input layer and an output layer (with a classification function, usually a softmax classifier [Def. 2.6] , as well as multiple hidden layers. The hidden layers of a CNN typically consist of different layers such as convolutional layers, pooling layers, fully connected layers and normalization layers, followed by a final classification layer.

2.5.1. Layers

The main components of ANN are the neurons, but these components can be combined to create more complex structures. These combinations of neurons are called **layers** [Def. 2.3]. Neurons in one layer connect only to neurons in the immediately before and immediately after them. The layer that receives external data is the input layer. The layer that produces the final result is the output layer. Between them there are zero or more hidden layers.

The following are some different types of layers:

Convolutions

The objective of the Convolution Operation is to extract the high level features, such as edges or shapes, from the input image. The convolution operation is performed by sliding a size n^2 filter over the image and computing the element-wise multiplication of its elements by the images on each pixel, as shown in the figure 2.8.

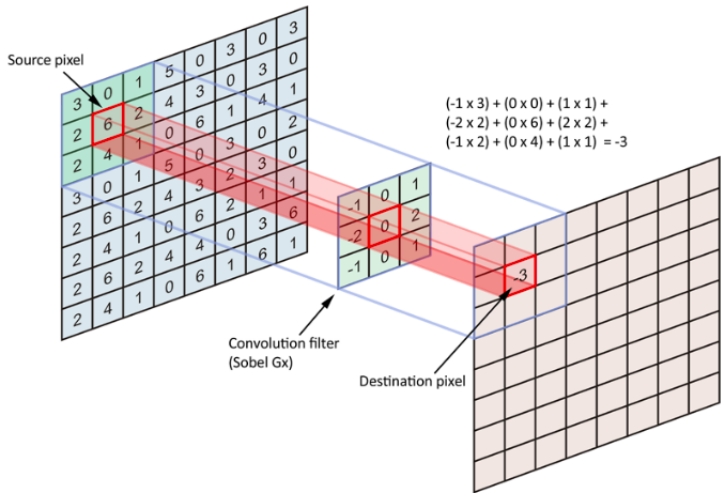

Figure 2.8: Convolution of a 3x3 filter

The following are examples of filters that can detect edges:

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

Figure 2.9: 3x3 edge detector filters

And they work as follows:

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

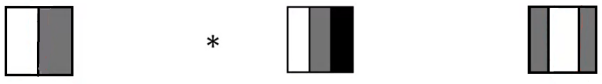

Figure 2.10: Vertical edge 3x3 filter detector

As shown on figure 2.10, on a greyscale case, where higher values indicate whitish colors and smaller values indicate darker colors, when applying the filter, the output image highlights a vertical edge, which is evident from the input image.

Convolutional layers have the following hyperparameters: Number of filters, as each layer can apply different filters to the images, filters size, stride and padding.

Stride is the number of pixels skipped in each computation. For example, figure 2.10 has a stride 1: the first central pixel is the element (starting at 0) $(1,1)$, and the second one is $(1,2)$. After completing the first row, the pixel $(0,1)$ of the result is computed by applying the filter to the input image with central pixel the element $(1,2)$. However, with a stride 2, the first pixel would also be $(1,1)$, but the second one would be $(1,3)$ and so on. One reason of using stride is to "blur" the effect of the input pixels. With a stride other than 1, the number of output pixels where an input pixels affects decreases.

When applying the convolution, the edge elements are used to compute fewer elements. For example, with a 3x3 filter, a center pixel will provide information (it will be used in the computation) of 9 pixels. In contrast, a corner element will be used in the computation of one pixel. This can be fixed using **padding**. Padding consists on adding neutral pixels around the image, so that the edge pixels become more important. Zero-padding is the most common.

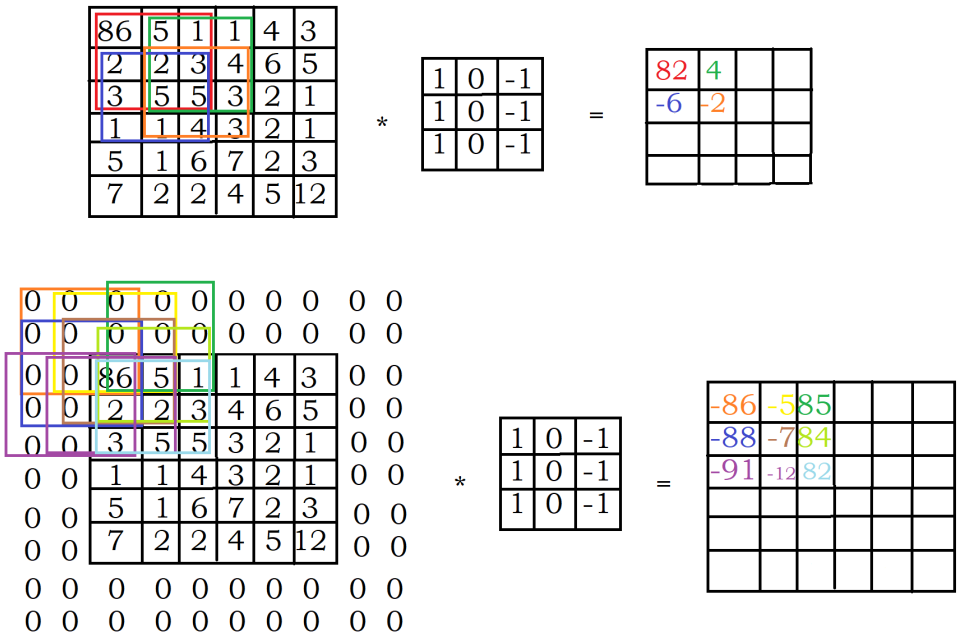

Figure 2.11: zero-padding = 2 example application, stride 1

As shown on figure 2.11, when padding is applied, the size of the output changes in relation to the size obtained without padding. This is the other reason for using padding and stride, changing the size of the output.

The parameters of the convolutional layer are the filters. Convolutional layers do not use fixed filters. During training, the network decides which filters are the best. The number of parameters for a convolutional layer is then: $n^2 \cdot f \cdot c$, where f is the number of filters and c is the number of channels ($c = 3$ in the case of RGB images).

Pooling

Pooling is a form of non-linear sample reduction. It divides the input image into a set of non-overlapping rectangles and, for each of these sub-regions, produces a single element. There are various non-linear functions for implementing pooling such as max pooling, average pooling, min pooling or stochastic pooling. The aim of pooling is to reduce the number of parameters, memory usage and amount of computations in the network, and therefore also to control **overfitting**.[1]

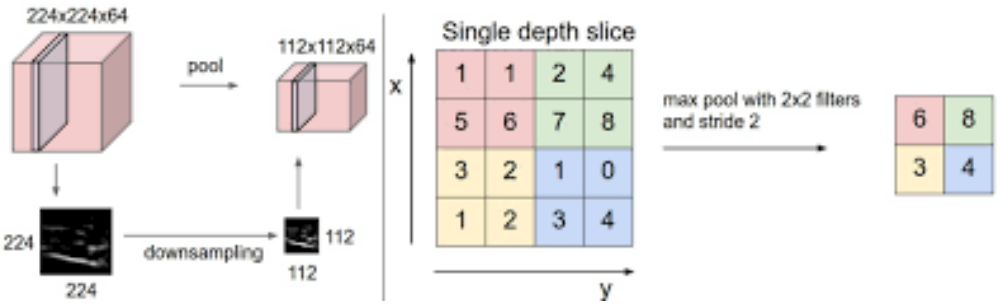

Figure 2.12: Downsampling representation and 2x2 max-pooling operation

As shown in figure 2.12, a 2x2 pooling, downsamples the input image by a factor of 2.
Pooling layers are normally applied after convolutions, in order to suppress noise and reduce the dimension.

Fully connected

The **fully connected**, or **dense layers** in CNN are used at the end of the network, to reason at a high level the neural network. Fully connected layers are

[1] overfitting appears when the model learns too well the training set, but it does not generalize, and fails to classify correctly new data, different from the training set

regular (non-convolutional) layers, made up of neurons. Neurons in a fully connected layer have connections to all activations in the previous layer. So, if an earlier layer has n neurons and the current layer has m neurons, the number of connections between the two layers is $n \cdot m$. In fully connected layers, each connection has a weight, which is a learnable parameter. When working with large images, this number of parameters increases to a point where it is unsustainable in terms of memory and computation. This is why convolutional layers are used before applying fully connected layers. After some convolutional layers and pooling layers, the network has already learned, and the number of fully connected layers (and parameters) is substantially less than in a regular non-convolutional network.

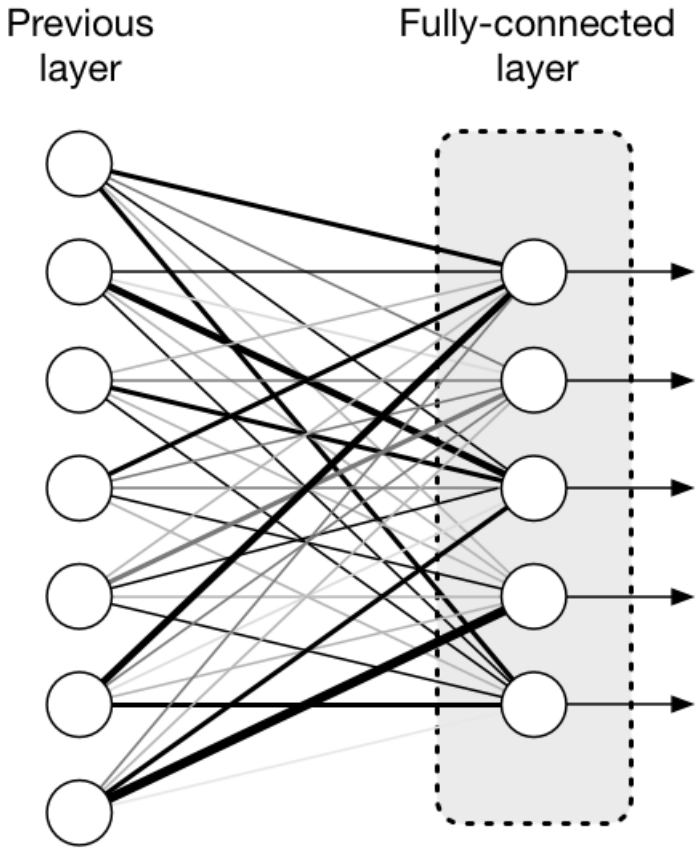

Figure 2.13: Fully connected layer example

2.6. Learning algorithm

Since the data used during this project is labeled and the goal is to find new categories, only supervised learning will be explained. The learning is supervised if the training data is classified into different categories, so that the algorithm will compare the correct output with the actual output, obtaining an error. Unsupervised learning trains with unlabeled data.

Learning is the adaptation of the network to better handle a task by considering sample observations. Learning is the process of adjusting the weights of the network to improve the accuracy of the results. The goal of the algorithm is to minimize observed errors. Learning ends when the accuracy does not improve after training. If after learning, the error rate is too high, the network typically must be redesigned.

The most commonly used algorithm in training feedforward neural networks for supervised learning is **backpropagation**.

2.6.1. Backpropagation

In this learning algorithm, first the inputs are propagated forward to get the cost of the error (or loss) and then backpropagate the errors from the output to the input, making each layer rectify its weights for a proportional part of its error. The most common algorithm used for this purpose is **gradient decent**[Section 2.6.2].

Loss function

The loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network. The loss function must fulfill two properties in order to be used by the learning algorithm:

1. It has to be possible to express it as an average over the individual loss functions for each training example:

$$C = \frac{1}{n} \sum_{x=x_0}^{x_n} C_x \tag{2.8}$$

   Since the backpropagation algorithm calculates the gradient of the loss function for each single training example. Afterwards, it has to be generalized to the general loss function.

2. It has to be expressed as a function of the outputs.

There are several types of loss functions. Four of them, are the most used in the field: Mean Squared Error (MSE), Binary Crossentropy (BCE), Categorical Crossentropy (CC) and Sparse Categorical Crossentropy (SCC).

1. Mean Squared Error: it is calculated by taking the mean of squared differences between actual and predicted values. MSE loss is used for regression tasks.

$$L = \frac{1}{2}\sum_{i=0}^{n}(x_i - y_i)^2 \tag{2.9}$$

   Where x_i is the output value and y_i is the actual value.

2. Binary Crossentropy: it is used when the classification is done over two categories. It adds the log probability of the prediction being incorrect. In binary classification, generally the target labels are 1 and 0. Since there are only two categories, the probability of each category is the complementary probability of the other one ($p_2 = 1 - p_1$).

$$L = -y_i \cdot log(x_i) - (1 - y_i) \cdot log(1 - x_i) \tag{2.10}$$

   When the predicted label is close to 1, its log is close to 0 and $log(1 - x_i)$ is close to $-\infty$. Then, if the target label is 1, the loss has to be close to 0 (since the prediction is correct), and if it is 0, the loss has to be large (incorrect prediction, big loss), so it is multiplied by y_i or $1 - y_i$ respectively. Its contrary is symmetrical.

3. Categorical Crossentropy: it is a generalization of BCE to C possible categories.

$$L = \sum_{i=1}^{C} y_i \cdot log(f(x)_i) \tag{2.11}$$

   Where C is the number of categories and f is the **softmax** [Def. 2.6] activation function.

   Definition 2.6. ***Softmax*** *activation function is a function that takes as input a vector of C real numbers, and normalizes it into a probability distribution consisting of C probabilities proportional to the exponentials of the input numbers:*

$$f(x)_i = \frac{e_i^x}{\sum_j^C e_j^s} \tag{2.12}$$

   All the resulting elements add up to 1. It is applied to the output scores.

4. Sparse Categorical Crossentropy: it is a notation modification of CC. In CC the actual categories are represented with one-hot encoded vectors, i.e. vectors of zeros except for the index of the target category. On the other hand, SCC categories are represented by indexes (integers).

   Example 2.7. Four possible categories:
   Labels: 1,2,3,4
   One-hot encoding: [1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1].

   The results must be the same for CC and SCC, with its respective encodings.

2.6.2. Gradient descent[2]

The goal of the learning algorithm is to minimize the loss function. Gradient descent consists on updating all the network learnable parameters with the aim of producing an output for the network that minimizes the error. A learning rate (α) is set, and each network parameter is updated with an α-proportion of the loss for this parameter:

$$w_{ij} = w_{ij} - \alpha \frac{\delta L}{\delta w_{ij}} \tag{2.13}$$

Where $\frac{\delta L}{\delta w_{ij}}$ is the gradient of the loss with respect to the weight w_{ij}. This gradient can be computed with the chain rule as follows:

$$\frac{\delta L}{\delta w_{ij}} = \frac{\delta L}{\delta output_j} \cdot \frac{\delta output_j}{\delta layer_j} \cdot \frac{\delta layer_j}{\delta w_{ij}} \tag{2.14}$$

Since $\frac{\delta layer_j}{\delta w_{ij}}$ is the output of the following layer, this output must be known. This is the reason backpropagation is used: in order to get one layer gradient, the output of the following layer needs to be known, so the error is backpropagated from the last layer to the first one.

Nowadays, this method is not used, since it requires the whole dataset to be forwarded and backpropagated at once. Nowadays datasets are so huge that makes this impossible. In order to fix this, Stochastic Gradient Descend and mini-batch gradient descent were designed, and consist on submitting the input one by one (Stochastic), or submitting the input by mini-batches (small groups of data).
On the last years, some gradient descent optimization algorithms[5] have been designed, in order to improve performance with respect to gradient descent, such as gradient descent with momentum, RMSprop Optimizer or Adam optimizer.

[2]Note that for all the following methods, the same than for weights is done for biases.

Momentum

Stochastic gradient descent has problems navigating areas where the surface curves much more sharply in one dimension than in another, which are common around the local optimal. In these scenarios, stochastic gradient descent oscillates through the slopes while it only advances hesitantly along the bottom towards the local optimum.

Figure 2.14

Momentum is a method that helps accelerate stochastic gradient descent in the relevant direction.

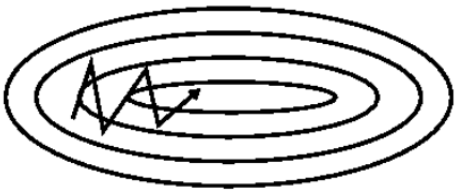

Figure 2.15

It is done by adding a fraction β of the update vector of the previous steps to the current update vector:

$$v_{dw} = \beta v_{dw} + (1 - \beta)\frac{\delta L}{\delta w_{ij}} \tag{2.15}$$

It is also known as exponentially decaying average of past gradients.
Then, the parameter update is done as so:

$$w_{ij} = w_{ij} - \alpha v_{dw} \tag{2.16}$$

β is usually set to 0.9. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.

RMSprop

RMSprop (Root Mean Square Propagation) face the same problem than momentum, but it faces it by updating the learning rate.

Learning rate is updated by dividing by the root of the squared gradient. Since the gradient is only the gradient for a single input (or a mini-batch) the moving average is used:

$$S_{dw} = \beta S_{dw} + (1-\beta)(\frac{\delta L}{\delta w_{ij}})^2 \tag{2.17}$$

Then, depending on the gradient, the leaning rate changes: when the gradient (in absolute value) is bigger (> 1) the learning rate decreases, and when the gradient is smaller (< 1) the learning rate increases, speeding up the learning.

$$w_{ij} = w_{ij} - \frac{\alpha}{\sqrt{S_{dw}}} \cdot \frac{\delta L}{\delta w_{ij}} \tag{2.18}$$

Adam

Adam (adaptive moment estimation) is a combination of momentum and RMSprop. Adam also computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like RMSprop, it also keeps an exponentially decaying average of past gradients similar to momentum:

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)(\frac{\delta L}{\delta w_{ij}}) \tag{2.19}$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)(\frac{\delta L}{\delta w_{ij}})^2 \tag{2.20}$$

In addition, those vectors are corrected depending on the number of iterations (t) done:

$$V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta_1^t} \tag{2.21}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t} \tag{2.22}$$

Then, the update is done as follows:

$$W = W - \frac{\alpha}{\sqrt{S_{dw}^{corrected}}} \cdot V_{dw}^{corrected} \tag{2.23}$$

Chapter 3

Out-of-distribution detection

3.1. Implementation environment

Due to the characteristics of this project, computational power as well as large memory capacities are required. Only the memory amount needed to store the images data set is 22GB.

Deep learning algorithms are generally executed over GPU. The main reason for this, is that unlike CPUs, which are designed for more general computing workloads, GPUs are less flexible, but they are so efficient for matrix multiplication and convolution.

Due to the GPU limitations on commercial computers, specialized platforms are usually needed for deep learning work. Google colaboratory (colab) is the environment chosen for this project. As shown on this [6] colab notebook, GPU speedup over CPU is 35x.

Tensorflow 2.2.0 with python 3.6 has been selected as development platform. TensorFlow is a widely used end-to-end open source platform for machine learning development. Keras is also used. Keras is an open-source neural-network library written in Python, which is capable of running on top of TensorFlow.

3.1.1. Colab specifications

The Google colab hardware specifications[7] are the following:

- GPU: Tesla K80
- CPU: Intel(R) Xeon(R) CPU @ 2.00GHz
- RAM: 13 GB

- Disk: 34 GB

This specifications depend on availability and change many times during a year. Google colab also has available the GPUs Tesla T4 and Tesla P100. In terms of RAM, after a crash due to limit reach, colab extends RAM up to 25GB.

3.2. Method

The aim of this method is to detect which images are in-distribution (the training distribution) and out-of-distribution (a distribution different from the training distribution).

For this project, as will be shown later, the training has been done with 4 different categories (Clear blob, undefined, wall and wrinkles). Then, with a softmax activation function, the outputs will be 4 probabilities, one for each category. The output category will then be the category with the highest probability. For example, if the softmax output is: [0.1, 0.2, 0.68, 0.02], the category will be 'wall', with a confidence probability of 68%.

This higher probability will also be considered as an indicator of whether or not it belongs to the training distribution. Initially, this does not seem to make any sense, as this probability only indicates the confidence of the prediction. In this project, this probability will be changed for a score S (no probability) which will widen the gap between in-distribution and out-of-distribution images. Finally a threshold δ will be set, and images with scores above the threshold will belong to the distribution, and images with scores below the threshold will be the out-of-distribution images.

$$OOD(S(x)) = \begin{cases} 1 & \text{if } S(x) \leq \delta \\ 0 & \text{if } S(x) > \delta \end{cases} \tag{3.1}$$

ODD (Out-of-distribution) where $S(x) = \max_i f(x)$ and $f(x)$ is the activation function.

3.2.1. Key points

There are two key points for the method implementation: adding perturbations to the input images and using a different activation function in the prediction. The implementation will then consist on defining an input processing strategy, and defining two models, one for training and one for testing (predict) with a different activation function: Softmax with temperature scaling, cosine similarity or inner-product.

3.3. Network

For this project 2 networks have been used. For the MNIST data set, a fully connected network has been used. The chosen optimizer is Adam with the default Keras parameters.

3.3.1. Fully connected network

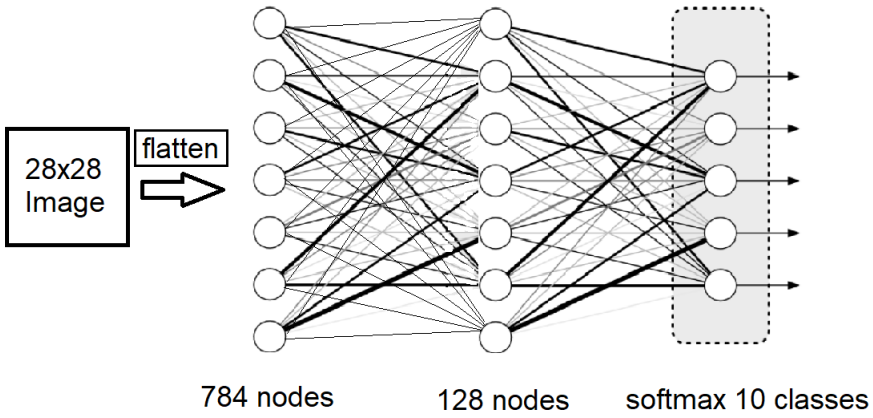

Figure 3.1: Fully connected network

For the GLF-SB2 dataset the network chosen is a Resnet50. The network used is the keras Resnet50 with an extra fully connected layer, which will be different depending on the method. For training, the layer is fully connected with softmax activation.

3.3.2. Resnet

Since the data set is more complex, a deeper network is needed. Deeper networks have to face learning problems: During back propagation, the weights are updated with a proportional part the partial derivation of the loss function. When the network gets deeper, this update can decrease to very small values, making the learning very slow, and can even stop the learning. This can intuitively be understood as data disappearing through too many layers of the network.

A 2015 paper by Microsoft researchers [8], found a solution for this problem: Residual networks.

ResNet is a deep CNN model with residual blocks. In order to understand what a residual block is, first of all the residual operation has to be explained: the residual operation, also known as identity shortcut connection, or skip connection,

consists on adding the original input to the output of the convolution block. When a residual operation is added to the network, the layers covered by this shortcut are known as a residual block.

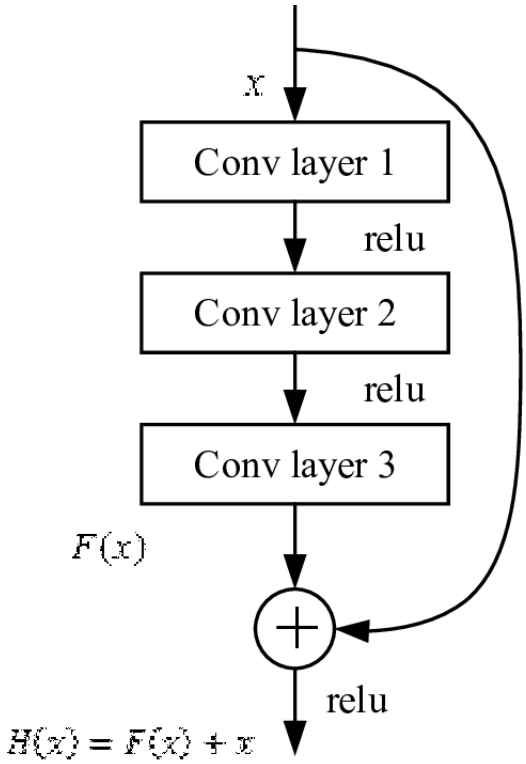

Figure 3.2: Residual block. Residual operation applied to a 3 convolution layers block

Since the input and output are matrices, the residual operation can only be done if the input and the output have the same shape. In order to ensure this, the shortcut passes through a convolution layer chosen in such a way that the output of this layer has the same dimension as the output from the convolution block.

Residual blocks mitigate the problem of vanishing gradients, and in addition they allow the model to learn an identity functions which ensure that the higher layer will perform at least as good as the lower layer, and not worse.

The model chosen for this project is ResNet50, which is a 50 layer Resnet. On the following images its complexity is shown. The chosen optimizer is Adam with the default Keras parameters, except for the learing rate, which is set at $\alpha = 0.0001$.

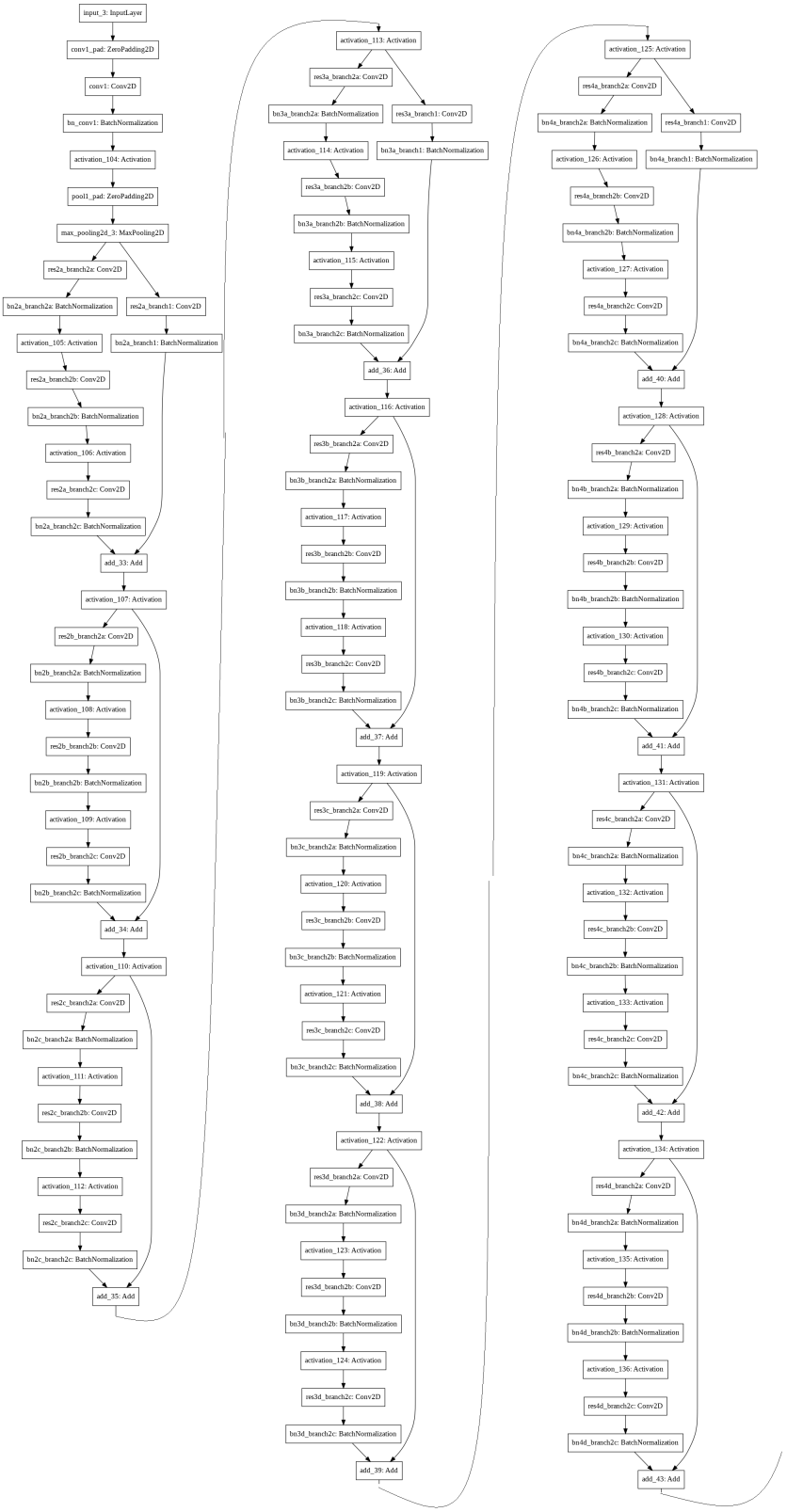

Figure 3.3: Resnet50 model architecture 1/2

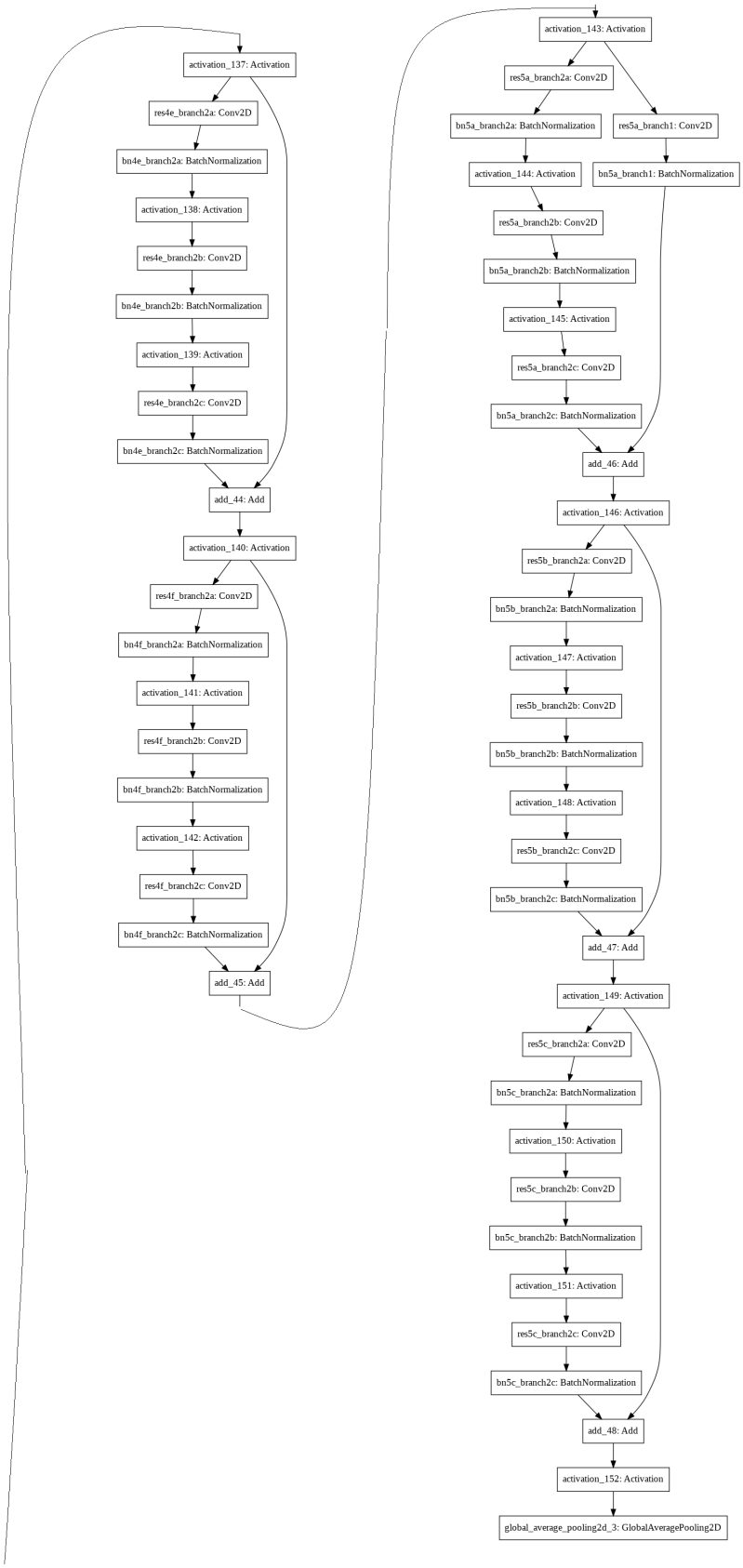

Figure 3.4: Resnet50 model architecture 2/2

Training and accuracy

The model described above has been trained during 10 epochs of 53413 iterations each. After 534k iterations, the test accuracy of the model is 94.59% for 4 classes classification (Clear blob, undefined, wall and wrinkles).

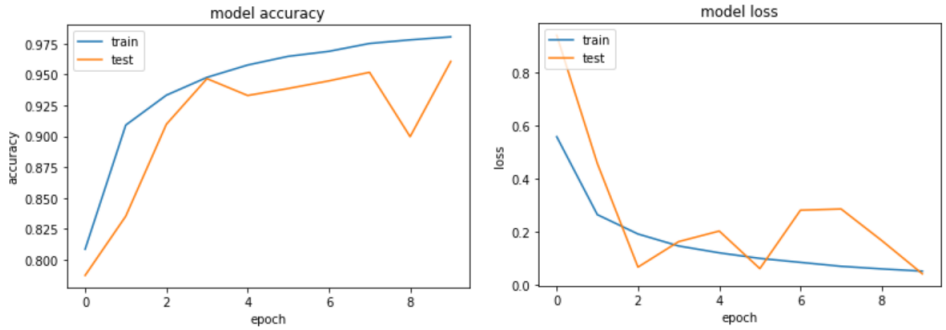

Figure 3.5: Model accuracy and loss during 10 training epochs

3.4. Training activation

During training, the activation function is a softmax classifier [Def. 2.6] for both networks, with 10 classes for the MNIST data set, and 4 classes for the GLF-SB2 data set.

3.5. Testing activation (score)

In order to classify between in-distribution and out-of-distribution, the output activation can be modified. If it is not modified, i.e. softmax is used, as will be shown later on the results analysis, the performance of the method is quite poor, roughly 25% for MNIST and 20% for GLF-SB2, for a 5% FPR[1], whereas the optimal performances are 85% and 44 % respectively.

[1]False positive ratio (FPR) is the probability of classifying as positive a negative event. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events

3.5.1. Softmax: Temperature scaling

Temperature scaling is applied in the softmax classification:

$$ODD_s(x) = Softmax(x/T) = max_i \frac{exp(f_i(x)/T)}{\sum_{j=1}^{C} exp(f_j(x)/T)} \tag{3.2}$$

Where f is the neural network and C is the number of categories.

The predictions are the same for every value of T:

Example 3.1. Classification of an image with different activations:

- Output for T=1:

  [2.3787636e-07 1.3238816e-09 1.7010921e-08 7.4349166e-11 1.0913569e-10 4.3010269e-03 1.9239084e-07 1.9822845e-02 1.0024667e-08 9.7587568e-01]

  label predicted = 9 (higher value = 9.7587568e-01)

- Output for T=1000:

  [0.09990302 0.09938575 0.09963983 0.09909996 0.09913801 0.10088714 0.09988182 0.10104141 0.09958715 0.10143589]

  label predicted = 9 (higher value = 0.10143589)

Temperature scaling: Theoretical justification

As shown in parameter tuning [Section 3.8.1] , large temperature yields better detection performance although the effect diminish when T is too large. When T is sufficiently large, the Taylor expansion [2] of the softmax score is:

[2]Taylor polynomials are approximations of a function, which becomes generally better when n increases:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{3.3}$$

where $f^{(n)}(a)$ denotes the nth derivative of f evaluated at the point a.

$$
\begin{aligned}
ODD_s(x) &= \frac{exp(f(x)/T)}{\sum_{i=1}^{C} exp(f_i(x)/T)} \\
&= \frac{1}{\frac{\sum_{i=1}^{C} exp(f_i(x)/T)}{exp(f(x)/T)}} \\
&= \frac{1}{\sum_{i=1}^{C} exp(\frac{f_i(x)-f(x)}{T})} \\
&= \frac{1}{\sum_{i=1}^{C} [1 + \frac{f_i(x)-f(x)}{T} + \frac{1}{2!}\frac{(f_i(x)-f(X))^2}{T^2} + o(\frac{1}{T^2})]} \\
&\approx \frac{1}{C - \frac{1}{T}\sum_i [f(x) - f_i(x)] + \frac{1}{2T^2}\sum_i [f(x) - f_i(x)]^2}
\end{aligned}
\tag{3.4}
$$

Where $f(x)$ is the network output for the predicted label, and $f_i(x)$ are the network outputs for each class.

Since $f(x) = f_j(x)$ for some $j \in \{0, ..., C\}$, for simplicity of the notations, let $\Delta_i = f_j(x) - f_i(x)$. Then:

$$
Mean(\Delta) = \bar{\Delta} = \frac{1}{C-1}\sum_{i \neq j} \Delta_i \tag{3.5}
$$

And:

$$
\begin{aligned}
\frac{1}{C-1}\sum_{i \neq j} \Delta_i^2 &= \frac{1}{C-1}\sum_{i \neq j} (\Delta_i - \bar{\Delta} + \bar{\Delta})^2 \\
&= \frac{1}{C-1}\sum_{i \neq j} [(\Delta_i - \bar{\Delta})^2 - 2(\Delta_i - \bar{\Delta})\bar{\Delta} + \bar{\Delta}^2] \\
&= \frac{1}{C-1}\sum_{i \neq j} [\Delta_i - \bar{\Delta}]^2 - \frac{2\bar{\Delta}}{C-1}\sum_{i \neq j} [\Delta_i - \bar{\Delta}] + \bar{\Delta}^2 \\
&= Variance^2(\Delta) + Mean^2(\Delta), \quad \text{since} \sum_{i \neq j} (\Delta_i - \Delta) = 0
\end{aligned}
\tag{3.6}
$$

Now, rewriting the Taylor approximation:

$$
\begin{aligned}
ODD_s(x) &\approx \frac{1}{C - \frac{1}{T}\sum_i [f(x) - f_i(x)] + \frac{1}{2T^2}\sum_i [f(x) - f_i(x)]^2} \\
&= \frac{1}{C - \frac{C-1}{T} Mean(\Delta) + \frac{C-1}{2T^2}[Var^2(\Delta) + Mean^2(\Delta)]}
\end{aligned}
\tag{3.7}
$$

The mean measures how much the largest output diviates from the remaining outputs, while the variance measures how much diviate the remaining outputs from each other. Due to the fact that neural networks tend to make more confident predictions on in-distribution images, the mean is higher for in-distribution images, and the variance is higher for out-of-distribution.

Example 3.2.

- If the prediction is very accurate, the outputs are p $\approx$ [1,0,0,0]. Then, the mean is: $\frac{1}{3}\sum_{i\neq 0}(1 - p_i) = \frac{1}{3}(1 + 1 + 1) = 1$, and the variance is: $\frac{1}{3}\sum_{i\neq 0}[(1 - p_i) - \bar{\Delta}] = 0$
- If the prediction is almost arbitrary, one example of output could be: p = [0.4,0.3,0.2,0.1]. Then, the mean is 0.2 and its variance is 0.2.

Their effect in the scores are positive for $Mean(\Delta)$ while are negative for $[Var^2(\Delta) + Mean^2(\Delta)]$, since they belong to the denominator of the Taylor approximation for the score. Then, the higher the value of T is, the lower $[Var^2(\Delta) + Mean^2(\Delta)]$ affects to the score.

Hence, for higher values of T, the gap between in-distribution and out-of-distribution scores increases, making the two groups more separable. This can be observed on parameter tuning [Section 3.8.1], because when adding temperature scaling with T = 10, although it is small, its effect is notable.

3.5.2. Cosine similarity

A different score function is introduced: cosine similarity.

$$
ODD_c(x) = \frac{f(x) \cdot w}{\|f(x)\| \cdot \|w\|} \tag{3.8}
$$

Where w are the weights of the last layer (trainable parameters), and f is the output of the previous layer.

This function outputs a value for each class, which also works as classifier, but it also allows to have a different ODD score, which can provide different out-of-distribution detection.

The predictions are the same for training activation than for cosine similarity activation:

Example 3.3. Classification of an image with different activations:

- Softmax activation:

  [2.3787636e-07 1.3238816e-09 1.7010921e-08 7.4349166e-11 1.0913569e-10 4.3010269e-03 1.9239084e-07 1.9822845e-02 1.0024667e-08 9.7587568e-01]

  label predicted = 9 (higher value = 9.7587568e-01)

- Cosine similarity activation:

  [-0.06442402 -0.0919809 -0.08047021 -0.11144653 -0.10545421 -0.00931601 -0.06646427 -0.00185291 -0.0816487 0.02404125]

  label predicted = 9 (higher value = 0.02404125)

3.5.3. Inner-product

Another score function is introduced: inner-product.

$$ODD_i(x) = f(x) \cdot w + b \tag{3.9}$$

Where w and b are the weights and the bias of the last layer (trainable parameters), and f is the output of the previous layer.

The predictions are the same for training activation than for inner-product activation as well:

Example 3.4. Classification of an image with different activations:

- Softmax activation:

  [2.3787636e-07 1.3238816e-09 1.7010921e-08 7.4349166e-11 1.0913569e-10 4.3010269e-03 1.9239084e-07 1.9822845e-02 1.0024667e-08 9.7587568e-01]

  label predicted = 9 (higher value = 9.7587568e-01)

- Inner-product activation:

  [-11.364547 -16.555729 -14.002443 -19.435282 -19.051462 -1.5619336 -11.576769 -0.03395233 -14.531249 3.8625479]

  label predicted = 9 (higher value = 3.8625479)

3.6. Input processing strategy: perturbations

Before feeding the image x into the neural network, the input is preprocessed by adding small perturbations to it. The preprocessed image is given by:

$$\tilde{x} = x - \epsilon sign(-\nabla_x ODD_j(x)) \tag{3.10}$$

Where ODD_j can either be softmax score, cosine similarity score or inner-product score, and ϵ being a perturbation parameter.

This has been shown to have important effect on performance for some data sets. On the 2018 paper [10] by Liang, Li & Srikant, some results are exposed on different data sets, about the impact of this input processing. On the same paper, is shown that for MNIST data set perturbations does not provide improvement. On this project this will be checked, and the performance will be tested for GLF-SB2 data set. [Section 3.8.2]

3.7. Hyperparameters

In addition to the network hyperparameters themselves, there are hyperparameters inherent to the method. Those are temperature (T), noise magnitude (epsilon) and the threshold (δ).

1. Temperature: it is the value used for temperature scaling on softmax classification. [Section 3.8.1]

2. Epsilon: it determines the amount of noise added to the input images. It has to be tuned for each network and data set. For this project it has been tuned. It will be more easily understood after having some results to compare, so the parameter tuning is explained in a further section.[Section 3.8.2]

3. Threshold: it determines which images are detected as out-of-distribution. It can change depending on the admitted error.

3.8. Parameter tuning

3.8.1. Temperature

In order to get the best value for the hyper parameter T, fixing a configuration, some different values are tested in order to find which one has a better performance.

Mnist

Configuration:

- Network: 1 fully connected layer [def.]. Accuracy 88%.
- Tested over 10k in-distribution and 10k out-of-distribution data set.
- Comparison done for a false rate positive of 5%. For each T value, a threshold that yields the 5% FPR is found.
- Epsilon = 0.

Results:

$T = 1$, correct = 2426/10000
$T = 10$, correct = 6023/10000
$T = 100$, correct = 6388/10000
$T = 1000$, correct = 6414/10000
$T = 10000$, correct = 6406/10000

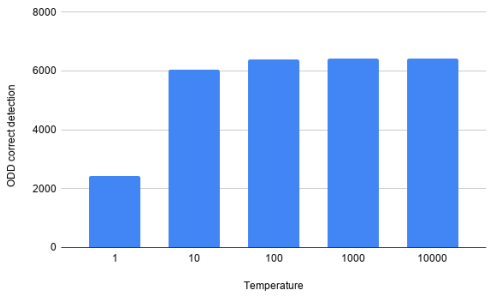

Figure 3.6: Accuracy over 10k elements for different temperature values.MNIST data set

Hence, the hyper parameter T is set at T=1000.

GLF-SB2

Configuration:

- Network: Resnet50 [def.]. Accuracy 94%.
- Tested over 3695 in-distribution images and 2928 out-of-distribution data set.
- Comparison done for a false rate positive of 5%. For each T value, a threshold that yields the 5% FPR is found (185 in-distribution images detected as out-of-distribution).
- Epsilon = 0.

Results:

- T=1, correct $589/2928 \rightarrow 20\%$
- T=10, correct $1206/2928 \rightarrow 41,1\%$
- T=100, correct $1272/2928 \rightarrow 43,4\%$
- T=1000, correct $1275/2928 \rightarrow 43,5\%$
- T=10000, correct $1275/2928 \rightarrow 43,5\%$

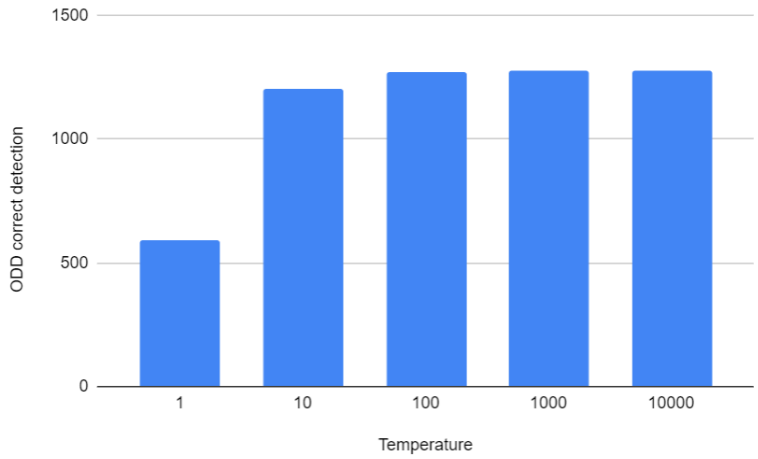

Figure 3.7: Accuracy over 2928 elements for different temperature values. GLF-SB2 data set

3.8.2. Perturbation

MNIST

As the best performance without perturbation is found with inner-product classifier, the tuning for epsilon is done with inner-product activation. The threshold is set at $\delta = 2.175$ which gives a 5% FPR for epsilon = 0.

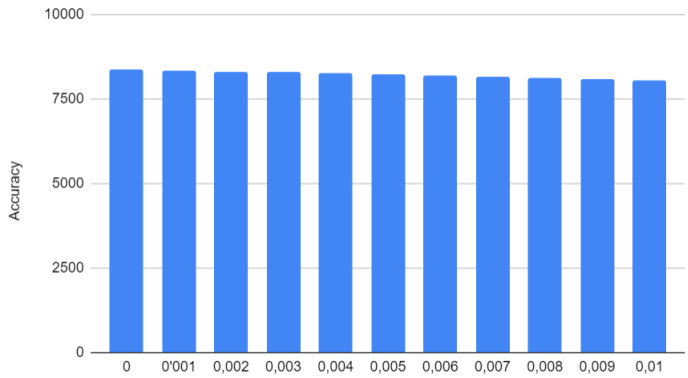

Figure 3.8: Accuracy over 10k elements for different epsilon values. MNIST data set

As found by Facebook researchers on this paper [10] (Appendix A.2), perturbations does not give better results when working with MNIST data set, so the optimal epsilon value on this project is $\epsilon = 0$ as well.

GLF-SB2

As the best performance without perturbation is found with inner-product classifier, at first try the tuning for epsilon is done with inner-product activation. The threshold is set at $\delta = 6.37$ which gives a 5% FPR for epsilon = 0.

Perturbation calculation requires high computation power and a lot of memory. Since the resources are limited, at the first attempt testing is done over a sample of 200 images. (Testing over the whole test set takes longer than 100 minutes, and without splitting the data set, the program crashes due to lack of memory).

Data sample: 35 clear blob images, 29 undefined images, 12 wall images, 42 wrinkles images, 36 bubbles images and 46 turbid images: in-dist 118, out-of-dist 82 images.

Results:

$\epsilon = 0$: 40 correct / 10 incorrect $\rightarrow$ 49%/8.5%

$\epsilon = 0.001$: 29 correct / 8 incorrect $\rightarrow$ 35.4%/6.8%
$\epsilon = 0.002$: 28 correct / 4 incorrect $\rightarrow$ 34.1%/3.4%
$\epsilon = 0.003$: 28 correct / 4 incorrect $\rightarrow$ 34.1%/3.4%
$\epsilon = 0.004$: 26 correct / 4 incorrect $\rightarrow$ 31.7%/3.4%
$\epsilon = 0.005$: 25 correct / 3 incorrect $\rightarrow$ 30.4%/2.5%

Since it seemed that could be room for improvement between 0.001 and 0.002, the following epsilon values where tried:
$\epsilon = 0.0011$: 29 correct / 7 incorrect $\rightarrow$ 35.4%/5.9%
$\epsilon = 0.0012$: 29 correct / 7 incorrect $\rightarrow$ 35.4%/5.9%
$\epsilon = 0.0013$: 29 correct / 6 incorrect $\rightarrow$ 35.4%/5.1%
$\epsilon = 0.0014$: 29 correct / 5 incorrect $\rightarrow$ 35.4%/4.2%
$\epsilon = 0.0015$: 29 correct / 5 incorrect $\rightarrow$ 35.4%/4.2%
$\epsilon = 0.0016$: 28 correct / 5 incorrect $\rightarrow$ 34.1%/4.2%

As the results where not clear, a new sample was selected: 101 in-dist and 99 out-of-dist.

Results:
$\epsilon = 0$: 46 correct / 6 incorrect $\rightarrow$ 46%/6%
$\epsilon = 0.001$: 38 correct / 3 incorrect $\rightarrow$ 38%/3%
$\epsilon = 0.002$: 37 correct / 3 incorrect $\rightarrow$ 37%/3%
$\epsilon = 0.003$: 36 correct / 2 incorrect $\rightarrow$ 36%/2%
$\epsilon = 0.004$: 36 correct / 2 incorrect $\rightarrow$ 36%/2%

Smaller values for epsilon were also tried, for a bigger sample and with different threshold, with the following results:
$\epsilon = 0$: 167 correct / 39 incorrect $\rightarrow$ 52.3%/9.3%
$\epsilon = 0.0001$: 164 correct / 39 incorrect $\rightarrow$ 51.4%/9.3%
$\epsilon = 0.0002$: 116 correct / 29 incorrect $\rightarrow$ 36.4%/6.9%
$\epsilon = 0.0003$: 116 correct / 25 incorrect $\rightarrow$ 36.4%/5.9%
$\epsilon = 0.0004$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%
$\epsilon = 0.0005$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%
$\epsilon = 0.0006$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%
$\epsilon = 0.0007$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%
$\epsilon = 0.0008$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%
$\epsilon = 0.0009$: 116 correct / 24 incorrect $\rightarrow$ 36.4%/5.7%

With these results, finally was decided to test through the whole test set for values epsilon = 0.002 and epsilon = 0.003:

$\epsilon = 0.002$, $\delta = 6.7$: 1090 correct / 167 incorrect $\rightarrow$ 37.2%/4.5%
$\epsilon = 0.003$, $\delta = 6.9$: 1131 correct / 189 incorrect $\rightarrow$ 38.6%/5.1%

The conclusion is that as happened with MNIST, perturbations does not improve the results for GLF-SB2 data set with ResNet50. Hence, for all the results epsilon is set at 0. (input preprocessing is not applied)

3.8.3. Threshold

As has been said several times during the parameter tuning, different thresholds are set depending on the desired accuracy and depending on the activation function.

Chapter 4

Results

4.1. Results analysis

Different experimental results of the application of the method are shown:

4.1.1. MNIST

With perturbation $\epsilon = 0$. Softmax with temperature T=1000. See [Section 3.8] for parameter tuning.

5% FPR

- Softmax : 6414/10000 correct at threshold $\delta = 0.100664$
- Cosine similarity : 7520/10000 correct at threshold $\delta = 0.000246$
- Inner-product : 8373/10000 correct at threshold $\delta = 2.175$

95% Correct detection

- Softmax : 6205/10000 incorrect at threshold $\delta = 0.1015111$
- Cosine similarity : 5603/10000 incorrect at threshold $\delta = 0.000825$
- Inner-product : 3139/10000 incorrect at threshold $\delta = 4.944$

4.1.2. GLF-SB2

With perturbation $\epsilon = 0$. Softmax with temperature T=1000. See [Section 3.8] for parameter tuning.

5% FPR

- Softmax : $1275/2928 \rightarrow 43.5\%$ correct at threshold $\delta = 0.2501372$
- Cosine similarity : $1146/2928 \rightarrow 39.1\%$ correct at threshold $\delta = 0.1106$
- Inner-product : $1289/2928 \rightarrow 44.0\%$ correct at threshold $\delta = 6.37$

Hence, the best method is **inner-product**.

Error evolution

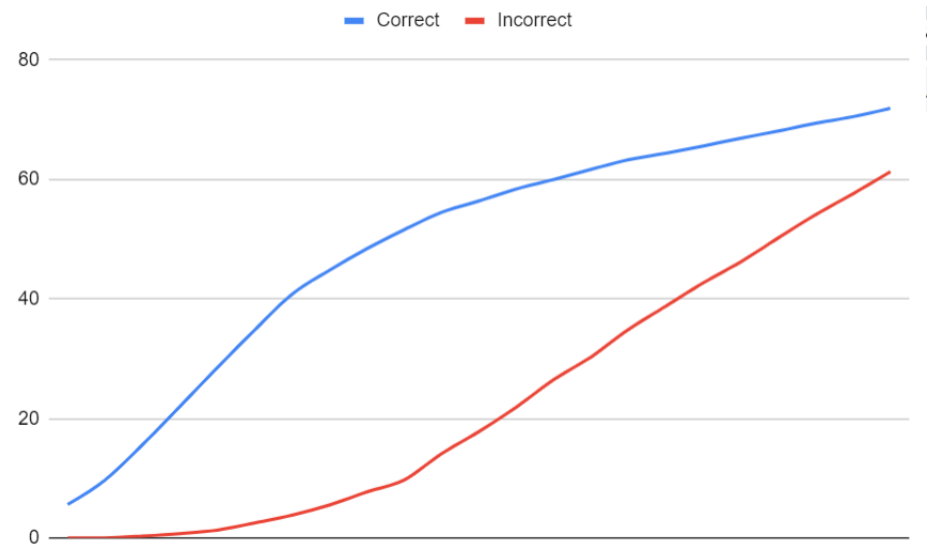

Figure 4.1

For 0% of false positive, a 10% of images are correctly detected. This give a high confidence on those correctly detected images. So in a scenario where trust is demanded, a small sample of out-of-distribution images can be detected within the GLF-SB2 data set.

For a 9.7% of false positive, more than a half of the out-of-distribution is detected (51.6%). In a scenario where human supervision is complementary to

the automatic detection, an important amount of out-of-distribution images is detected, for an acceptable amount of false positive, which can be removed by the supervisor.

From this point of around 50% of correct detection, the model shows an error exponential growth, which makes the method to be almost useless.

This is in contrast to the good performance showed on MNIST data set. In the next section, answers to this fact are searched.

Difference of performance depending on data set

T-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for visualization. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. It models each high-dimensional object by a two or three dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

In order to try to find some answers to the difference of accuracy between the two data sets tested, some points of each data set are plotted. It is not an absolute proof of all the reasons, but it may give a hint.

Plot legend:

1. Blue dots: actual out-of-distribution, detected as out-of-distribution
2. Yellow dots: actual in-distribution, detected as out-of-distribution
3. Green dots: actual in-distribution, detected as in-distribution
4. Red dots: actual out-of-distribution, detected as in-distribution

Then, the actual out-of-distribution are the blue and red dots, and the actual in-distribution are the yellow and green dots.

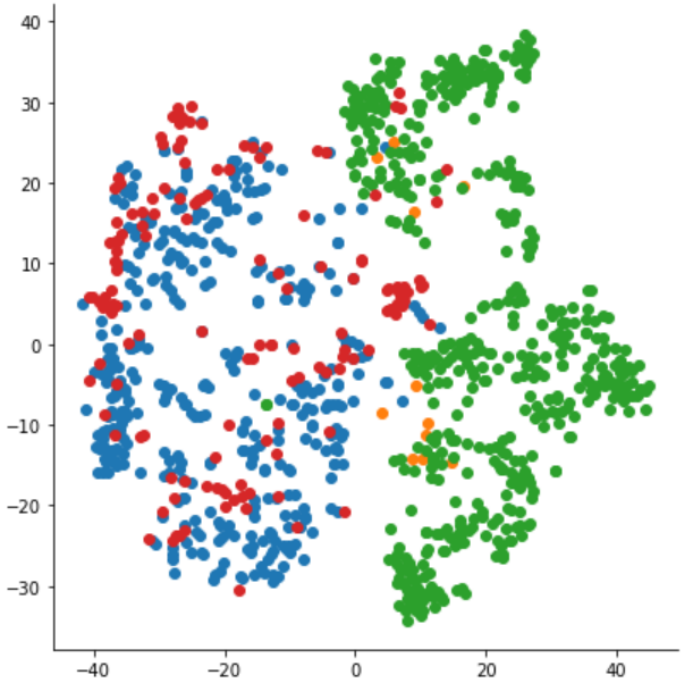

Figure 4.2: t-SNE plot for a 1000 images sample of MNIST data set

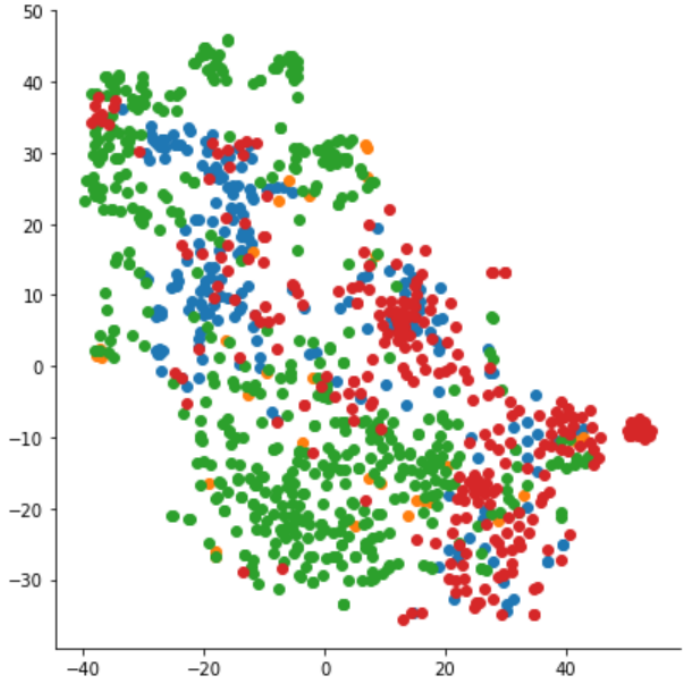

Figure 4.3: t-SNE plot for a 1000 images sample of GLF-SB2 data set

As shown on figure 4.2, MNIST data set and the out-of-distribution data set are clearly separables, since even with a straight line (figure 4.4), a pretty accurate separation could be done between both data sets.

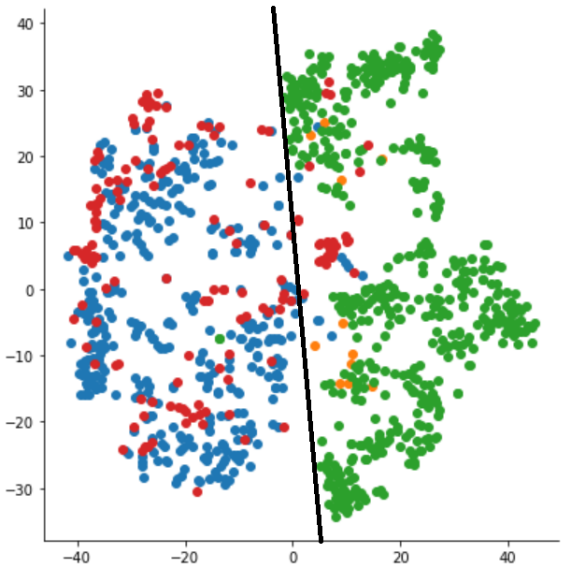

Figure 4.4: t-SNE plot for a 1000 images sample of MNIST data set, with a linear separation

On the other hand, GLF-SB2 data set is not separable in the same sense. In and out-of-distribution images are plotted interspersed along the plot, without a clear possible separation between them.

4.2. Size influence and conclusions

One important observation to do is the influence of the data set sizes. Testing has been done with similar data sizes between in and out of distribution, but it has been shown that the smaller is the out-of-distribution data set, the higher is the accuracy of out-of-distribution detection. For example with the MNIST data set, those are the results depending on the sizes: For a 5% FPR and a 10k-10k images data sets, the accuracy is of a 83.7%, while for a 10k-1k images, the accuracy grows up to 87.1%, and for a 10k-100 images, the accuracy is 89%, i.e., only 11 images are not detected.

Since different pathologies are not very common among the endoscopy images, on a real scenario, the accuracy of "other" images, when "other" means out-of-

distribution may be higher than the expected here, only by the fact of set sizes, which should be confirmed in further work.

Chapter 5

Further work

The first goal of this project was to create a new category for the GLF-SB2 data set. A method to detect out-of-distribution images has been applied to the data set. This detection method depends on the training set chosen to train the model, and the model itself. Then, the two natural next steps are:

- Training the model with the whole data set, and test with data where other images are among them, being the other images the out-of-distribution set, in order to check the real performance in a real scenario.
- Try different models. Resnet50 has been used on this project, but many other architectures are widely used on pattern recognition and neural networks in general, so a check over different architectures should be done in order to find which one fits better for the goal and with the data set.

Finally, depending on the real scenario where this method would be used, different parameters should be chosen. Two different possibilities seem the more realistic: Decreasing the FPR, i.e., having a smaller rate detection, but with high confidence, in order to give consistent data to the doctor, or detecting the most possible images. As shown on figure 4.1, this method is well-suited for the first scenario, more than for the second. Hence, in further work, after having a consistent architecture, and a good training process, a good threshold selection is a must.

In addition, once having an optimal architecture and a trained model, could be interesting testing the effects of input preprocessing [Section 3.6], since it could have positive effects with the new data sets (other images as out-of-distribution).

Bibliography

[1] Wireless Capsule Endoscopy
https://pubmed.ncbi.nlm.nih.gov/24119509/

[2] DeepLearning.ai
https://www.deeplearning.ai/

[3] MNIST dataset
https://www.tensorflow.org/datasets/catalog/mnist

[4] Machine learning google glossary
https://developers.google.com/machine-learning/glossary

[5] Gradient descent
https://ruder.io/optimizing-gradient-descent/index.html#fn4

[6] Google colab notebook. GPU performance
https://colab.research.google.com/notebooks/gpu.ipynb (run same commands in our own project)

[7] Google colab notebook. System specs
https://colab.research.google.com/drive/151805XTDg--dgHb3-AXJCpnWaqRhop_2

[8] **Deep Residual Learning for Image Recognition**. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Dec. 2015
https://arxiv.org/pdf/1512.03385.pdf

[9] **Generalized ODIN: Detecting Out-of-distribution Image without Learning from Out-of-distribution Data.** Yen-Chang Hsu, Yilin Shen, Hongxia Jin, Zsolt Kira.
Mar. 2020
https://arxiv.org/pdf/2002.11297.pdf

[10] **ENHANCING THE RELIABILITY OF OUT-OF-DISTRIBUTION IMAGE DETECTION IN NEURAL NETWORKS.** Shiyu Liang, Yixuan Li, R. Srikant
Feb. 2018
https://arxiv.org/pdf/1706.02690.pdf