POLITECNICO DI TORINO

Master's Degree in ELECTRONIC ENGENEERING

POLITECNICO
DI TORINO

Master's Degree Thesis

Capsule Networks Robustness against Adversarial Attacks and Affine Transformations

Supervisors

Prof. MAURIZIO MARTINA

Prof. MUHAMMAD SHAFIQUE

Project Ass. ALBERTO MARCHISIO

Candidate

ANTONIO DE MARCO

2019\2020

Summary

Recently, after the advent of Artificial Intelligence, Deep Neural Networks (DNNs) become popular for several machine learning-based applications, like image classification or speech recognition, inspired by brain-behavior to process information. They had a significant impact toward complex tasks, showing their superiority over human capabilities. Moreover, many recent studies revealed that slight perturbations of the test inputs can fool well-trained networks to mis-predict. Adversarial examples (AE), generated by adding small perturbations to the original inputs, imperceptible to human eyes, highlight the tendency of DNNs to learn superficial and brittle features that make the model vulnerable.
The same limitation of DNNs in image classification is noticed when the input is affected by affine transformations that do not modify the pixels but their relative position in space. These problems could lead toward drastic consequences, especially in safety-critical applications, i.e autonomous driving, robotics, smart healthcare, privacy and banking applications. Then, towards the deployment of DNNs, it is relevant to ensure robustness against inputs transformations and against malicious adversarial attacks, by developing a more robust DNN model and/or improving the defense mechanisms.
The most desired goal is to increase the generalization level of a DNN to limit these problems, similarly as the human brain works.
Many researches proposed solutions of increasing the depth of CNN architectures, others proposed to modify the hyperparameters, use data pre-processing or introduce regularization during training. For a CNN, the convolutional and the max pooling layers provide generalization and the capability to detect high order features in a large region of the image. Thanks to these layers, if a relevant feature of the object that has to be detected is slightly shifted or perturbed, the CNN is still able to detect it, but without preserving any relation with other identified features from the original image.
With the introduction of the Capsule Networks (CapsNet), the basic building block of a neural network, i.e., the neuron, has been replaced by a group of neurons, called *capsule*. Each capsule stores the feature information in a vector form, in contrast to the scalar form used by the neurons in traditional CNNs. The capsules encode

spatial information as well as the probability of an object being present: when a detected feature moves around the image, the probability of being detected does not vary, but its *pose* information changes (*equivariance property*). An efficient way of learning the coupling between capsules from different layers was also proposed, the so-called *dynamic routing* algorithm, an iterative process that replicates the behavior of max pooling, but without losing any information.
Hence, such capsule structure improves the generalization of the network, because can efficiently learn cross-correlations between different features of the inputs. Recently, other works showed that a deeper version of CapsNet can achieve high accuracy also on mid-complex datasets like the CIFAR10, despite reducing the number of parameters compared to the shallower first model.
Existing works have analyzed the vulnerabilities and robustness of CapsNets against affine transformations and adversarial attacks, respectively, showing promising results.

Research Questions and Associated Challenges

Towards this, the goal of this work is to answer the following research questions:

1. *Are CapsNets more robust than CNNs against adversarial attacks and affine transformations?*
2. *If yes, how can this phenomena be shown in a systematic way?*
3. *Which features of the CapsNets contribute more to the robustness improvement?*

Answering these questions is a challenging task. Firstly, we evaluate a good metric of comparison between CapsNets and CNNs, i.e., which network models give a fair and significant robustness comparison, which type of adversarial attacks has to be applied, etc. Then, for adversarial attacks which are based on the network model, i.e., white-box attacks, it should be interesting to analyze the transferability. *If an adversarial example has been generated to fool the network A, can it also fool the network B?* Moreover, how to analyze the relative contribution of each difference between the CapsNets and the CNNs to the difference in the robustness is still an open challenge.

In summary, our key results show that the CapsNets are more robust than deeper CNNs against affine transformation and different types of adversarial attacks. As we will demonstrate, such improvements in the robustness also hold when the adversarial examples are transferred from one network to the other one, and vice-versa.

After showing the power of the capsules, we focus our analysis on the Dynamic Routing contribution, the other main difference that characterizes the CapsNet

architectures. This algorithm, by means of its iterative voting procedure, propagates only the information that confer a high coupling with the output. The novel proposed method increases the confidence of the prediction, with a consequent improvement in terms of accuracy. By knowing that, our challenging question is: *Is the Dynamic Routing also useful for Robustness against adversarial examples and affine transformations?*
Moreover, our results show that the dynamic routing does not contribute much for improving the CapsNets robustness. Indeed, the main generalization contribution is due to the hierarchical feature learning through capsules.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently, Neural Networks (NNs) reached a large deal of interest in many different applications thanks to their ability to reproduce the human brain-behavior in the classification of complex data. By exploiting many layers of neurons and using the experience of a *training process*, the NNs learn to distinguish, hierarchically, the key-features common for a set of data, achieving even higher performances for more complex tasks, like image or speech recognition.

1.0.1 Motivation: Why evaluating the Robustness of Neural Networks?

Many recent studies revealed that small perturbations of the test inputs can fool well-trained networks to mis-predict. High confidence models are vulnerable to adversarial examples (AEs), which are generated by attacker algorithms, adding imperceptible perturbations to the original inputs. Furthermore, the power of NNs in image recognition is compromised if the test inputs suffer small transformations in space, i.e rotation translation and zooms. As a consequence of this, it is relevant to evaluate the vulnerability of NNs to *adversarial attacks* and *affine transformations* before deploying such networks in safety-critical applications, i.e., autonomous driving, privacy and banking applications and smart healthcare.

1.0.2 Scientific challenges

In order to increase the robustness of NNs many solutions have been proposed. The most desired goal is to increase the generalization level of a NN, similarly as the human brain works. As shown in Figure 1.1, many researches tried to modify the hyperparameters, use data pre-processing or introduce regularization during training. However, nowadays, improving generalization and accuracy with this kind of solutions is becoming increasingly difficult.

NNs, then, need to be revised/redesigned to fill these gaps and make them stronger against attacks and affine transformations. In the last few years new concepts have been introduced in traditional Convolutional Neural Networks (CNNs), increasing their resilience to external attacks. The proposed solution is the Capsule Network (CapsNet), a novel architecture that learns and stores more feature information (i.e orientation, position in space) in a vector form as well as the probability that a feature is present, substituting the neuron basic element with a group of neurons (*capsule*). Hence, such capsule structure improves the generalization of the networks, because it can efficiently learn cross-correlations between different features of the inputs. Furthermore, a *Dynamic Routing* between two capsule layers has been defined. It substitutes the max pooling of CNNs, propagating only the most significant features, without losing any information. Recently, other works showed that a deeper version of CapsNet can achieve high accuracy and an even better level of generalization for more complex datasets, despite reducing the number of parameters compared to the shallower first model.

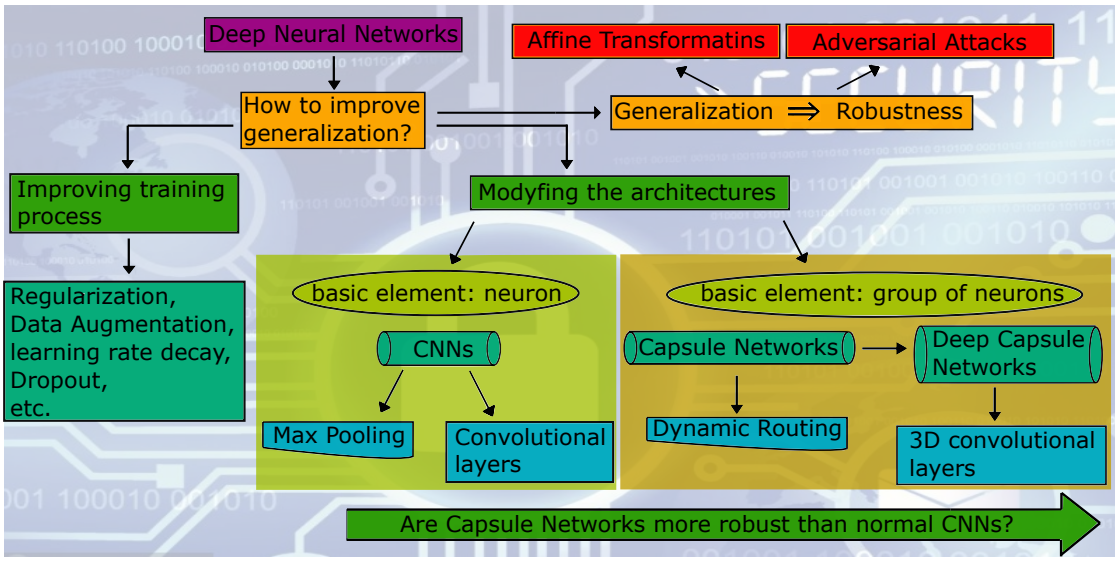

Figure 1.1: Scientific challenges.

1.0.3 Novel Contributions

Overall, promising results have been obtained by evaluating CapsNets robustness, but a systematic study which compares different types of CapsNets and CNNs is missing. Towards a comprehensive comparison, we test two CapsNets and two CNN models on the MNIST, the GTSRB and the CIFAR10, as well as on affine-transformed versions of such datasets and against adversarial attacks. With a thorough analysis (Chapter 4), we show which properties of these architectures

better contribute, increasing the robustness against attacks and their limitations.

These are the steps followed also summarized in Figure 1.2:

- We generate an affined transformed version of the CIFAR10 and GTSRB datasets, which we call **affCIFAR** and **affGTSRB** (Chapter 5.1).
- We compare the robustness against **affine transformations** for different datasets and different networks (Chapter 5.1).
- We compare the robustness against **adversarial attacks** for different datasets and different networks(Chapter 5.2).
- We evaluate the contribution of the **dynamic routing** to the CapsNets robustness (Chapter 6).

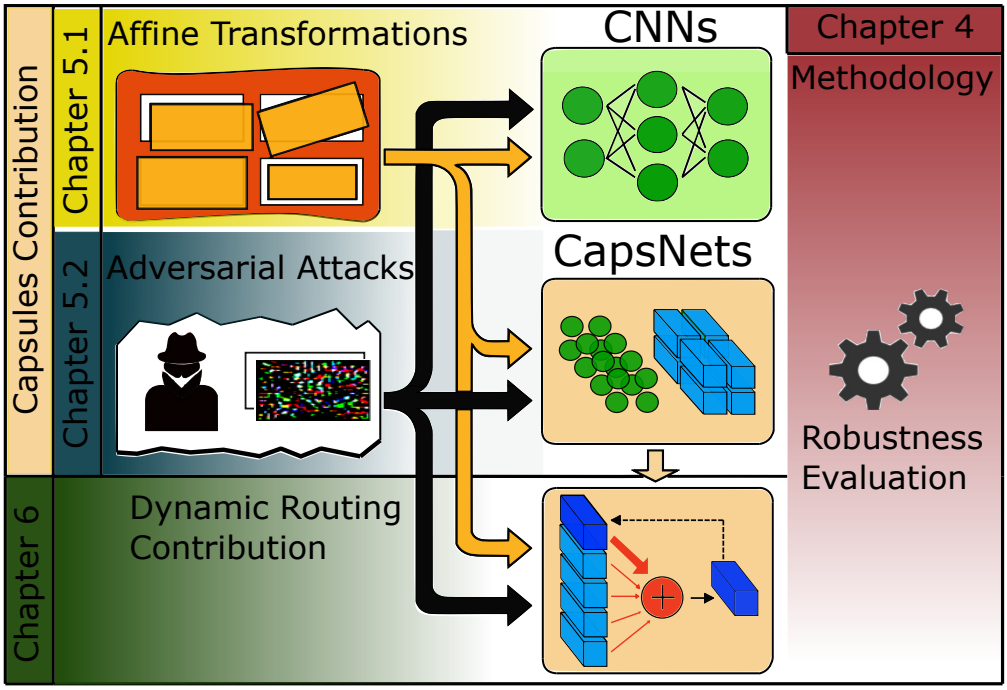

Figure 1.2: Overview of chapters organization.

Before proceeding to the technical sections, we present an overview in **Chapter 2**, to a level of detail necessary to understand the research contribution of the work.

Chapter 2

Background

2.1 From Artificial Intelligence to Deep Learning

Machine Learning (ML) is a current application of **Artificial Intelligence** (AI) inspired by the human ability to identify elements in the real world. Every human interfaces with different objects every day and can understand the environment from his experience [1]. The basic idea behind ML is similar: to solve apparently natural tasks for the human brain, i.e. recognition of complex data, using *examples* or *past experience*. Then, differently from any other AI algorithm, which use a certain code for the respective task, ML models learn without being programmed. **Deep learning** is a sub-branch of ML and covers all the architectures and the algorithms that exploit multiple layers of basic elements, called *neurons*, to classify a set of data. These architectures reproduce the brain-behavior, learning, hierarchically, the statistical relationships among the features of real objects.

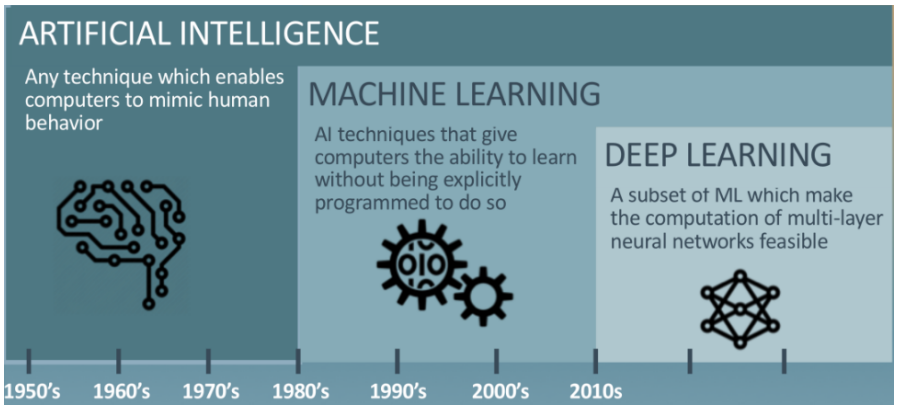

Figure 2.1: From Artificial Intelligence to Deep Learning. Source [2]

2.2 Multi-layer Perceptron architectures (MLP)

The simplest element exploited by Deep Learning is the **perceptron** (Figure 2.2b), a mathematical model of a biological neuron that stores pieces of information. While in a biological neuron (in Figure 2.2a) the *dendrites* receive electrical signals coming from other neurons, the perceptron receives probability values, processed and sent out toward the *axon*.

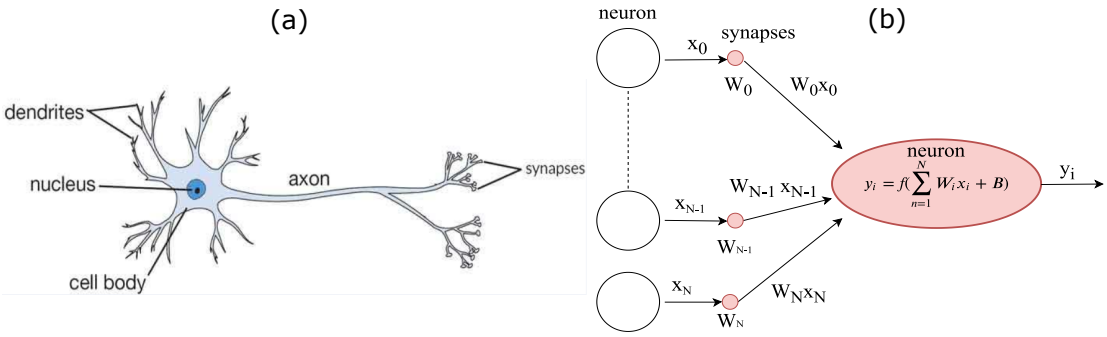

Figure 2.2: (a) Biological Neuron, (b) Perceptron model example.

The output of each element is the result of a weighted sum of the inputs, generated by other neurons or coming from the environment, compressed in a certain interval by means of an *activation function* (2.1).

$$y = f_{activation}\left(\sum_{n=1}^{\#inputs}(W_i x_i + B)\right) \tag{2.1}$$

The activation function transforms the perceptron outputs in probability values, introducing non linearity in the model. This grants generalization and a faster convergence of the learning algorithm toward the solution of the task.

The most popular activation functions are reported in Table 2.1:

Table 2.1: Examples of most common Activation Functions.

Activation Function	equation
Sigmoid or logistic	$f(x) = \frac{1}{1+e^(-x)} = \frac{e^x}{(e^x+1)}$
Tanh - Hiperbolic tangent	$f(x) = \frac{e^x - e^{-x}}{e^x + e^(-x)}$
ReLu - Rectified linear Units	$f(x) = max(0, x)$

Figure 2.3 shows an example of whole architecture composed of many perceptron layers (Multi-Layer Perceptron architecture (MLP)), in which many basic elements are connected together, one after the other, and activated according to the external stimuli, such as the human brain works.

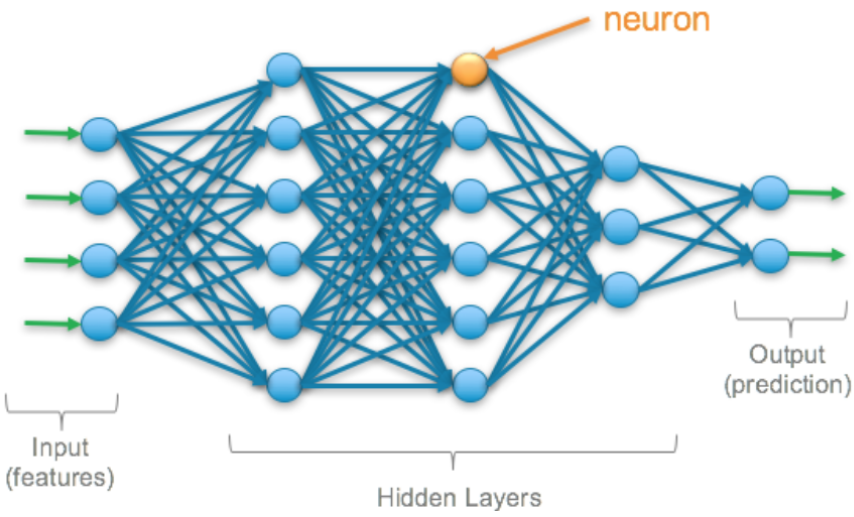

Figure 2.3: An example of Multi-Layer Perceptron (MLP). Source [3]

The *layer* is the highest-level building block, composed of many perceptrons that receive and transform the weighted inputs and pass the result as output to the next layer. The shallower neurons learn to identify low-level features (like edge or lines for image classification), that are combined together in the following hidden layers, to classify real objects in the last layer, with a certain probability. Usually, in the last layer, a different activation function is exploited, called *softmax*, that takes as input the activations and normalizes them into a probability distribution,

in order to facilitate the classification.

All the connections in Figure 2.3 perform a similar work done by the *synapses*, which, connecting *axon* and *dendrites* of different neurons, propagate the weighted information along the chain. A connection in the architecture corresponds to a parameter (or weight W_i), that has to be updated by a *training process* in line with the task that the network has to solve. Furthermore, also the biases B, present in the perceptron model, has to be updated for the same purpose, improving the convergence rate of the model.

It is possible to distinguish two main important phases in a machine learning procedure:

- The training phase: the means with which the architecture learns to perform the real task.
- The inference phase: a test of the ability of the network to perform the learned tasks.

2.3 Learning process

A machine learning model learns by itself how to solve a certain task, differently from any other computer application that before the testing has to be programmed by means of a code. The final purpose of Neural Networks (NNs) is to associate a class to each object, with a certain probability. Given an example x_i, belonging to a class y_i, the trained model M has to ensure that $y_i = M(x_i) + \epsilon$. To achieve reasonable performances and low errors, a careful learning process needs, like for the human brain with external experiences.
The learning process consists of providing to the network a collection of data, belonging to a set of classes, in agreement with the objects that would be discriminated from the external environment.
For that purpose, to make comparable the response of the networks, existing datasets are used. They correspond to a set of different data, with the same characteristics and properties, collected for a specific purpose. Each example has a corresponding true label that represents the correct class it belongs to. An example is the MNIST (Modified National Institute of Standards and Technology [4]), a simple dataset for image recognition, representing hand-written numbers. It is composed of 60,000 training images and 10,000 testing images, each one of 28 $\times$28 grey-scale pixels. All the images belong to 10 different classes, representing a different number from 0 to 9.

We have three types of learning:

- **Supervised learning**: the models use the given labeled data to learn as a comparison metric.

- **Unsupervised learning**: the network tries to recognize features without knowing the labels.
- **Semi-supervised**: not every example is labeled.

The supervised learning is common for image recognition: machines learn by iteratively minimizing a *loss function*, that expresses how much predicted labels are distant from the true ones. By means of a *back propagation* process, all the values of the weights in the chain are updated in direction of the minimum loss.

It is like to solve a logistic regression problem.

Considering the output of a single neuron with N inputs (Figure 2.2b), equal to $\sum_i (x_i w_i + b) = \vec{w}^T x + b$, the backpropagation process correspond to find an approximated model M that solves the following equation:

$$\vec{w} = argmin \sum_{i=0}^{N} (\frac{1}{N}(y_i - M(x_i, \vec{w}))^2$$

where $\sum_{i=0}^{N} (\frac{1}{N}(y_i - M(x_i, \vec{w}))^2$ is the quadratic loss function $L(\vec{w})$ (see Table 2.2).

The most common algorithm used for this task is the *Gradient Descend* that iteratively approximate the solution adjusting the values of $\vec{w}$, to bring the model toward the minimum point of the loss function. The algorithm follows these points:

Algorithm 1 Gradient Descend

1: initialize randomly all the weights $\vec{w}$ (included the bias b)
2: provide in input the dataset examples and evaluate the loss, comparing the predicted labels with the real ones
3: evalute the gradient of the loss with respect to each parameter

$$\frac{\partial L(\vec{w})}{\partial w_i} = \frac{\partial}{\partial w_i} (\sum_{i=0}^{n} \frac{1}{N} (y_i - M(x_i, \vec{w})^2)$$

4: chose a learning step ϵ, the *hyperparameter* that control how fast the algorithm moves toward the minimum of the loss
5: update all the parameters $w_i \rightarrow w_i + \epsilon \frac{\partial L(\vec{w})}{\partial w_i}$
6: check if the loss improves
7: Go to step 3 until the loss function converges towards the minimum

The number of iterations, or *epochs*, is the hyperparameter that controls how many times the full dataset passes through the training process.
Running a learning cycle for the full dataset is expensive in terms of simulation time, moreover, it requires several memory resources. It is possible to reduce the memory bandwidth and the simulation time sending to the network a fraction of

the dataset examples in parallel, according to the hardware limits. The number of parallel inputs is defined as *batch size*, another gradient descent hyperparameter representing also the number of training samples working before that the model's internal weights are updated.
Many loss functions and *optimizers* exist and also many hyperparameters have to be set for a converging training process. Table 2.2 shows three different methods that can be used to evaluate the loss function during training.

Table 2.2: Most common loss functions used for the backpropagation. N is the batch size multiplied by the dimension of inputs , C is the number of classes, y is the correct label and y' is the predicted output.

Loss Function	equation
MSE / L2 Loss / Quadratic Loss	$\frac{\sum_{i=1}^{N} (y_i - {y'}_i)^2}{N}$
(Binary) Cross Entropy	$\frac{\sum_{i=1}^{N} \sum_{j=1}^{C} {y'}_i \log (y_{i,j})}{N}$
Categorical Cross Entropy	$-\sum_{i=1}^{N} {y'}_i + \log \left(\sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \right)$

2.4 Inference and Generalization problem

The final task of machine models is to classify each real image belonging to those classes for which the network has been trained. The ability of the networks to discern examples, different from the ones used for training, is called *generalization*. For this purpose, a *test dataset* needs: it is a set of unseen images belonging to the same classes and with the same dimensions of the *train set*, used to test the ability of the network to perform the learned task. This is known as *forward process*, or *inference*, for which the inputs are propagated along with the network, applying the trained weights (Figure 2.4). The generalization of the models is tested by evaluating loss and accuracy response on the test set.

- The **accuracy** represents the parameter that indicates how many times the model correctly classifies the input.
- The **loss** expresses the distance between the true and the predicted labels.

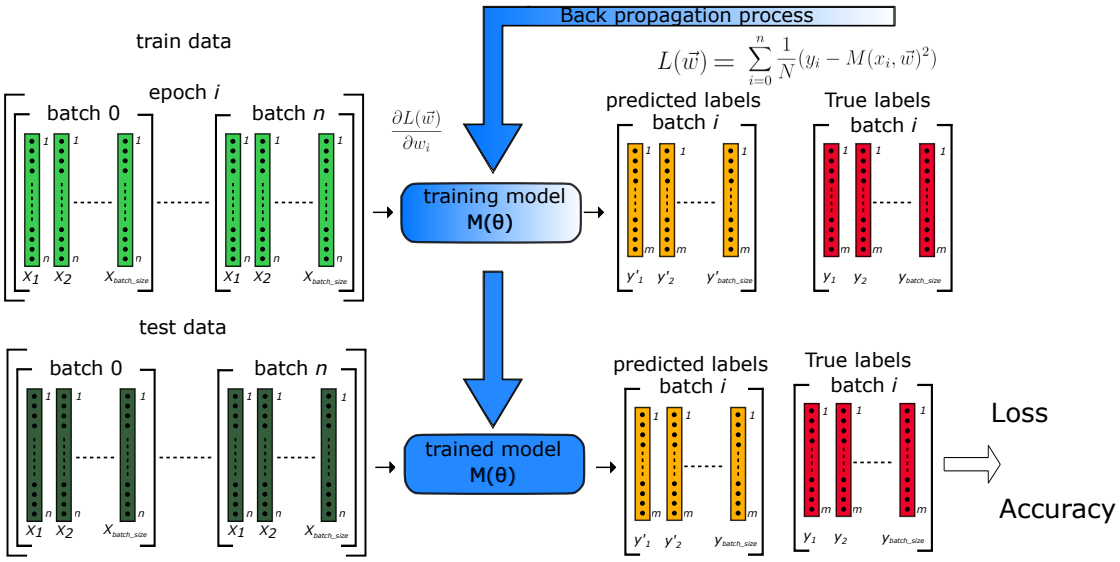

Figure 2.4: Example of inference, after training.

When the accuracy gap between test and train set is too large the problem of *overfitting* occurs, implying that the model has learned many specific features of the train dataset without generalizing.
Underfitting happens, instead, when the train set itself does not reach a sufficient accuracy. These problems occur mainly for more complex datasets, especially, if the input images used for the test are affected by noise or spatial transformations. This means that a further generalization needs in order to achieve successful accuracy and reduce the gap with the train set also in more critical conditions.
The most significant solution proposed was going deeper with Convolutional Neural Networks (CNNs), (see Section 2.5). Many other works, instead, proposed an improvement of the backpropagation process, i.e. modifying hyperparameters, applying loss regularization or using data-preprocessing, without altering the structure of the networks. It is usual to test the model after each epoch with a *validation set* of images, unseen during the training (as for the test set), to check and manage the overfitting at each iteration of the backpropagation algorithm. i.e. if the training set reaches high performances and the validation accuracy stays constant or starts to drop, the training can be stopped, because, probably, overfitting occurs.

A *regularization* (Kukacka et al. [5]) term, usually, is added to the loss function in order to avoid that the models learn so much, during the backpropagation process. Using many layers, it is possible to achieve better results, but with the increasing of the number of parameters, the network starts to learn both key-features and the noise. i.e., if a model must learn to distinguish a cat or a dog in the same

background, it is not interested in learning the background. The regularization penalizes large weights, using penalties to prevent the learning of features that are not needed. The following equation represents an example of a loss function with a l_2 regularization term, evaluated as the norm of all the weights:

$$\sum_{i=0}^{n} \frac{1}{N}(y_i - M(x_i, \vec{w})^2) + \frac{\lambda}{2m} \sum_{j=0}^{n} ||w_j||^2$$

where n is the number of layers, w_j the weighted matrix for the j^{th} layer, m the number of layers and λ a regularization hyperparameter. By adding this squared term means that the backpropagation process has to minimize the loss but also has to keep down the biggest weights.
Another solution is the *Dropout* (Srivastava et al. [6]): especially in fully connected layers, it can be useful to disconnects some neurons at training time to reduce the number of parameters to train, and keep all connections at the testing time.
Also is usual to perform random *data-augmentation* (like rotation translations and zooms) before providing each image as input for the training process, in order to make the network less vulnerable against affine transformations during inference (Mikołajczyk and Grochowski [7]).

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) replaced the first-generation MLP architectures, introducing the concept of convolution in at least one layer. In MLPs each node in each layer is connected with a certain weight to every node in the following layer, and only general matrix multiplications are performed. With the need to recognize increasingly complex data, even closer to real objects, the MLP networks become deeper and larger, with a high number of parameters to train. In addition, the problem of overfitting occurs: linking each input to every neuron, the network learns everything about the training set, but the aim is to discern only the key features common to the same class, to do classification. CNNs solve this problem by achieving a comprehensive degree of generalization reducing, also, the number of parameters required.

A convolutional layer controls sliding *kernels* (or filters) of fixed dimensions that move through all the neurons of a layer. *Sharing the same parameter across different locations, the kernels are able to recognize features on the whole image exploiting local correlations.* This grant to the architecture the *equivariance* property, in the sense that if the input image suffers a transformation, the output of each convolution does not change, i.e. if some features are translated in the image background the sliding kernel is still able to detect them.

Each convolution operation is performed between the values of the filter and the input activations coming from the previous layer, as shown in Figure 2.6. Each

step of the filter along one direction produces a new output that becomes a new activation for the following layer. Furthermore, all the convolutions performed with the same kernel generate a feature map that store information about a part-whole of the input (as shown in the upper part of Figure 2.5). In each layer, more than one filter is used to recognize many input features, which will be combined together in the following convolutional layers to learn more complex objects. The *stride* indicates how far the filter moves along one direction and determines the dimensions of the feature maps. The operation performed for each kernel is the following:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

where $S(i,j)$ represents the result of a convolution operation, I an example of input image, K the kernel and m and n the dimensions of the filter [1]. The Figure 2.5 shows a whole architecture composed of three convolutional and three fully connected layers. The architecture is able to classify correctly the input, identifying hierarchically the learned features that compose the full image, starting from the simpler, i.e. edge or curves, toward the more complex objects.
A typical CNN uses an activation function, i.e. the ReLU (Rectified Linear Unit in Table 2.1) and a *Pooling* at the end of each layer. The pooling operates over each feature map independently, by using a window that selects the maximum (*Max-Pooling*) or the mean value (*Average-Pooling*) between neighbor activations. This layer makes the representation smaller and more manageable and also increases the level of generalization providing *invariance*, in the sense that if the input image is a little bit transformed, i.e. translation of the pixels in the space is applied, the max-pooling does not change the output. *The drawback is that a lot of information is discarded.*

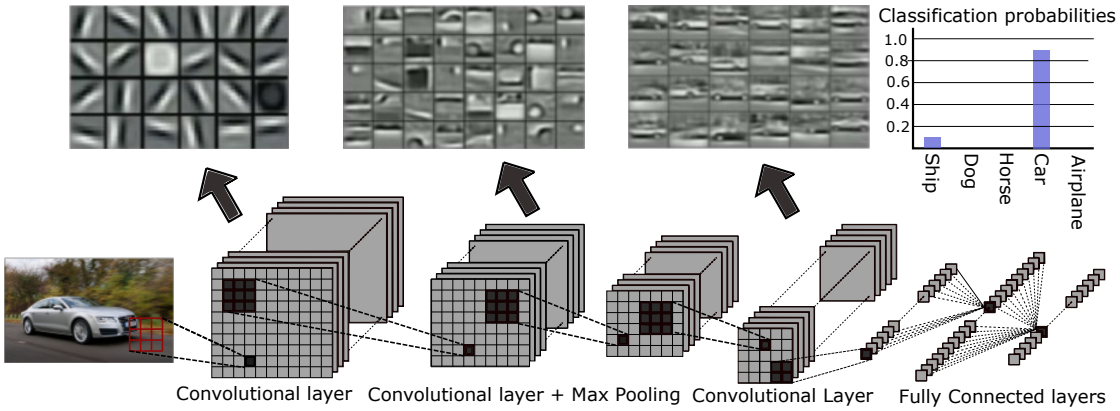

Figure 2.5: An example of Convolutional Neural Network.

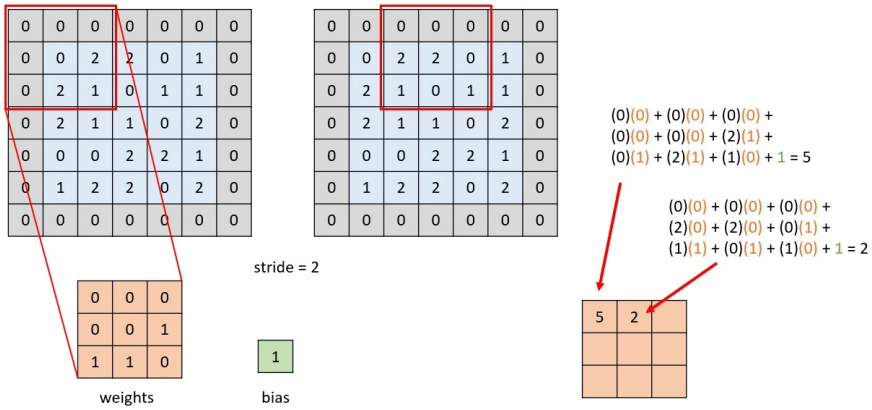

Figure 2.6: A simple illustration of a convolution operation.

2.6 Capsule Networks

Capsule Networks (CapsNets) reached a great deal of interest, thanks to their ability to achieve classification accuracies comparable to other traditional state-of-the-art classifiers, such as CNNs. The convolutional layer, as seen in Section 2.5, learns and then recognizes the key features that compose a class of real objects, but unfortunately *does not preserve any hierarchical relationship between the parts*. This means that two images, composed by the same features with different orientations in space, are not well distinguished by a normal CNN. Sabour et al. [8] proposed this new innovative architecture, based on the following main differences with respect to traditional CNNs:

- the concept of *capsules* (Hinton et al. (2011)): multidimensional entities, instead of single neurons, as basic element that constitutes each layer.
- a *Dynamic Routing* between two adjacent layers selects the capsules that have to be propagated, based on their pose agreement.
- a *squash* function that compresses the components of each capsule in a small interval at the end of each layer.

2.6.1 Capsules

A capsule is a group of neurons collecting many scalar activations in a vector form. Each vector stores different properties of the same entity, i.e the probability that a feature is present as well as for neurons in CNNs, and further information, i.e position and orientation of features in space. The orientation of the vector

(*the pose*), preserves the relationships between the learned features, the length (*activation*), instead, represents the probability that a feature is detected. Therefore, unlike CNNs, CapsNets are able to recognize the position of an object relative to another one. The key idea behind CapsNets is the *inverse rendering* process used by humans to recognize images. When our eyes recognize an object, it is decoded in small parts, and matched with the same representation in the brain, preserving spatial relations.

2.6.2 Layers of Capsules

The first architecture designed in [8] is a shallow CapsNet (ShallowCaps, also shown in Figure 3.1 in Section 3.1), composed of three layers, where the last one is a fully connected layer of capsules. Figure 2.7 shows the passage from a simple structure made of neurons toward a capsule architecture, reshaping the output coming from a convolutional layer in a vector form.

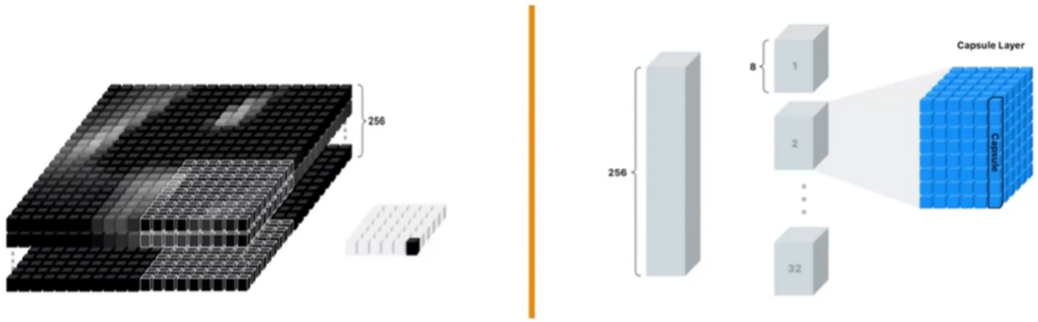

Figure 2.7: From neuron to capsule architecture. In the left side: starting from 20 × 20 × 256 features maps, a convolution with kernel 9 × 9 and stride 2 generates a 6 × 6 × 256 output. In the right side the 6 × 6 × 256 ouput is reshaped in 32 features map, each with 6 × 6 8D-capsules.

Sabour et al. [8] defines a *transformation-matrix* that, during the training, learns the relationships between each capsule i and j, of two adjacent layers, l and $l+1$ respectively (Figure 2.8). From the multiplication of the transformation-matrix W_{ij} by the input capsules u_i, $n \cdot m$ activation vectors are provided ($u_{j|i}$ in Figure 2.8), whit n and m as number of capsules in the layer l and $l+1$ respectively. Each $u_{j|i} = W_{ij}u_i$ value represents a coupling relation between each capsule of the two layers.

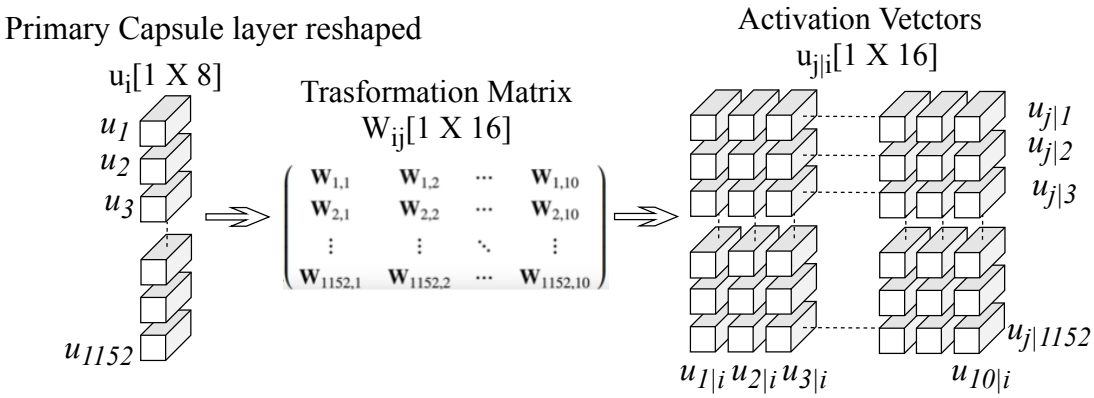

Figure 2.8: Transformation matrix and activation vectors. 1152×10 activation vectors are generated from the multiplication between the input capsules (1152 of dimension 8) and the Transformation Matrix W. Each element W_{ij} has size 1×16, where 16 is the dimension of 10 output capsules.

2.6.3 Dynamic Routing

Between two consecutive capsule layers Sabour et al. [8] also propose a new algorithm: the *Dynamic Routing*. It has the task to propagate only the activations $u_{j|i}$ with a high influence on the output capsules. Specifically, this iterative algorithm ensures that only the most voted opinion among the predictions has to be propagated.
Figure 2.9 is a representation of the algorithm and exhibits how each activation vector $u_{j|i}$ is propagated toward the output v_j by means of *coupling coefficients* c_{ij}. The initial values of the coupling coefficients c_{ij} are iteratively refined by measuring the *agreement* a_{ij} between the current output v_j and the activation vectors $u_{j|i}$ ($a_{ij} = v_j * \vec{u}_{j|i}$). The higher the agreement, the higher is the effect of increasing the coupling coefficients for that high-level capsule v_j. After a certain number of iterations, only the activation vectors with higher coefficients are propagated down. In other words, it is an improved version of max pooling, specialized for CapsNets, which preserves all the information. Each step is reported in Algorithm 2.

2.6.4 Squash Function

Each capsule in each layer is compressed in a certain interval by a squashing function (2.2), different from the ones used in normal CNNs, seen in Section 2.2.

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{2.2}$$

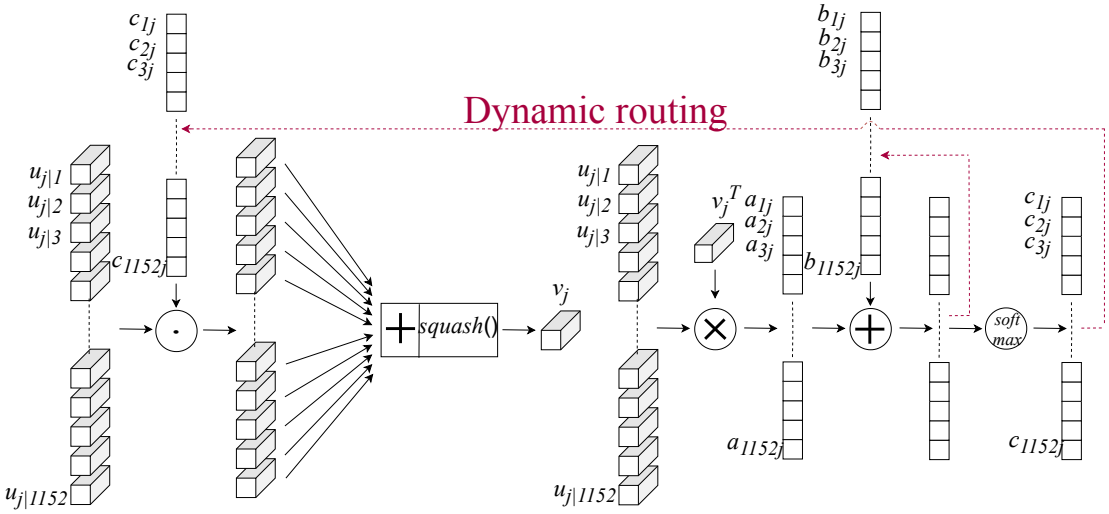

Figure 2.9: Dynamic Routing.

Algorithm 2 Dynamic Routing.

procedure ROUTING($\hat{u}_{j|i}, r, l$)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all the capsule i in layer l: $c_i \leftarrow$ softmax(b_i)
    for all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
    for all capsule j in layer $(l+1)$; $v_j \leftarrow$ squash(s_j)
    for all capsule i in layer l and j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$
  end for
  return v_j
end procedure

where v_j is the output vector of the capsule j and s_j its input. As every activation function, the task is to transform the output of linear matrix operations in a probability. The term $\frac{s_j}{||s_j||}$ scale the length of the input vector so that it becomes lower than one, without modifying its direction. The term $\frac{||s_j||^2}{1+||s_j||^2}$ is an additional scaling factor ensuring that short vectors reach a length close to 0 and long vectors a length slightly below 1. The behavior of the function is the same as Figure 2.10.

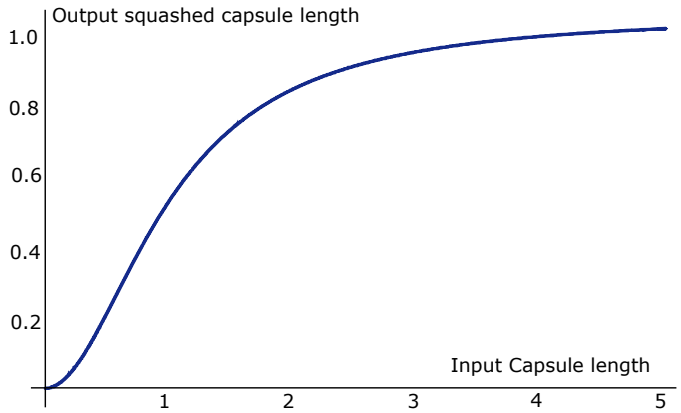

Figure 2.10: Squash function behaviour.

2.7 Adversarial Examples (AEs)

All the discussions done in the previous sections treated the problem of generalization and its solutions, starting from the modification of the training process going toward the resizing of the architectures. Moreover, the introduction of Adversarial Examples (AEs) showed that the generalization level reached by well-trained DNNs is still not sufficient to correctly classify input with small perturbations. The concept of AE was introduced by Szegedy et al. [9] like an input data, generated with a carefully constructed perturbation, with the task to mislead DNNs. Formally, having an example x that is correctly classified by a well-trained model $M(x) = y_{true}$, an adversarial example $x' = x + \eta$ is defined as a new input, perceptually identical to the original one, but wrongly classified by the model, i.e., $M(x') \neq y_{true}$.

Adversarial attacks can be distinguished based on the choice of the target class and the knowledge of the networks under attack (Papernot et al. [10]) as reported in Figure 2.12. A *white-box assumption* implies that the attacker knows the entire model, i.e., it has access to the parameters and exploits the gradient of the model to generate an AE. A *black-box assumption*, instead, does not imply any knowledge of the attacker about the network and therefore it is less powerful. If the goal of an adversarial attack is simply to mislead the network toward a wrong class ($M(x') \neq y_{true}$), the attack is considered *untargeted*, otherwise if the model M is forced to predict a targeted label ($M(x') = y_{target} \neq y_{true}$) we refer to *targeted* attack. Furthermore, a well-designed attack requires to be *imperceptible*, i.e., the crafted example is hardly distinguishable from the original input by a human eye, and *robust*, i.e., the misprediction reaches high confidence.
Figure 2.11 shows as an image belonging to the MNIST dataset, that, when perturbed by means of an attacker, fools a model $M(\theta)$ toward misclassification.

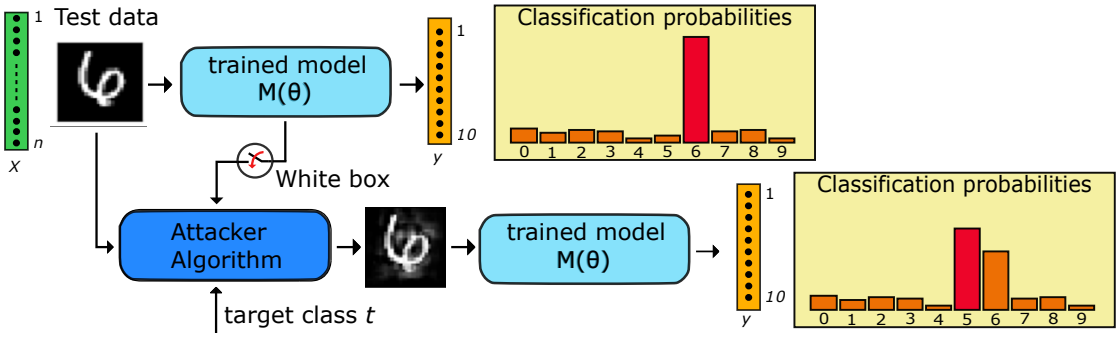

Figure 2.11: Example of adversarial image generation, that fools a trained model.

According to the attacker algorithm used, many criteria to evaluate the goodness of an attack exist. The *succes rate* is the most direct and effective criterion used that indicates how many times the AEs generated by the attacker fool the network under analysis. The *perturbation*, instead, represents the distance between the original and generated AE. As already said, the perturbation has to be balanced in order to mislead the networks and to be imperceptible by humans at the same time. Figure 2.12 summarizes the taxonomy used to distinguish the different kind of attacks.

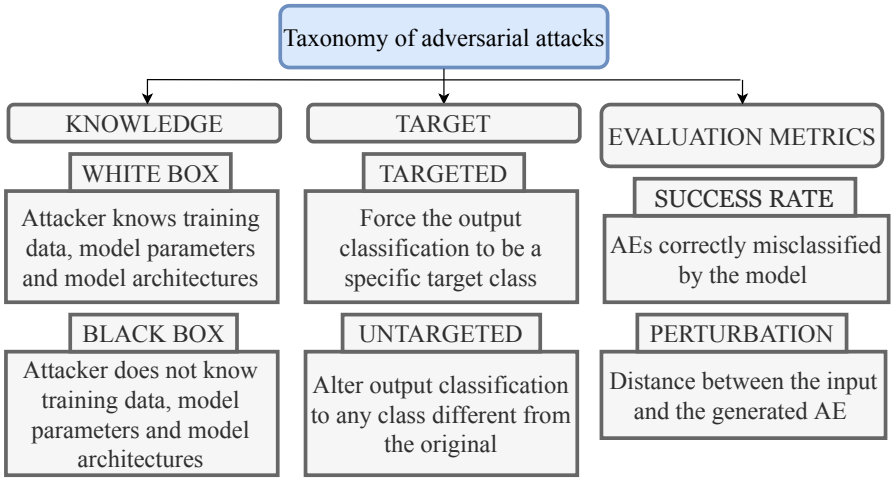

Figure 2.12: Taxonomy of Adversarial Examples.

Chapter 3

State of the art and related works

In this chapter, we analyze more in detail the state-of-the-art architectures and the algorithms exploited in this work, in order to introduce our analysis. Two capsule architectures and two CNN models with their respective training set-up have been selected, to compare them against existent attacker algorithms as will be explained in Chapter 4.

3.1 ShallowCaps Network

The first structure employed is the one already introduced in Section 2.6.1 (Sabour et al.[8]). It is the first neural model that introduces the concept of capsule instead of neuron as basic element in a shallow architecture.
The architecture designed in [8] is a shallow CapsNet (ShallowCaps), trained for the MNIST dataset, composed of three layers:

- a first standard convolutional layer with 256 9 $\times$ 9 kernels and stride 1. The shape of the output feature maps is 20 $\times$ 20 $\times$ 256. The activation function is a standard ReLu.
- a Primary Capsule layer, a convolutional layer reshaped to form 8-dimensional capsules (Figure 2.7). 256 kernels of shape 9 $\times$9, with stride 2, generating 6 $\times$ 6 $\times$ 256 feature maps, reshaped in 1552 8-dimensional capsules.
- a Digit Capsule layer (DigitCaps layer) of 10 capsules of dimension 16.
- a decoder, composed of three fully connected layers, is exploited in order to provide in output the reconstructed input images. The first layer receives in

input the 16 × 10 outputs of the DigitCaps layer, masked by means of the true labels of the input images. The two intermediate levels respectively produce 512 and 1,024 elements. The output at the third and last fully connected layer is composed of 784 elements. Reshaping the output is possible to gain the reconstructed 28 × 28 input image.

Between the two capsule layers, (i) Primary Capsule layer and (ii) DigitCaps layer, the transformation matrix multiplication and the Dynamic Routing iterations are performed, as already explained in the section 2.6, in the Figures 2.8 and 2.9.

Figure 3.1 shows the whole structure defined by Sabour et al.

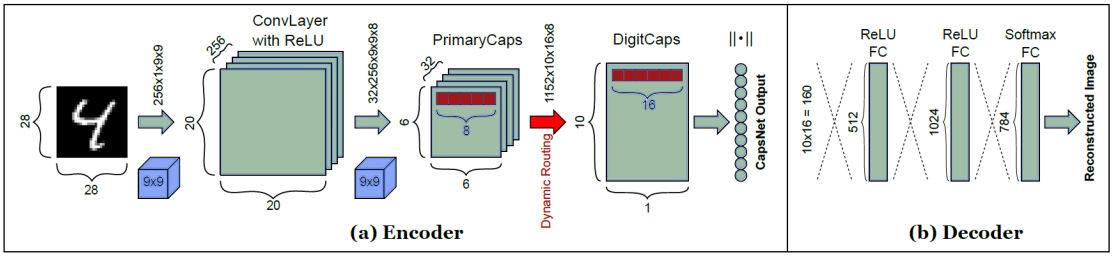

Figure 3.1: ShallowCaps original model as described in [8].

The loss function used for the backpropagation process is the result of two contributions, the *margin loss* and the *reconstruction loss*.

- the *margin loss* indicates how far the predicted labels are from the real ones and it is expressed by the following equation:

$$L_k = T_k max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k)max(0, ||v_k|| - m^-)^2 \tag{3.1}$$

  where $||v_k||$ are the output activations, each one evaluated as the norm of the vectors coming out from the last layer of capsules (DigitCaps layer). It has to be applied for each digit class k, with $T_k = 1$ for the correct label and 0 otherwise. $\lambda = 0.5$, $m^+ = 0.9$ and $m^- = 0.1$ are three hyperparameters that improve the training process.

- the *reconstruction loss* is equal to the l_2 distance between the input and the reconstructed image. $reconstruction\_loss = \frac{1}{N}\sqrt{\sum_i (x_i - x_i')^2}$

- the *total loss* is a weighted sum of the previous two equations

$$total\_loss = margin\_loss + \mu\ reconstruction\_loss \tag{3.2}$$

  The *reconstruction loss* is scaled by a factor $\mu = 0.0005$ so that it not dominates the margin loss during training. Including also the weights of the decoder in the evaluation of the gradient for the backpropagation process, works as regularization.

Figure 3.2 summarizes the training process used by Sabour et al. [8].

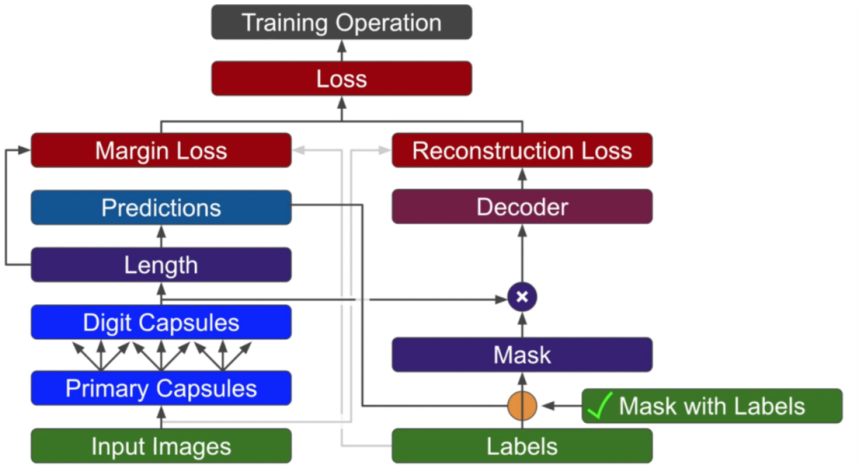

Figure 3.2: Steps followed for ShallowCaps training. [11]

Each image of the MNIST training set is sent as input to the architecture that produces 10 pose vectors of dimension 16 coming out from the DigitCaps layer. These output vectors are used both (i)to evaluate the activations (the length of the output vectors), that need for the margin loss estimation, and (ii) as inputs for the decoder block. Before entering in the decoder, the pose vectors are masked, with the true label during the training process and with the predicted label during inference.
For the training, data-augmentation is performed, shifting by up two pixels in each direction the input images.

Each digit of the masked output vector, stores a property of the input image. Some of the 16 dimensions represent a combination of global transformations, in fact as proved in [8], by adding small perturbations to the vector digits, coming out from the DigitCaps layer, the reconstructed images suffer transformations like in Figure 3.3.

The results prove that the MNIST dataset achieves 99.75% of accuracy. In addition, in order to test the ability of the capsules to preserve hierarchical relationships between the features detected, it has been tested the model on the affNIST[1] dataset. A set of images belonging to it is shown in Figure 3.4, a collection of 40 $\times$ 40 MNIST digits with random affine transfotmations i.e. rotation, translation and

[1]Available at http://www.cs.toronto.edu/tijemen/affNIST/

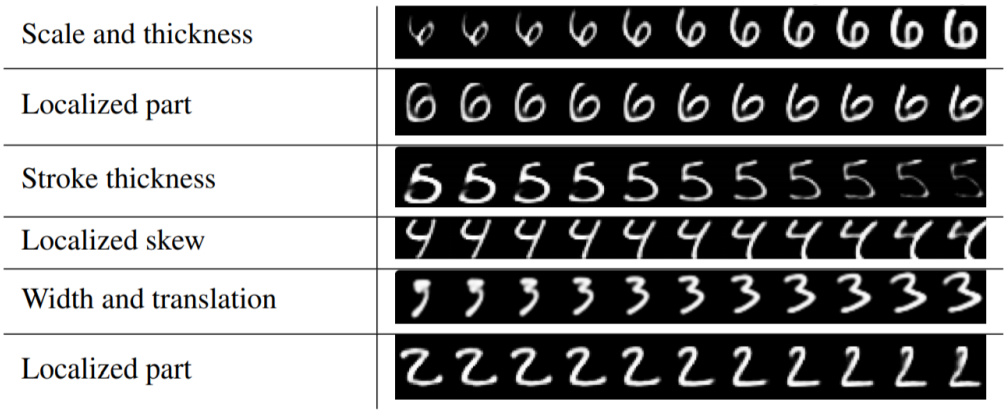

Figure 3.3: Meaning of capsules. In each row are represented the reconstructed images of the decoder, by modifying a dimension of the output predicted capsule, with steps of 0.05 in the range [-0.25, 025]. Source: [8]

zooms.

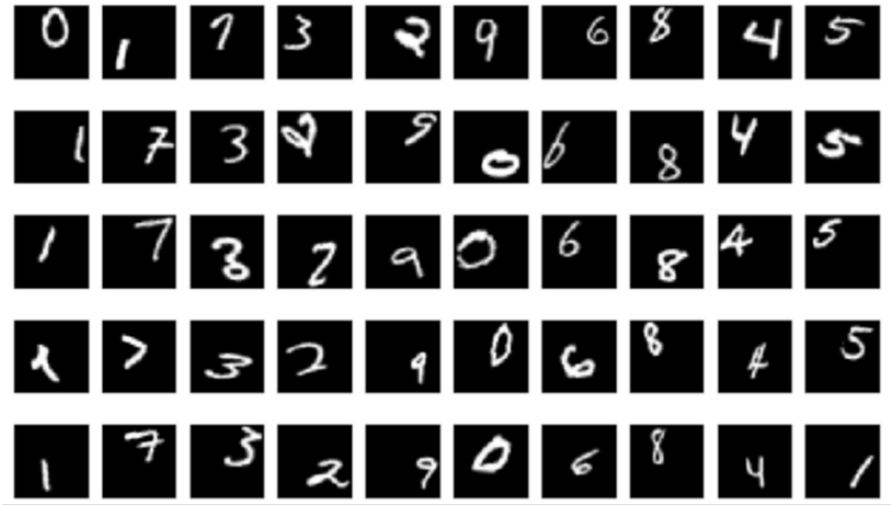

Figure 3.4: AffNIST dataset.

The state-of-the-art offered by Sabour et al. proves that the affNIST dataset trained and tested on ShallowCaps achieves 79% of accuracy, while a similar CNN model with the same number of parameters only reaches 66%. However, there is still a huge gap between the two untransformed and transformed version of the MNIST from over 99% to 79% respectively.

3.2 Going deeper with the Capsule Networks

The limit of the ShallowCaps architecture is that it is not able to correctly generalize a complex dataset. Kumar [12] proposed a three-layer architecture, like the previous one, for the GTSRB dataset (Houben et al. [13]), increasing the number of capsules coupled with the DigitCaps layer. This one needs a huge number of parameters and wasteful use of resources to reach similar performances as traditional CNN models. To solve this problem, Rajasegaran et al. [14] introduced the DeepCaps (Figure 3.5), which reduce the number of parameters, exploiting deeper capsule architectures compared to the ShallowCaps.

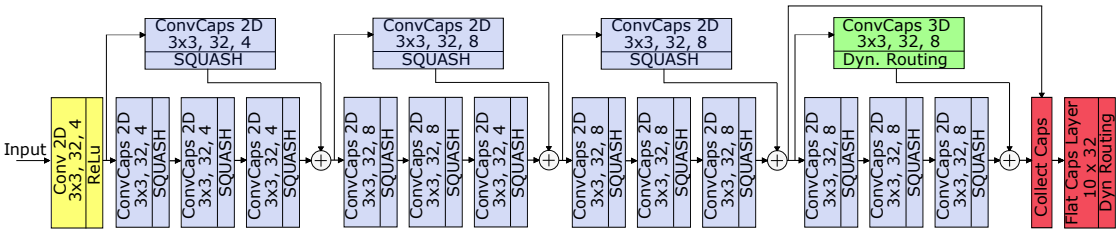

Figure 3.5: DeepCaps architecture [14].

Without stacking more than one fully connected layer of capsules, the DeepCaps introduced a new kind of 3D dynamic routing that exploits 3D convolutions, similar to the 2D version in CNNs, but with capsules. By means of a 3D convolution, the kernels are able to recognize features across different locations, preserving also the capsule structure. Figure 3.6 shows a 3D convolutional layer that uses 3D kernels, sliding through the capsules like in a normal 2D convolution with single neurons. Furthermore, at the end of the 3D convolutional layer, the DeepCaps introduces a *3D dynamic routing*.

Figure 3.6 explains how the c^l reshaped 3D capsule tensors (each one of dimensions $w^l \times w^l \times n^l$), outputs of the layer l, are transformed in c^{l+1} 3D capsule tensors (each one of dimensions $w^{l+1} \times w^{l+1} \times n^{l+1}$), outputs of the layer $l+1$.

The Algorithm 3 explains step by step how 3D dynamic Routing with 3D convolution works.

The whole architecture (Figure 3.5) is composed of 4 main groups of normal 2D convolutions, with a 3D convolutional layer in the last skip connection and a DigitCaps layer.

A further difference with respect to the ShallowCaps architecture is the implementation of the decoder. The decoder used by DeepCaps consists of deconvolutional layers that capture more spatial relationships than the fully connected ones used in ShallowCaps [8]. Furthermore, in DeepCaps, only the vector corresponding to the *true label* during the training or the one corresponding to the *predicted label* during inference is sent toward the decoder, as illustrated in Figure 3.7.

Algorithm 3 Dynamic Routing using 3D convolutions. $\phi^{\,l}$ is the output of the capsule layer l, w^l is the length and the width of the feature maps, c^l is the number of 3D capsule tensors, n^l is the dimension of capsules.

procedure ROUTING
Require: $\phi^{\,l} \in \mathbb{R}^{(w^l, w^l, c^l, n^l)}$, r and c^{l+1}, n^{l+1}
▷ Initialization
$\phi^{\,l} \leftarrow \text{Reshape}(\phi^{\,l}) \in \mathbb{R}^{(w^l, w^l, c^l \times n^l, 1)}$
$\mathbf{V} \leftarrow \text{Conv3D}(\phi^{\,l}) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, c^l, c^{l+1} \times n^{l+1})}$
$\tilde{V} \leftarrow \text{Reshape}\ (\mathbf{V}) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, n^{l+1}, c^l, c^{l+1})}$
$\mathbf{B} \leftarrow 0\ \mathbb{R}^{(w^{l+1}, w^{l+1}, c^l, c^{l+1})}$
Let $p \in w^{l+1}, q \in w^{l+1}, r \in l+1$ and $s \in c^l$
for i iterations **do**
    for all p, q, r, $k_{pqrs} \leftarrow \text{softmax\_3D}(b_{pqrs})$
    for all s, $S_{pqr} \leftarrow \sum_s K_{pqrs} * \tilde{V}_{pqrs}$
    for all s, $S_{pqr} \leftarrow \text{squash\_3D}(S_{pqr})$
    for all s, $b_{pqrs} \leftarrow b_{pqrs} + S_{pqr} \cdot \tilde{V}_{pqrs}$
end for
return $\phi^{\,l+1} = S$
end procedure

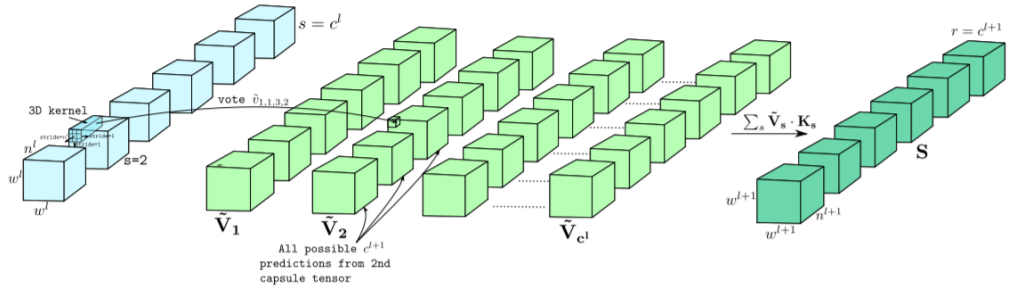

Figure 3.6: 3D Dynamic routing using 3D convolutions. Source: [14]. Each kernel of dimension $3 \times 3 \times n^l$ trasforms 9 adjacent capsules in a single vote $\tilde{v}$, and the respective stride define the dimension w^{l+1} of the layer $l+1$. $c^{l+1} \cdot n^{l+1}$ kernels generate $c^{l+1} \cdot n^{l+1}$ features maps for each c^l capsule tensor (a group of $w^l \times w^l$ capsules n^l-dimensional). To generate the output capsules, the coupling coefficients are updated according to the agreement with S and $\tilde{V}$.

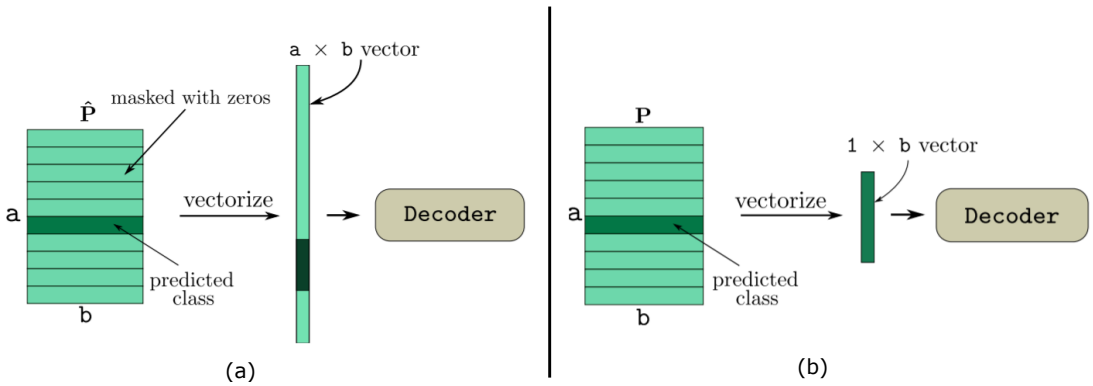

Figure 3.7: ShallowCaps decoder (a) vs DeepCaps decorer (b)

For the training process, it has been used data-augmentation with random translations, rotations, zooms and horizontal flips. Furthermore, considering the lower number of parameters required, 7.22 million against 22.48 million of the shallower architecture for CIFAR10 classification, Rajasegaran et al.[14] increases the resolution of the train images, resizing each example from $32 \times 32 \times 3$ to $64 \times 64 \times 3$.

The DeepCaps achieves higher classification accuracy results than the ShallowCaps for CIFAR10 (91.01%), reducing by 68% the number of parameters.

3.3 Dynamic Routing drawbacks and Self Routing

As seen in Section 2.6.3, the Dynamic Routing proposed by Sabour et al. [8] uses learned part-whole relationship to vote for objects output. The algorithm can be considered a linear combination of tensors that transforms input vectors u_i, in output vectors v_j, updating a set of coupling coefficients between them.

As expressed by the equations below, the transformation matrix W_{ij} transforms the inputs u_i in intermediate vectors $u_{j|i}$ and, then, each of these is weighted by the coupling coefficients c_{ij} before to be combined together.

$$v_j = \sum_i c_{ij} \hat{u}_{j|i}, \qquad \hat{u}_{j|i} = W_{ij} u_i$$

However, both this iterative algorithm and also the EM routing used by Hinton et al. [15] are computationally expensive in terms of simulation time. Many works tried to accelerate the procedure (Zhao et al. [16]) and others proposed novel routing strategies (Choi et al. [17]; Li et al.[18]).
Many others, instead, suggest to incorporate the routing procedures into the optimization process, removing it. In other words, it is possible to learn the coupling coefficients c_{ij} implicitly, including them in the weights of the transformation matrix W'_{ij}.

$$v_j = \sum_i c_{ij} W_{ij} u_i = \sum_i W'_{ij} u_i$$

In this way the weights W'_{ij} of the transformation matrix learn also the coupling between different capsule layers, without learn separately c_{ij} and W_{ij}. Our analysis (Chapter 6) proves that also against attacks and affine transformations the contribution of the Dynamic Routing is not effective, then, incorporating it in the training process could be a solution to avoid this expensive procedure.

Hahn et al.[19] proposed a different way to introduce the coupling coefficients inside the learning process, called **self routing**. Considering that each capsule i in the layer l has a pose vector u_i and an activation scalar a_i, the algorithm defines two learnable weight matrices W_{ij}^{pose} and W_i^{route} to evaluate a_j and u_j of the layer $l+1$.

- The coupling coefficients c_{ij} are the result of the multiplication between W_i^{route} and the input pose vectors u_i. By multiplying the activations a_i, coming from the layer l, and the learned coupling coefficient c_{ij}, the weighted intermediate votes a_{ij} are provided. Then, summing the i coupling contributions of the weighted votes, it is possible to obtain the activations a_j.

$$c_{ij} = softmax(W_i^{route} u_i)_j \qquad a_j = \frac{\sum_i c_{ij} a_i}{\sum_i a_i}$$

- W_{ij}^{pose} is used to generate the activation matrix $\hat{u_{j|i}}$ (it performs the same work done by transformation matrix in [8]). The output pose vector v_j is the result of an average weighted sum between the activation vector $\hat{u_{j|i}}$, the coupling coefficients c_{ij} and the activations a_i.

$$u_{j|i} = W_{ij}^{pose} u_i, \qquad v_j = \frac{\sum_i c_{ij} a_i \hat{u}_{j|i}}{\sum_i c_{ij} a_i}$$

3.4 Residual Neural Networks and ResNet20

After showing the capsule architectures, this section introduces the Residual Neural Networks and in particular the ResNet20, the one exploited for our comparative analysis. In recent years, state-of-the-art architectures became even deeper, going from few layers (e.g. AlexNet [20]) toward more than hundred layers (Simonyan and Zisserman [21]). Increasing the depth allows the networks to solve more complex tasks and, as already explained in Section 2.4, overfitting may be avoided with many techniques, i.e. regularization or dropout.
However, often, the accuracy of very deep networks gets quickly saturated and then degrades rapidly. The problem is not caused by overfitting, but by the *vanishing gradient*, due to the huge number of multiplications that the backpropagation solves at each step, from the final layer back to the first one, making the gradient exponentially close to zero. As consequence of that, the weights never update their values and therefore, no learning is performed.

Comparing two trained CNNs with different depths, one of 20 and the other of 56 layers, He et al. [22] showed that adding more layers to a proper deep model leads to higher training error. Using deeper networks, then, could degrade the performance of the model and He et al. [22] tried to solve the problem using Deep Residual Neural Networks.

The functions 'squashing'/'activation' are the ones that introduce small derivatives, so by removing those from the chain itself, it is possible to mitigate the vanishing gradient problem considerably. Figure 3.8 represents the basic block of a Residual Neural network (ResNet), that uses *skip connections*, or *shortcuts* to jump over some layers. An operation here refers to a convolution, a batch normalization and a ReLU activation, except the last operation of a block, that does not have the ReLU.

The skip connections between layers add the outputs from previous layers to the outputs of stacked layers. By staking more than one layer is possible to train deeper architectures than normal CNNs achieving convergence. The ResNet20 model, the one used for our analysis is composed of 20 layers, organized like in Figure 3.8, with a final average pooling before the last dense layer. The state-of-the-art from the work [22] indicates an accuracy value, for the CIFAR10 dataset, equal to 91.25%.

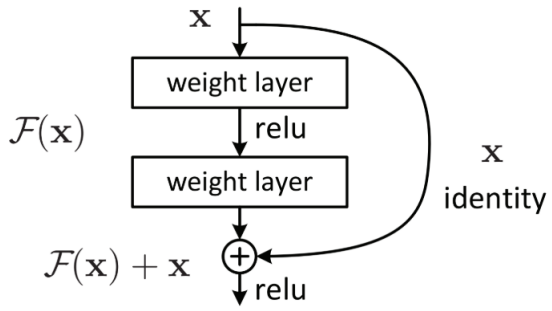

Figure 3.8: Residual Block. Source [22].

3.5 Adversarial Examples State of the Art

For our purpose, we chose, by means of a careful analysis, some attacker algorithms, to test these networks on robustness and compare their response. For a question of completeness, existing algorithms are here mentioned, focusing on the ones useful to our goal. Several algorithms for the generation of adversarial attacks have been proposed and explained (Goodfellow et al. [23]) and the corresponding defenses (Carlini and Wagner [24]) have been proferred. Goodfellow et al. [23] proposed the fast gradient sign method (FGSM), a white-box attack able to generate AEs by exploiting the gradient of the model with respect to the input image. Given an original image X, the problem can be solved with:

$$X^{adv} = X + \epsilon sign(\nabla_X J(X, y_{true}, \theta)) \tag{3.3}$$

The algorithm finds the perturbation that maximizes the loss J of a given model θ, while keeping the size of the perturbation equal to ϵ.

Figure 3.9 shows a generated AE with the FGSM method.

Projected Gradient Descend

Afterward, Madry et al. [25] and Kurakin et al. [26] proposed two different versions of the projected gradient descent (PGD) attack.

This is an iterative version of FGSM that introduce a perturbation α at every step in the direction of the maximun loss, by evaluating the gradient sign with respect to the input image. If the perturbation is higher than a certain value, it is projected inside a ball with a radius ϵ.

(Madry et al.[25]) PGD attack consists of the following iteration:

$$x'_i = x'_{i-1} - proj_\varepsilon(\alpha \cdot sign(\nabla_x loss(\theta, x, t)) \tag{3.4}$$

More in detail these are the steps followed by (Madry et al.[25]) PGD attack:

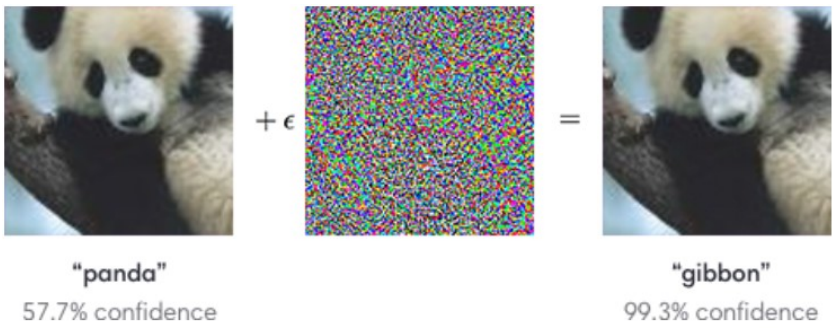

Figure 3.9: Generated Adversarial Example with the FGSM method: the image of the "panda", after attack, fools the network to predict a "gibbon" with confidence equal to 99.3%. Source [23]

1. Start from a random perturbation in the β ball around a sample $x'_i(0)$;
2. Perform a step α in the direction of the maximum loss by following the Equation 3.4;
3. Perform a projection inside the β ball if necessary.
4. Repeat 2–3 times.

By means of this iterative algorithm, the equation finds the maximum loss inside an l_{inf} distance ball $\beta(\epsilon, x)$, generating a more robust attack than FGSM and keeping the size of the perturbation smaller than ϵ. Many versions of the algorithm exist according to the initialization of the perturbation chosen and the metric of the ball evaluation (l_∞, l_1 or l_2).

Figure 3.10 shows some example of attacked images belonging to the MNIST dataset evaluated with L_2 and L_{inf} bounded ball metrics. The l_{inf} bounded attack acts on all pixels of the picture, the l_2 on the pixels with a high deviation between them.

Carlini Wagner

Carlini and Wagner [24] proposed a powerful white-box targeted attack method, by exploiting l_∞, l_1 and l_2 distances to preserve the imperceptibility of the AE. The algorithm solves the following optimization problem:

$$||\delta||_2 + c \cdot max(G(x, \delta, t) - M(x)_t, -k) \tag{3.5}$$

The algorithm iteratively aims to minimize both the components of the equation: (i) the distance δ between the input and the adversarial image and (ii) the distance

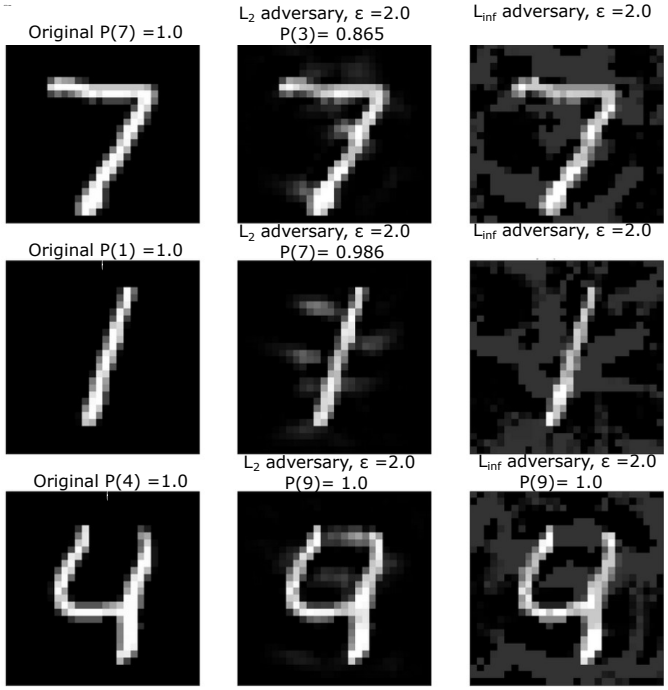

Figure 3.10: Left column: original examples. Middle column: L_2 bounded adversarial examples. Right column: L_{inf} bounded adversarial examples.

between the max output activation ($G(x, \delta, t) := max_{i \neq t}(M(x + \delta))$) and the confidence $M(x)_t$ of the target label t. The value c is updated at every iteration, in order to balance the two terms during the generation of the attacked data. When the confidence of the target label surpasses the other ones, the algorithm starts to decrease the c value in order to minimize the distance between AE and original image, increasing the first contribution of the Equation 3.5.

Many works showed the success of such attacks in fooling CNNs, and provide state-of-the-art success rate results. Some researchers believed that the reasons are over-fitting, insufficient regularization or the introduction of nonlinearity. Goodfellow et al. [23] supposed that the problem of vulnerability is due to the linear behavior of high dimensional CNNs. Given an adversarial example $x^{'} = x + \eta$, and considering a generic activation function $w^T x^{'} = w^T x + w^T \eta$ the perturbation added provides a noisy contribution $w^T \eta$ that propagates along the depth of the network.

3.5.1 Adversarial attacks on Capsule Networks

Recent works showed the vulnerability of CapsNets against adversarial attacks. Frosst et al. [27] investigated the detection of adversarial examples using the reconstruction quality of the CapsNets. Peer et al. [28] and Marchisio et al. [29] applied the FGSM method (Goodfellow et al. [23]) and their proposed attack on CapsNets, respectively. Michels et al. [30] compared the results of different attacks on CapsNets trained on different datasets. However, such experiments need to be properly investigated and analyzed in a systematic way, before employing CapsNets in safety-critical applications.

3.5.2 Adversarial Training

By knowing the power of AE to fool models, many works started to provide solutions against such problems. Specifically, conferring robustness against adversarial attacks increases the interpretability of machine learning models and prevents the learning of brittle features for many real-world applications going beyond security. The most common state-of-the-art defense algorithm against AE is the *adversarial training*, a different Data Augmentation that includes attacked images during training. More in detail, at the end of each backpropagation cycle, the updated weights are exploited for the generation of AEs, the new training data for the next epoch (Madry et al. [25]). This allows that the networks can be trained also taking into account such possible perturbations of external inputs increasing the level of generalization. The algorithm at the same time has (i) to maximize the loss for the generation of the attacked image and, then, (ii) minimize the loss for the model convergence, by modifying the weights of the network. Overall, the following equation has to be minimized:

$$\min_{\theta} \max_{\delta \in \Delta} L(x + \delta, y, \theta) \tag{3.6}$$

Specifically, instead to minimize the loss L for the normal input image x, the minimum is evaluated by considering the attacked image $x_{adv} = x + \delta$, where Δ is a set of perturbations for which the model has to be invariant. The set of perturbation belonging to Δ can be evaluated by considering the different metrics of distance l_1, l_2 or l_{inf}.

Madry et al.[25] using adversarial training with PGD defense for ResNet models, achieves promising results against PGD attack itself, with the only drawback of decreasing a bit the accuracy performances, for large values of perturbations ϵ.

Chapter 4

In-Depth View of our Analysis

Several works analyzed the problem of attacks on CNNs and found many solutions to these problems improving defenses. The introduction of the CapsNets suggests indications that these architectures might be more robust towards adversarial attacks than other CNNs. To demonstrate this intuition, we present a detailed analysis with the aim of answering our main research questions and to show (i) if and why the Capsule Network under attack provides a better response than normal CNNs, (ii) which model quality plays an important role and their limits. Knowing the main differences of CapsNets with respect to traditional CNNs, we explore the impact of these networks on affine transformations and adversarial attacks. Towards a fair and comprehensive evaluation, the results already provided by Michels et al. [30] for the ShallowCaps have been compared with 3 different architectures, chosen according to their properties, their number of parameters and their depth.

- *ResNet20* (He et al. [22]) is one of the best performing CNN architectures for CIFAR10, used in various applications. It would be interesting to compare the capabilities of the CapsNets with a widely used CNN, which is deeper and employs only convolutional and average pooling layers (see Section 3.4).

- a deeper CapsNet architerure, like the *DeepCaps* model [14]. Despite deeper than the ShallowCaps, it has fewer parameters (see Section 3.2).

- another traditional *CNN*, with similar depth and number of parameters as the DeepCaps, with the same depth, but without multidimensional entities such as capsules. Its comparison with respect to the DeepCaps highlights the contribution to the robustness of 3D convolutions and capsules.

All these architecture are tested for a simple dataset like the MNIST and more complex ones like the GTSRB and CIFAR10.

In this work we follow these steps:

1) **Evaluation of robustness on affine transformations.**

   - We train our networks with the normal MNIST, GTSRB and CIFAR10 datasets with the same hyperparameters and data augmentation.
   - We test our trained network on the affNIST dataset, an affine transformed version of MNIST.
   - For the CIFAR10 and GTSRB datasets, we design a novel transformed dataset with random translations, rotations and zooms (which we call *affCIFAR* and *affGTSRB*, see Section 5.1).
   - We use such affCIFAR and affGTSRB datasets for inference, like in the case for affNIST, to evaluate the response of the networks against affine trasformations.

2) **Evaluation of robustness on adversarial attacks.**

   We use the saved parameters of the trained models to evaluate the gradient, with respect to the input, for the two implemented white-box attacks.

   - We apply the projected gradient descent (PGD) attack for each architecture, for the MNIST, the GTSRB and the CIFAR10 datasets, to generate adversarial examples.
   - We test the networks with the generated adversarial inputs, evaluating the behaviour of the accuracy, increasing the level of the perturbation.
   - Then, we apply the Carlini Wagner attack, again for all the datasets.
   - We evaluate the mean distortion required to the algorithm to misclassify 500 images of the test dataset and its fooling rate.
   - As a further step, we apply at the input to a network the adversarial image generated with another one, to test the transferability of the attack.

4.1 Experimental Set-Up

The networks chosen for our analysis are:

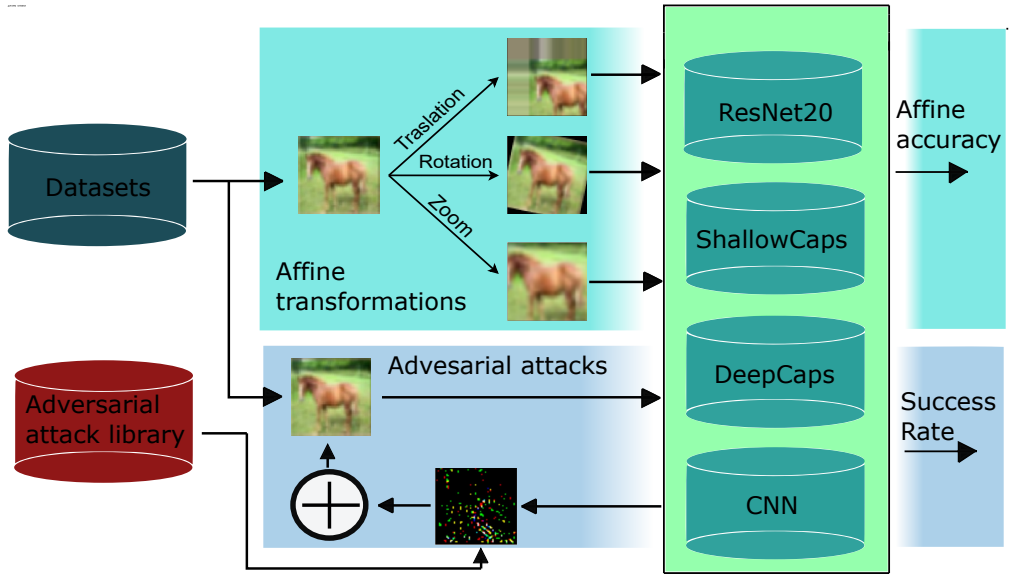

Figure 4.1: Overview of our analysis.

- the original CapsNets (ShallowCaps) [8].[1] It is composed of a convolutional layer with kernel 9 x 9 and stride 2, a convolutional capsule layer with kernel 5 x 5 and stride 1, and a fully connected Capsule layer.
- the DeepCaps [14], with 4 groups of 2D convolutional capsule layers, with the last one that presents a 3D convolution as skip layer, and a fully connected capsule layer of 10 capsules of dimension 32.
- A traditional CNN, similar to the DeepCaps architecture, with same number of layers and same sizes, excluding the concept of capsules. This architecture presents traditional convolutional layers with batch normalization at the output of each layer instead of capsules with squash compression, and a traditional fully connected layer instead of the DigitCaps layer with dynamic routing.
- the ResNet20 [22], a CNN with batch normalization at the end of each layer, composed of Residual Blocks and a final Average Pooling layer.

These architectures have been trained with the 40×40 sized version of the MNIST dataset, and tested on the affNIST for evaluating the robustness against affine transformations. For all the architectures tested on CIFAR10, input data have been resized before the training, from 32×32 to 64×64, following the pre-processing

[1]For the CIFAR10 and affCIFAR datasets, the architecture is a slightly different version of CapsNet, as compared to the model of [8], which is adapted for the dataset.

steps used in [14]. For the GTSRB the size of the input images are kept 32×32. The data augmentation and hyperparameters used for the training are kept the same for all the networks. As regularization term, for capsule architectures, it has been used the reconstruction loss provided by the decoder. For the evaluation of the loss, we use the same as in [8] for CapsNets (see Section 3.1) and the Cross Entropy for CNNs. Table 4.1 summarizes the basic characteristics of these networks, and their differences.

Table 4.1: Characteristics of the CapsNets and CNNs that are used in our experiments.

Network	params MNIST40x40	params GTSRB	params CIFAR10	depth
ShallowCaps	11,288,609	16,695,059	112,508,163	3
DeepCaps	9,674,433	13,918,915	13,427,283	13
CNN	7,475,978	7,611,435	7,596,042	13
ResNet20	274,154	276,587	276,362	20

The adversarial attack algorithms used for our analyses are:

- Projected Gradient Descend (PGD): the iterative version of FGSM. It is a white-box attack and for our evaluations we chose its untargeted version.
- The Carlini Wagner (CW): a targeted white-box attack.

For more details about the attacker algorithms see Section 3.5. For the algorithms implementation, we exploited the cleverhans [31] library, adapted for the Keras framework [32] with Tensorflow backend [33].

The training of the networks it has been run on the Nvidia RTX-2080Ti GPU with CUDA 10.

Chapter 5

Capsule Networks Robustness

5.1 Robustness Against Affine trasformations

5.1.1 Affine Datasets Generation

While a dataset with affine transformed images of the MNIST (affNIST) is already available, we create an affine version of the CIFAR10 and GTSRB datasets, which we call *affGTSRB* and *affCIFAR* respectively, to compare the response of the networks defined in Section 4.1 against affine transformations. The test data was created by modifying the whole test images from the original datasets with random affine transformations. Every image is randomly transformed with translations, rotations and zooms following these criteria:

- *Translations:* random pixels translations in one or in two dimensions by a factor between 10% and 25% of the input image size, with a fixed interval.

- *Rotations:* random rotations between +20 and -20 degree with a predefined fixed step.

- *Zooms:* the vertical and the horizontal expansion are chosen uniformly between 0.8 (i.e. shrinking the image by 20%) and 1.2 (i.e. making the image 20% larger).

Figure 5.2 and Figure 5.1 show some example images of our affCIFAR anf affGTSRB datasets rispectively.

Figure 5.1: Affine GTSRB (affGTSRB) dataset.

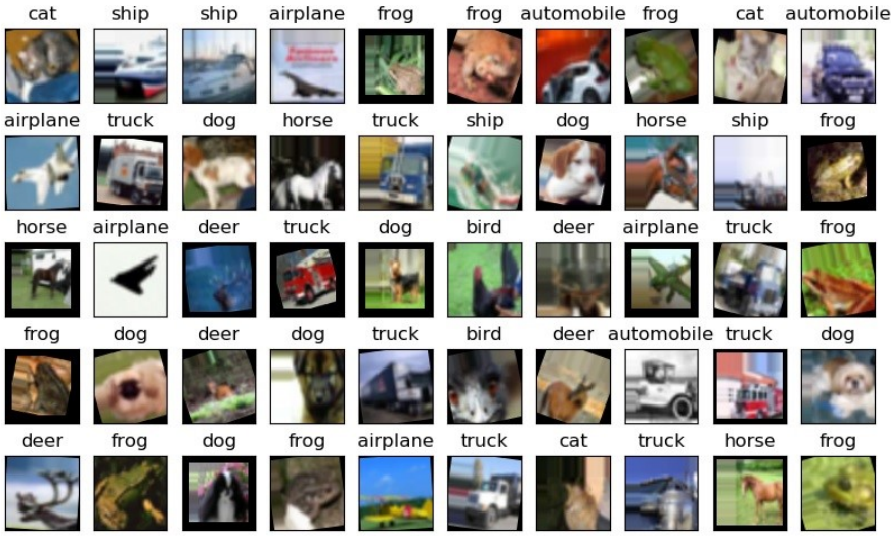

Figure 5.2: Affine CIFAR10 (affCIFAR) dataset.

5.1.2 Affine Trasformations Results

For each architecture model defined in Section 4.1, we evaluate the accuracy for the MNIST[1], GTSRB, CIFAR10, affNIST, affGTSRB and the affCIFAR10 datasets. The results are shown in Table 5.1 and Figure 5.3.

Table 5.1: Robustness results against affine trasformations.

Network	MNIST40	GTSRB	CIFAR10	AffNIST	affGTSRB	AffCIFAR
ShallowCaps	99.18%	95.29%	77.32%	75.61%	78.88%	48.85%
DeepCaps	99.19%	95.64%	91.52%	87.60%	84.14%	**78.66%**
CNN	99.22%	94.73%	91.68%	82.83%	79.03%	69.90%
ResNet20	99.16%	96.39%	91.48%	**96.39%**	**89.75%**	75.84%

ShallowCaps vs. DeepCaps

Accordingly to the results proposed by [8], with the ShallowCaps is possible to achieve better accuracy on the transformed version of the MNIST dataset, which is an unreachable result with traditional CNNs of similar size.

The big limit of the basic ShallowCaps is that the number of capsules and the layers involved in the model are not enough to better generalize more complex datasets, like the CIFAR10, and affine transformed images, despite the high number of parameters required. As shown in the results of Table 5.1, in fact, the ShallowCaps on the CIFAR10 dataset achieves an accuracy far below the state-of-the-art (just 77.32%).

To reach reasonable results with more complex datasets, like the CIFAR10, and ensure generalization (and also robustness), the ShallowCaps architecture needs improvement in terms of number and size of capsules, with the drawback of increasing a lot the dimension of the transformation matrix and the parameters to train (see Section 2.6.1).

Then, going deeper, preserving the capsule structure along with the architecture, not only it is possible to reduce the number of parameters but is also possible to gain better results with CIFAR10 (91.52%) and with the affine transformed datasets. In fact, by considering the results achieved with affMNIST and affGTSRB, despite the shallower model reaches a good accuracy on the normal datasets (99.18% and

[1]All the simulation and results obtained with the MNIST dataset are referred to 40×40 version in order to make the trained model compatible with testing on the affMNIST dataset.

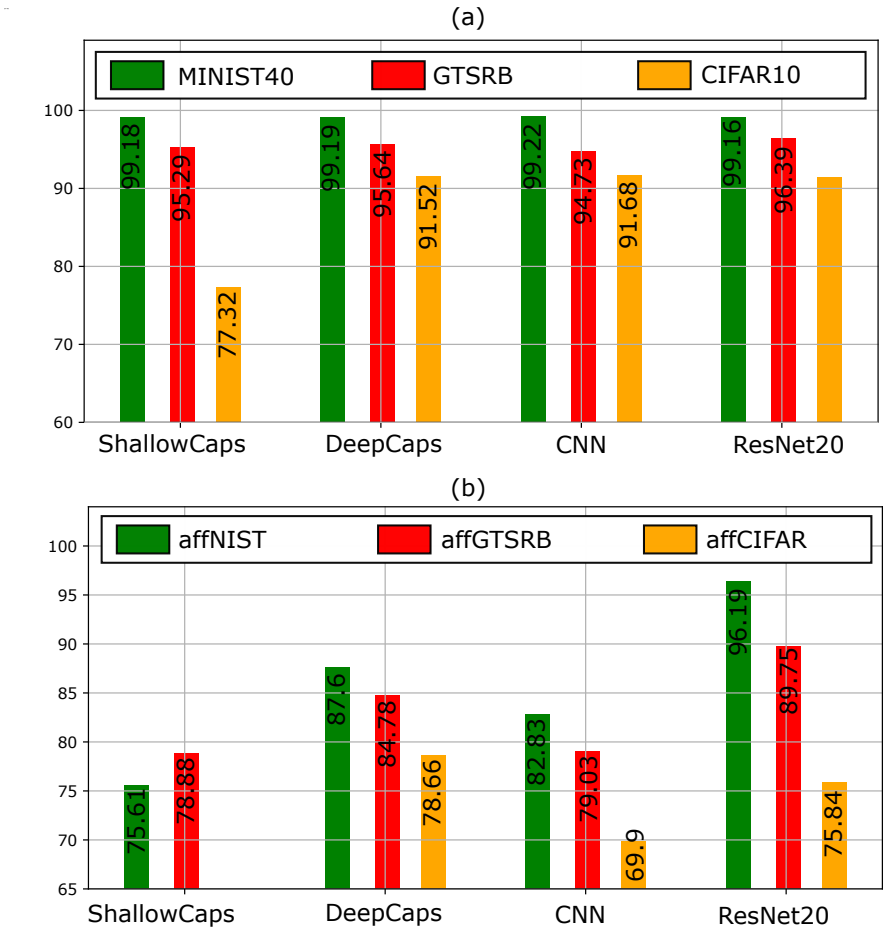

Figure 5.3: (a) Normal accuracies, (b) affine transformation results.

95.29% respectively), it is still not able to generalize as the DeepCaps against affine transformations.

A further contribution to the improved accuracy of the affine datasets with DeepCaps, could be explained by the presence of the 3D convolutional layer. As explained in Section 2.6, the DigitCaps layer, i.e., last layer of the ShallowCaps, plays a similar role as a fully connected layer in CNNs, with the difference that the basic block is not a single neuron but a group of neurons. The effect of having 3D convolutions, compared to a stack of fully connected capsules, is similar like when we compare the generalization level offered by the Multy Layer Perceptrons (MLP) and the CNNs. As shown in Figure 5.4, in the DigitCaps layer, each element of the transformation matrix W_{ij} learns if capsules of two adjacent layers are correlated. When the affine transformations are applied, the information previously stored in the capsule u_i could be then associated to a different capsule u_j and, therefore,

the transformation element W_i might not be able to detect the feature anymore. On the contrary, with the 3D convolution, sliding a 3D kernel, the same weights are used among all the capsule of the layer. This characteristic allows to learn the presence of a particular feature also if the input image is spatially transformed (e.g., traslated, rotated, or zoomed). The results in Figure 5.3b and in Table 5.1 proves that a deeper network performs better on affine transformations than the ShallowCaps for all the datasets.

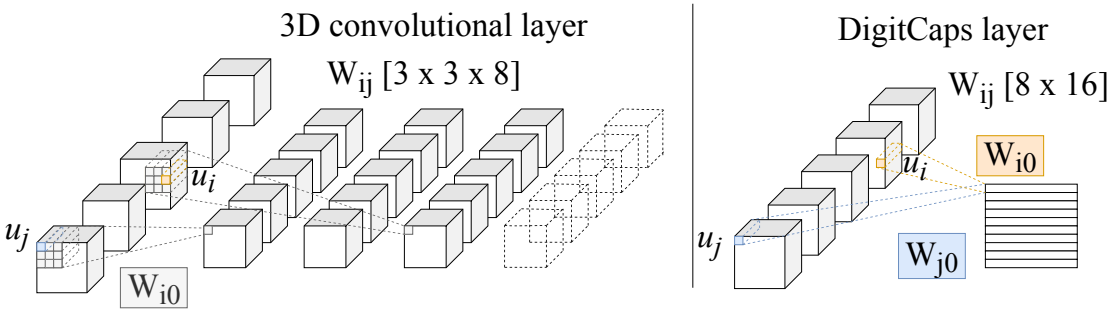

Figure 5.4: 3D convolution vs DigitCaps layer

DeepCaps vs. CNN and ResNet20

Another significant result is provided by comparing the DeepCaps with a traditional CNN with a similar base architecture. While the accuracy on the MNIST, CIFAR10 and GTSRB datasets are similar to the DeepCaps, CNN robustness against the affNIST, affGTSRB and affCIFAR is much lower. These results confirm the benefits of capsules against affine transformations.
The ResNet20, instead, compared to the DeepCaps, is deeper, but with a lower number of parameters. It is able to generalize better for the affMNIST and affGTSTB, but worse for the affCIFAR dataset. This apparently contradictory result is due to the difference of complexity between the datasets. While for simple datasets a deep CNN like the ResNet20 can generalize very well, for more complex tasks like the affCIFAR it is outperformed by the DeepCaps. This highlights the capability of the capsule architectures to preserve spatial correlations between the features detected, with respect to traditional CNNs, and this difference is even more clear when the input dataset is composed of complex features like the CIFAR10.

5.2 Robustness Against Adversarial Attacks

5.2.1 Projected Gradiend Descent (PGD) Attack

We analyze the response of the networks defined in Section 4.1 increasing the level of the perturbation ε of the images generated by the PGD algorithm. Figures 5.5a, 5.5b and 5.5c show the results for the MNIST, GTSRB and the CIFAR10 datasets, respectively. Since a successful PGD attack makes the classification accuracy drop down to 0%, the robustness of the networks is evaluated as the accuracy.

ShallowCaps vs. ResNet20

Applying the PDG attack for the MNIST dataset, the ResNet20 appears to be less vulnerable than other networks, for low levels of ε. The ShallowCaps robustness behavior, not so far from the one of the ResNet20, overperforms the ResNet20 when $\varepsilon \approx 0.065$. Hence, despite the low number of layers, the ShallowCaps is affected by the PGD attack similarly as a deeper CNN.

DeepCaps vs. ShallowCaps

According to the results, the ShallowCaps is more robust than the DeepCaps, in contrast to what happens for affine transformations. This means that increasing the depth of a CapsNet does not provide more robustness against perturbed images.

Note, the response of the ShallowCaps for the CIFAR10 dataset has not been examined, because of its low baseline accuracy and the huge number of parameters required, therefore not comparable with other networks.

DeepCaps vs. ResNet20 vs. CNN

For this kind of algorithm and the MNIST dataset, the Figure 5.5a shows that the DeepCaps works worse than the ResNet20. On the contrary, for a more complex datasets like the CIFAR10 and the GTSRB, the results, in Figure 5.5, show that the ResNet20 is not as robust as the MNIST dataset.
Note, increasing the size of the perturbation, the success rate of the attacks grows faster than on DeepCaps. Such outcome is inline with the takeaway from Section 5.1.2, which showed the DeepCaps be more robust than the ResNet20 against the transformation in the affCIFAR.
The behavior of the CNN curve, for GTSRB and CIFAR10, always stays below the curve of the DeepCaps. It means that the capsule architecture plays a fundamental role in improving the robustness against the PGD attacks when the dataset becomes more complex.

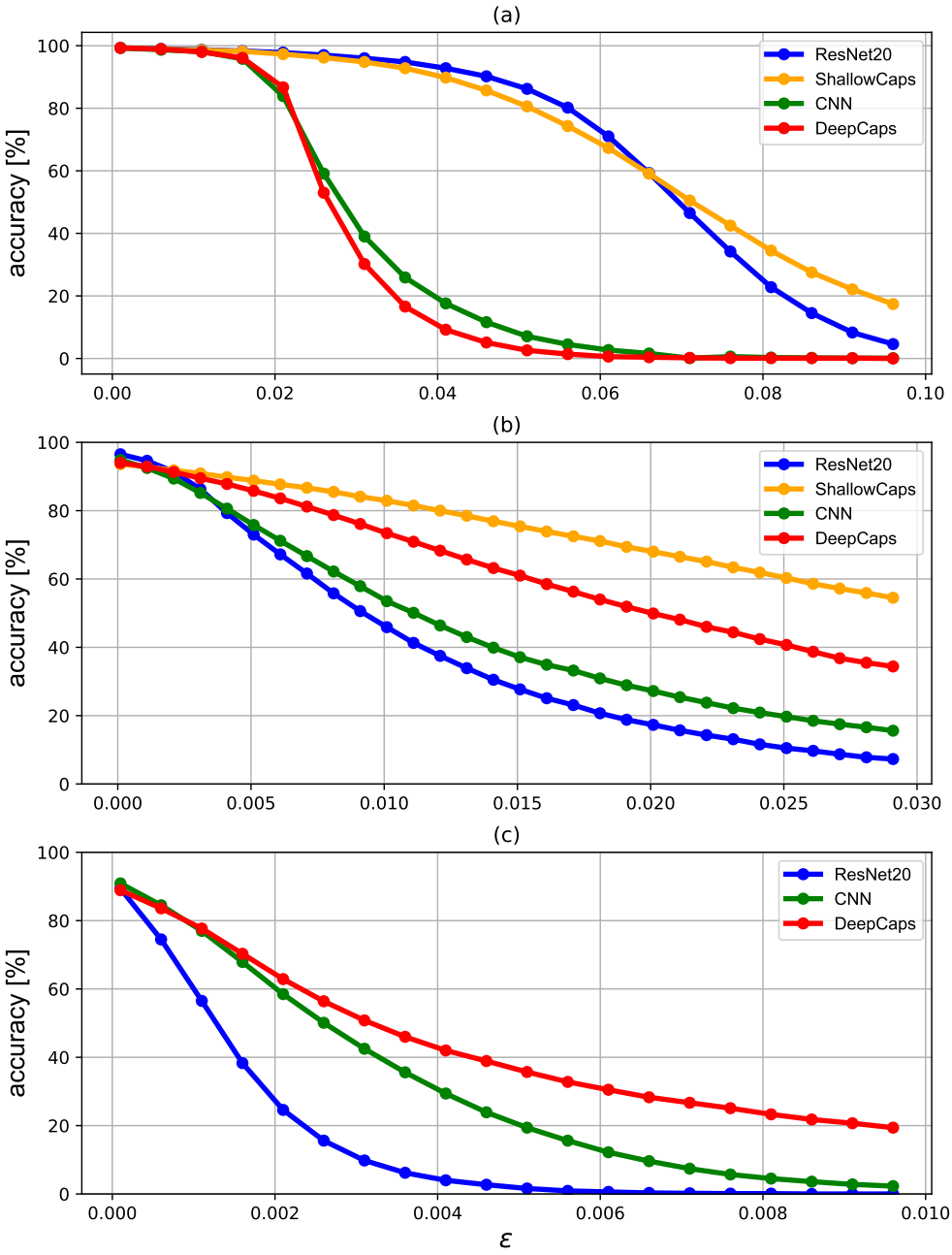

Figure 5.5: Robustness against the PGD attack for (a) the MNIST, (b) the GTSRB and (c) the CIFAR10 datasets.

Transferability ResNet20 $\Longleftrightarrow$ DeepCaps

Towards a more comprehensive study of the robustness against the PGD attack, we analyze the transferability of the attacks, between the ResNet20 and the DeepCaps,

presenting the two opposite behaviors. We provide at the input to the DeepCaps the adversarial examples generated with the gradient of the ResNet20, and vice-versa. The Figures 5.6a and 5.6b show the transferability for the MNIST and for the CIFAR10 datasets, respectively.

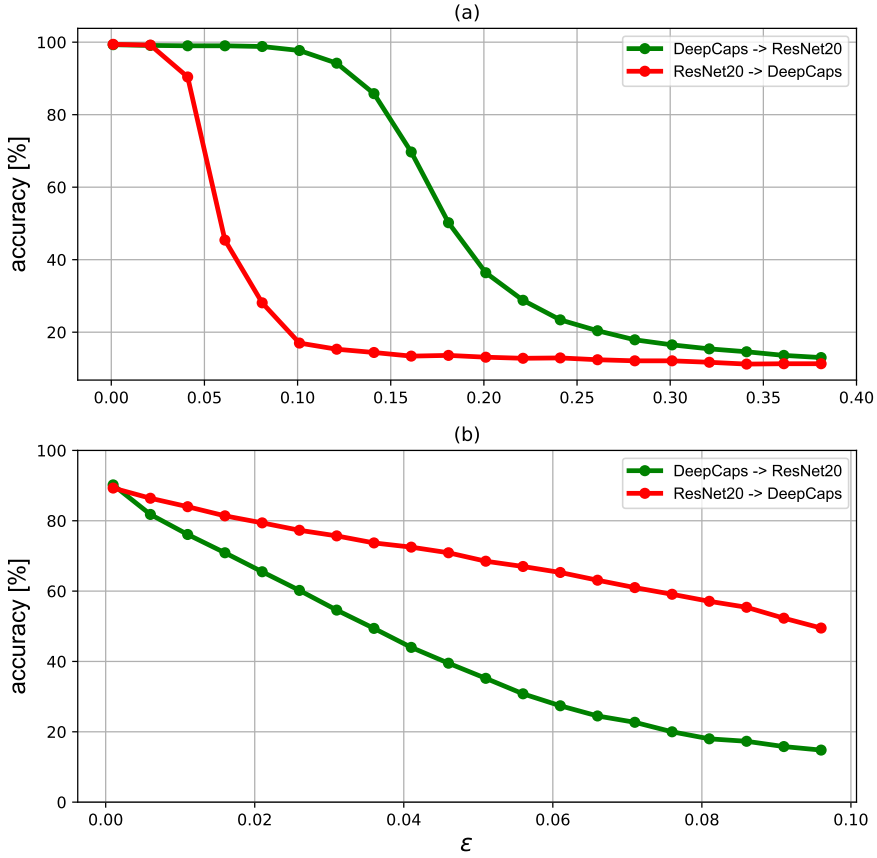

Figure 5.6: Transferability for the PGD attack: comparison of the networks response for (a) the MNIST and (b) the CIFAR10 datasets.

For the MNIST dataset, the attacks generated for the ResNet20, tested on DeepCaps, have a greater effect than the other way round. This outcome confirms, like in Figure 5.6a, that the ResNet20 appears robust for the generalization of the MNIST. The opposite results can be derived for the CIFAR10 dataset, where the DeepCaps shows greater robustness than the ResNet20, due to a better ability to generalize for a more complex dataset.

Transferability ShallowCaps $\Longleftrightarrow$ DeepCaps

For the GTSRB dataset the transferability is tested between the ShallowCaps and the DeepCaps, in order to appreciate a further comparison between the two different capsule architectures with different depths. The Figure 5.7 shows the two curves by providing at the input of the DeepCaps the AEs generated with the gradient of the ShallowCaps and vice-versa. The results prove again that the

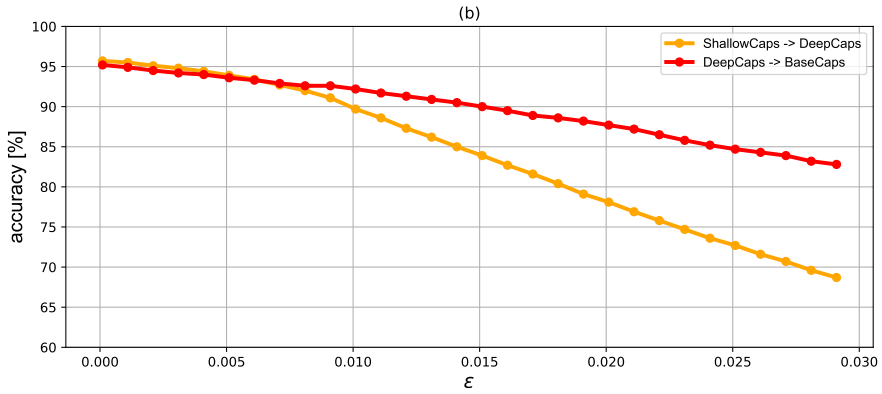

Figure 5.7: Transferability for the PGD attack: comparison of the networks response for GTSRB dataset.

ShallowCaps ensures a better robustness against such attack, because its curve behaviour overperforms the DeepCaps at increasing the perturbation ϵ.

5.2.2 Carlini Wagner (CW) Attack

To have a more solid comparison, the CapsNets and CNNs have been tested also against the CW attack, a different kind of algorithm that does not define a threshold for the magnitude of the perturbation (like the ε in the PGD attack). It is an iterative targeted algorithm that tries to reduce the gap between the target and the predicted class (success rate) with the minimum perturbation (mean distortion), estimated as the l_2 distance. The more robust the network, the more iterations the probability of the targeted class needs to overcome the probability of the targeted class. As a consequence, more iterations also imply a higher l_2 distance between the original image and the adversarial example. For our estimations, we set a maximum of 10 iterations for the MNIST, and 5 iterations for the CIFAR10 and GTSRB datasets. In addition, for the attacks on CIFAR10 and GTSRB, the algorithm has been forced to set the confidence of the targeted class to be 0.5 higher than the confidence of the true label. Table 5.2 reports the fooling rate, i.e., the percentage of successful attacks, and the mean distortion for both the datasets.

Table 5.2: Robustness results against the CW attack.

Network	MNIST		GTSRB		CIFAR10	
	Mean Distortion	Fooling Rate	Mean Distortion	Fooling Rate	Mean Distortion	Fooling Rate
ShallowCaps	1.59	98.6%	1.31	100%	-	-
Deepcaps	1.24	86.8%	1.16	98.8%	0.34	100%
CNN	0.95	100%	0.59	100%	0.23	100%
ResNet20	0.94	100%	0.34	100%	0.12	100%

CapsNets vs. CNNs

The CW attack is very effective for traditional CNNs. In fact, it reaches 100% fooling rate for all the MNIST, the GTSRB and the CIFAR10 datasets, as also shown in [24]. On the other hand, both CapsNets show greater robustness (i.e., lower fooling rate) than CNNs, for the MNIST dataset (and also for GTSRB even if the Fooling Rate of the DeepCaps is just a little bit lower than 100%). The CapsNets also require a higher mean distortion than the CNNs, hence, the resulting adversarial example would be more perceptible to fool the networks.

For the CIFAR10 dataset, the CW attack shows its effectiveness, because for all the networks the fooling rate is 100%. However, we can notice higher robustness of the CapsNets due to a higher mean distortion.

DeepCaps vs. ShallowCaps

The DeepCaps shows to be more robust than ShallowCaps, because of a lower fooling rate, despite having slightly lower mean distortion. Therefore, the depth and the 3D convolutions help to generalize better against such attack.

Transferability ResNet20 $\Longleftrightarrow$ DeepCaps

The Table 5.3 shows the transferability between the ResNet20 and the DeepCaps for the CW attack. The reported values represent the accuracies of the two models that receive as input a sample of 500 targeted AEs generated by the CW algorithm applied to the other network.

The high accuracy values demonstrate the low level of transferability of the targeted CW attack. Despite this, the ResNet20 still achieves lower accuracies than DeepCaps, thereby performing less robust.

Table 5.3: Transferability of the CW attack between the DeepCaps and the ResNet20.

Network	MNIST	CIFAR10
DeepCaps → ResNet20	97.4%	86.8%
ResNet20 → DeepCaps	**97.8%**	**89,2%**

Transferability ShallowCaps ⟺ DeepCaps

Like done for PGD, we compare the behaviour of the two CapsNets by studying the transferability of CW between them, exploiting the GTSRB dataset.

Table 5.4: Transferability of the CW attack between the ShallowCaps and the DeepCaps for GTSRB.

Network	Mean Distortion	Fooling Rate
DeepCaps → ShallowCaps	2.16	**58.2%**
ShallowCaps → DeepCaps	2.15	57.8%

The results in Table 5.4 show again the low level of the transferability of such attack, demonstrated by the lower fooling rates with respect to the original ones achieved in Table 5.3. Both fooling rate and mean distortion are not so different for both the cases, even if the DeepCaps appears a bit more robust against the attacked image generated with the gradient of the ShalloCaps, contrary to what happens for PGD.

5.2.3 Projected Gradient Descent Adversarial Training

After analyzing the robustness of CapsNets compared to traditional CNNs, we apply the defense algorithm proposed by Madry et al. [25] on the DeepCaps.

We test the adversarial training on the GTSRB and the CIFAR10 datasets, including in the backpropagation process the AEs generated with the PGD attack, (see Sections 3.5.2).

The choice of the training perturbation magnitude has been influenced by the previous analysis, taking into account the behavior of the normally trained DeepCaps against the PGD attack (see Figure 5.5). For the adversarial training with the CIFAR10 dataset, we chose an input perturbation equal to $\epsilon = 0.003$, a

value for which the accuracy of the DeepCaps model without defense is already around 50%. Moreover, increasing the magnitude of the perturbation during training, the final accuracy starts to move away from the original one, thereby preventing the training from converging.

For the GTSRB, we chose a perturbation magnitude equal to $\epsilon = 0.03$, a value for which the DeepCaps without defense is already working worse than a random classifier (see Figure 5.5).

The Figure 5.8 shows the response of the adversarially trained DeepCaps against the PGD attack, for GTSRB and CIFAR10 datasets.

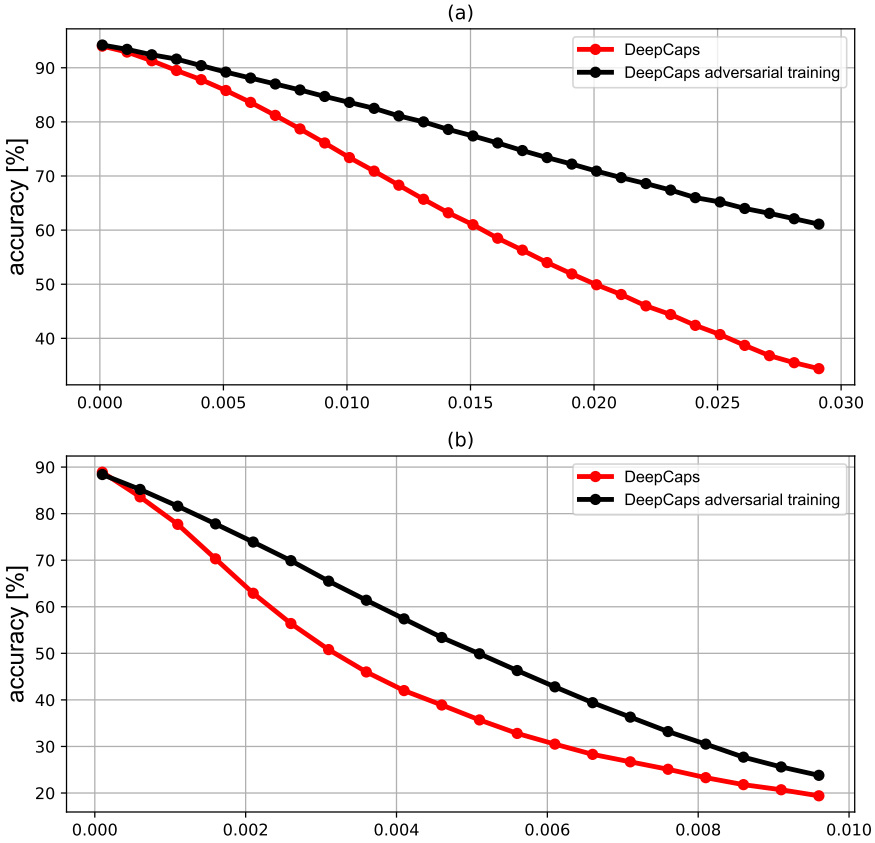

Figure 5.8: Asversarially trained DeepCaps with (a) the GTSRB, (b) the CIFAR10 datasets.

From the Figure 5.8, we can derive that the adversarial training increases the robustness of the DeepCaps against the PGD attack, both for the CIFAR10 and the GTSRB datasets, even if with the more complex CIFAR10, the defenses are more difficult to achieve. This because the CIFAR10 dataset reaches a lower final accuracy using lower perturbation magnitudes than GTSRB during the training.

Accordingly with the discussions provided in the previous sections, the difference with respect to the GTSRB dataset could be explained by the higher complexity of the CIFAR10 dataset.
It is also interesting to analyze how the adversarially trained model reacts against Affine Transformations and the CW attack.

- **Affine Transforations response**:

Table 5.5: Comparision between adversarially and normally trained model against affine transformations.

Network	GTSRB	affGTSRB	CIFAR10	affCIFAR
Normally trained model	**95.64%**	**84.14%**	**91.52%**	**78.66%**
Adversarially trained model	94.08%	79.73%	89.22%	73.04%

The Table 5.5 shows that the adversarial training procedure produces drawbacks against affine transformations, as well as the normal accuracy gets worse for both the datasets.

- **CW attack response**

Table 5.6: Comparision between adversarially and normally trained model against the CW attack

Network	GTSRB		CIFAR10	
	Mean Distortion	Fooling Rate	Mean Distortion	Fooling Rate
Normally trained model	1.16	98.8%	0.34	100%
Adversarially trained model	**1.44**	**98.6%**	**0.84**	**96.6%**

In contrast to what happens for affine transformations, the adversarial training with PGD defense helps the networks also against the CW attack. For both the datasets, from the Table 5.6,comparing both the mean distortion and the fooling rate, it appears that the adversarially trained model is more robust. This means that the adversarial training improves the model interpretability and reduces the learning of brittle features, also if the attacker algorithm used for defenses is different from the one used for the testing.

Chapter 6

Contribution of the Dynamic Routing to the Robustness of the DeepCaps

As a further analysis, we explore the contribution of the Dynamic Routing (DR) towards the generalization and, as a consequence, towards the robustness, on the CapsNets.

In order to have a good metric of comparison, we train different versions of the DeepCaps architecture modifying the routing algorithm:

1. the original DR with 3 iterations has been replaced by a simple connection (i.e., one iteration of DR), in both the 3D convolutional and the DigitCaps layer.

2. the DR has been replaced by the Self Routing (SR) algorithm in the last fully connected layer, while in the 3D convolutional layer it has been left the 3D version of the DR .

We run the experiments on such networks and compare them with the original DeepCaps model already analyzed in the previous Chapter 5. From such comparisons, we can evaluate how the robustness significantly changes, either incorporating the coupling coefficients of DR in the training process (removing iterations) or introducing new coupling weights between two capsule layers (W^{route} in SR, see Section 3.3).

6.1 Affine Trasformations

The results reported in Table 6.1 compare the accuracy achieved by the original DeepCaps (with DR), without DR and with SR, for MNIST, GTSRB and CIFAR10 datasets.

Table 6.1: Robustness results against affine trasformations for the different DeepCaps models, according to the routing algorithm.

Network	MNIST40	GTSRB	CIFAR10	AffNIST	affGTSRB	AffCIFAR
Without Routing	**99.27%**	**96.27**%	91.47%	87.72%	**84.54%**	**79.86%**
Dynamic Routing	99.19%	95.29%	**91.52%**	87.60%	84.14%	78.66%
Self Routing	99.25%	95.60%	90.5%	**88.15**%	83.17%	77.37%

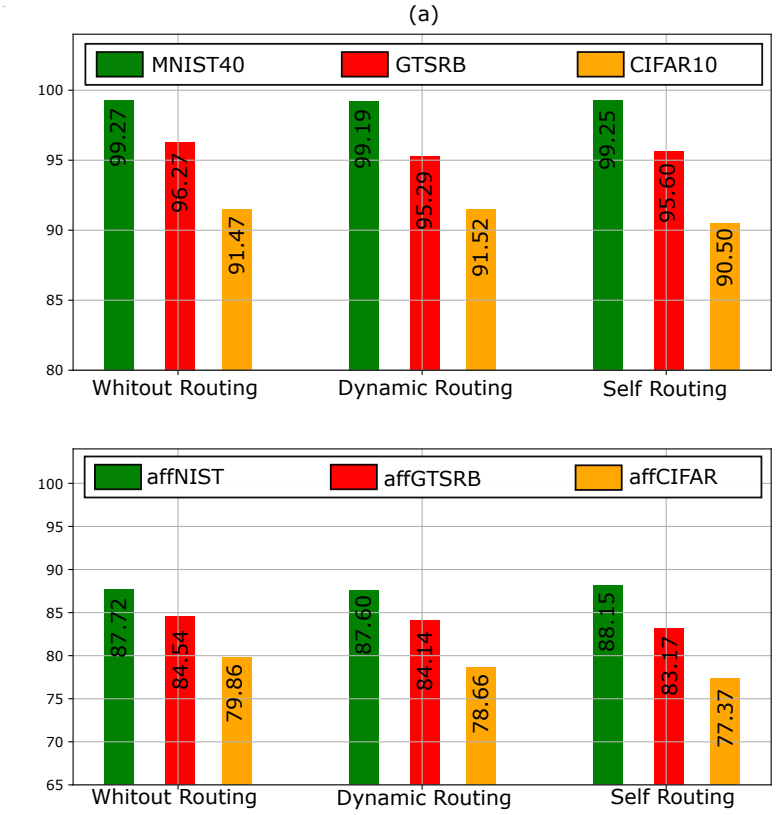

Figure 6.1: (a) Normal accuracies, (b) Affine Transformation results.

While the difference is minimal, the response of the DeepCaps without DR

against affine transformations appears to be slightly better than the others.

- For the MNIST dataset, the response against affine transformations of the three architectures is pretty much similar.
- With GTSRB both for the transformed and untransformed datasets, the DeepCaps without DR works better than the others.
- For the CIFAR10, even if the accuracy with the normal dataset reaches 91.52% with the DR, higher than 91.47% without it, the latter works better for the affCIFAR. The SR appears to be slightly the worst, showing some limits with a complex task like the CIFAR10.

We can derive that the DR does not provide a significant contribution to the robustness against affine transformations, indeed, it makes the DeepCaps much computationally heavier. Since the accuracies are close to each other, we analyze in detail what happens to the output activations, to explain why the DR may not contribute to the robustness of the network.

Dynamic Routing vs. Without Dynamic Routing

The functionality of the DR is to inhibit the propagation of the activation vectors with a lower contribution, by lowering the values of the coupling coefficients in such connections (see Section 2.6.3).
The Figure 6.2a and 6.3a, show the *coupling factors*, after three iterations of the DR in a DeepCaps architecture, providing in input an image belonging to the CIFAR10 and the MNIST dataset respectively.
The coupling factors are evaluated by summing all the coupling coefficients linked with each output capsule and subtracting the minimum between them. For example, for the CIFAR10 and the MNIST, 10 coupling factors are defined, each one representing the coupling with the 10 output classes.
By analyzing in detail the Figure 6.2:

- For the *original image* (an auto in the Figure 6.2c), the DR algorithm correctly updates the coefficients, as demonstrated by the highest coupling factor in correspondence of the true label (blue bars of the Figure 6.2a). Therefore, the output activativation of the model with DR achieves a higher confidence than the architecture whitout it (0.88 vs 0.85 respectively, comparing the orange bars of Figures 6.2b and 6.2d).
- For the *transformed image*, instead, the highest coupling factor amplifies the coupling with the wrong label (green bars of Figure 6.2a), fooling the network with DR to mispredict a truck. Therefore, the confidence of the predicted class is wrongly increased by the DR, making things worse. In fact, despite both the

architectures do not correctly classify the adversarial image (i.e, track instead of auto), the confidence of the incorrect predicted label with DR overperforms the one achieved without it (0.92 and 0.79 respectively, from the red bars of the Figures 6.2b and 6.2d).

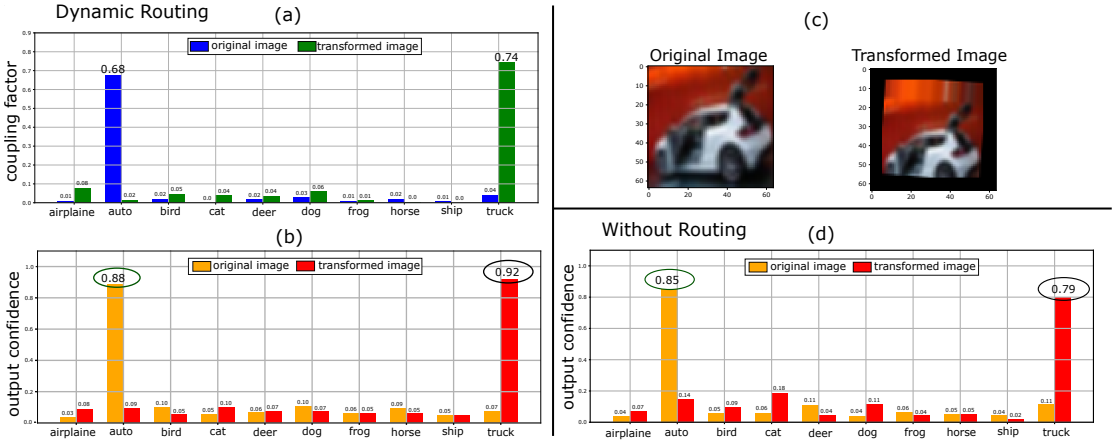

Figure 6.2: Comparison between the output activations of the DeepCaps with and without DR, by providing in input an affine transformed image of the CIFAR10 dataset. Left side: (a) the coupling factors and (b) the output activations of the DeepCaps with DR. Right side: (c) the input image used for the comparison, (d) the output activations of the DeepCaps without DR

The Figure 6.3 shows another example similar to the previous one, that underlines the limits of the DR on the robustness, also for the MNIST dataset.

It is possible to observe that also the *original image* (in Figure 6.3c), is not well classified by the architecture with DR. The algorithm amplifies with the same intensity the coupling with two different output capsules (the ones corresponding to the class 2 and the class 6, as suggested by the blue bars of the Figure 6.3a). Therefore, the confidence corresponding to the wrong label, overperforms the correct one with a confidence equal to 0.42 (orange bars of Figure 6.3b). Without DR instead, the architecture correctly classifies both the untransformed and the transformed images (Figure 6.3d).

6.2 PGD Adversarial Attack

Following the steps of our analysis, we investigate the contribution of the DR against the Projected Gradient Descend (PGD) and Carlini Wagner (CW) attacks. The comparison analysis for the PGD attack applied to the MNIST, GTSRB and CIFAR10 datasets are shown in Figures 6.4a, 6.4b and 6.4c, respectively.

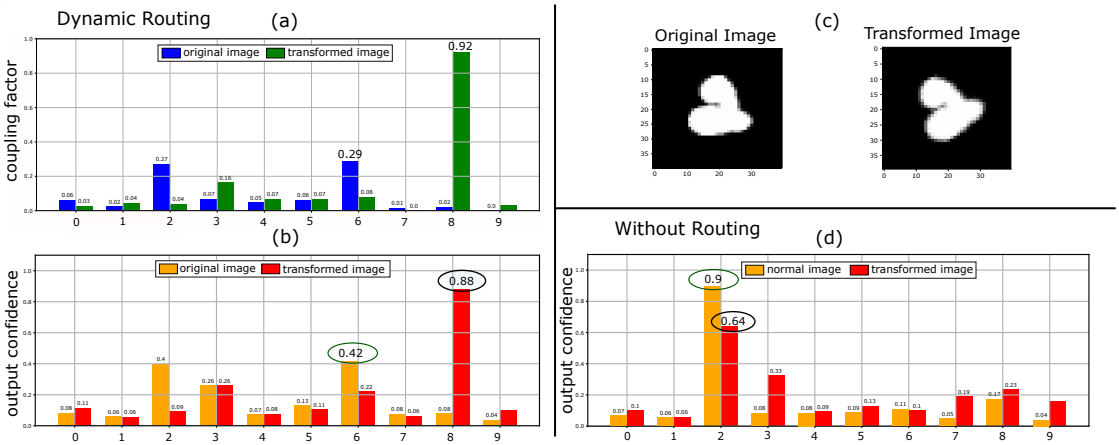

Figure 6.3: Comparison between the output activations of the DeepCaps with and without DR, by providing in input a transformed image of the MNIST dataset. Left side: (a) the coupling factors and (b) the output activations of the DeepCaps with DR. Right side: (c) the input image used for the comparison, (d) the output activations of the DeepCaps without DR.

- For the MNIST dataset, the DeepCaps with DR is more robust than the version without it. The SR increases the robustness of the network when the level of the perturbation starts to be large, but when the accuracy is already below the 40%.
- For the GTSRB the three architectures follow the same behavior.
- For the CIFAR10, instead, the accuracy of the DeepCaps without DR decreases slightly faster, when increasing the perturbation ε. The SR does not contribute to increase the robustness with such attack.

From our analysis we can understand that increasing the complexity of the dataset, from MNIST toward the CIFAR10 as in Figure 6.4, the DR does not improve the classification capability when the input starts to be perturbed. Furthermore, the SR does not contribute to the robustness, when the task becomes complex and when the inputs too noisy.

Dynamic Routing vs. Without Dynamic Routing

Also in this case, we conduct the same analysis carried out against affine transformations. We analyze how the DR, by updating the coupling coefficients, influences the output activations, in case of PGD attack. As for affine transformations we compare the response of two DeepCaps architectures with and without DR.

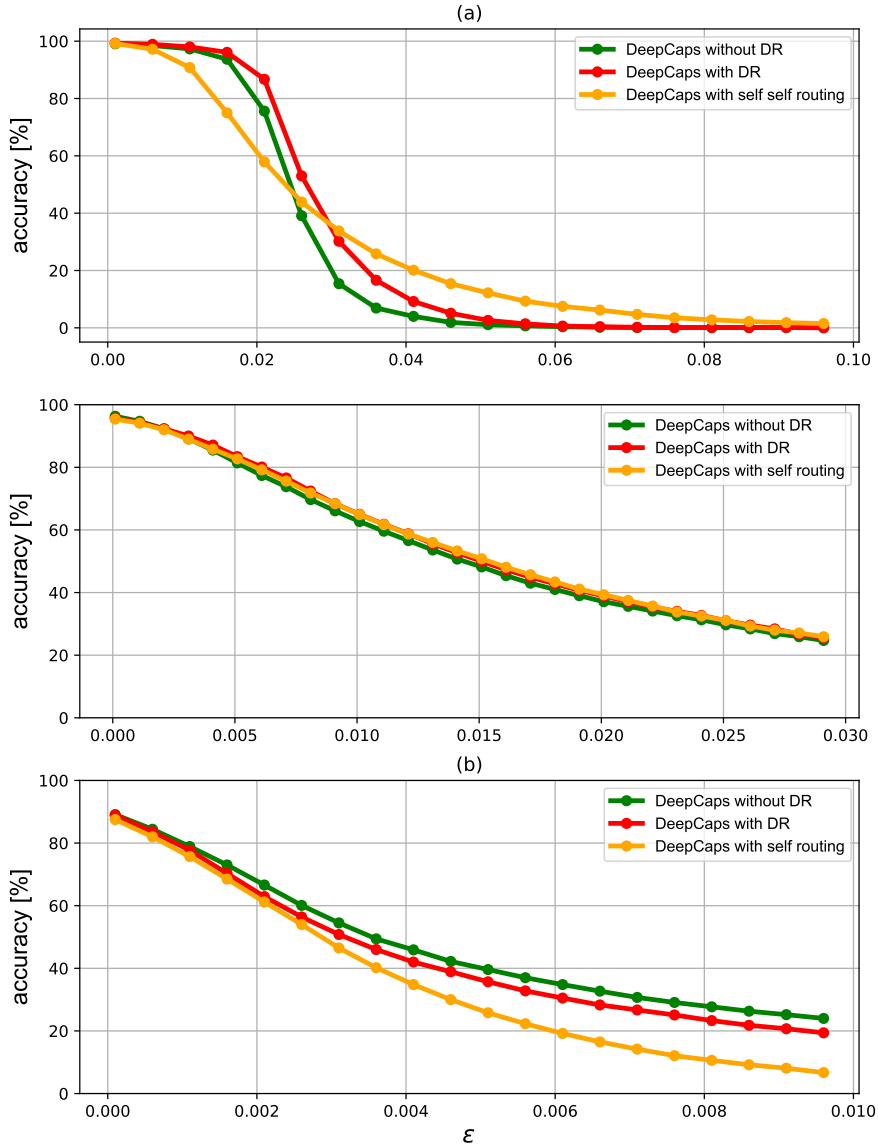

Figure 6.4: PGD results: comparison of the DeepCaps response with Routing and without Routing for (a) the MNIST, (b) the GTSRB datasets (c) the CIFAR10.

- For the *original image* belonging to the MNIST dataset (Figure 6.5c) the coefficients increase the coupling with the correct output capsule, and reduce the others. This is proved by the higher coupling factor in correspondence of the true label (Figure 6.5a) and by the fact that the confidence with DR is higher than without it (0.96 and 0.87, comparing Figures 6.5b and 6.5d).
- For the *adversarial image*, in most of the cases, especially for low levels of

perturbation, the DR, for the simple task of MNIST classification, works better, as suggested by the Figure 6.4a. In the specific case, the adversarial image in Figure 6.5c, perturbed with *epsilon* = 0.03, fools the architecture with DR, increasing the confidence of the wrong label (red bars of Figure 6.5b). The architecture without DR, instead, is still able to classify the perturbed image (red bars of the Figure 6.5d)

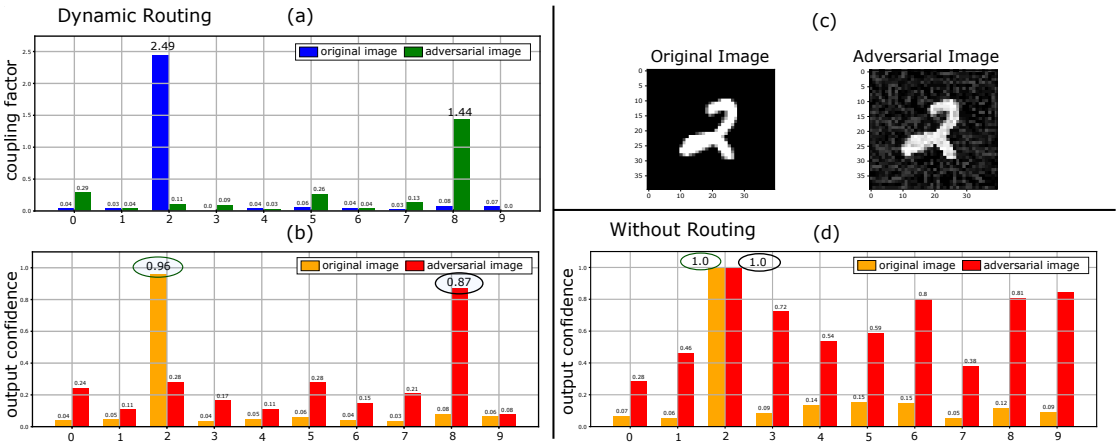

Figure 6.5: Comparison between the output activations of the DeepCaps with and without DR, by providing in input an adversarial image of the MNIST dataset. Left side: (a) the coupling factors (b) the output activations of the DeepCaps with DR. Right side: (c) the input image used for the comparison, (d) the output activations of the DeepCaps without DR

The Figure 6.4c suggests that with a higher level of perturbation the AEs with CIFAR10 are slightly better classified without DR. The Figure 6.6 shows the behavior of the coupling coefficients, for an example of the CIFAR10 dataset, with a perturbation equal to $\epsilon = 0.004$. This allows to understand what happens to DR when the perturbation starts to increase for a complex task like the CIFAR10.

This analysis exhibit that also the *original image* is not correctly classified by the architecture with DR, that increases the coupling with the wrong output label, i.e while the original image is a cat, the routing increases the coupling with the label corresponding to the dog (Figure 6.6a). The DeepCaps without DR, instead, correctly classifies the cat, also when the input is perturbed with PGD attack (Figure 6.6d).

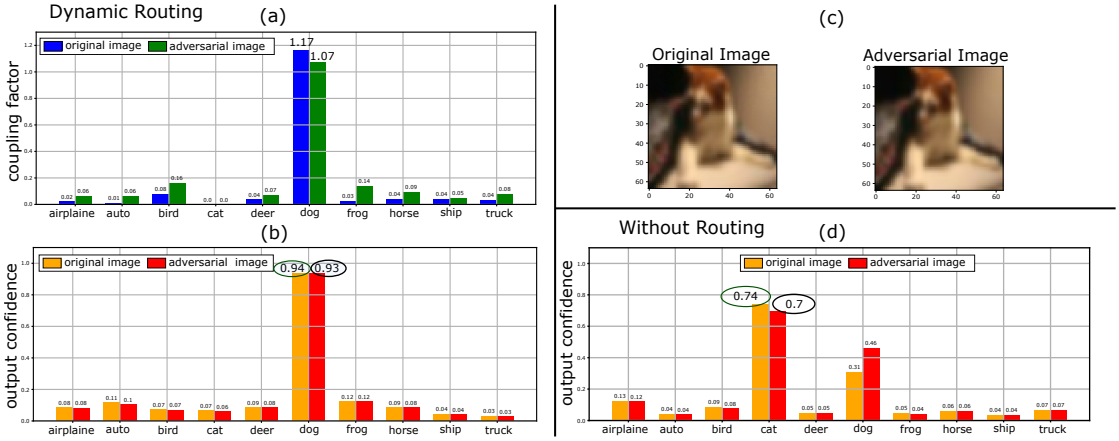

Figure 6.6: Comparison between the output activations of the DeepCaps with and without DR, by providing in input an adversarial image of the CIFAR10 dataset. Left side: (a) the coupling factors (b) the output activations of the DeepCaps architecture with DR. Right side: (c) the input image used for the comparison, (d) the output activations of the DeepCaps without DR

6.3 CW Adversarial Attack

The table 6.2 shows the results of the CW attack. Since the fooling rate is lower and the mean distortion is higher without DR, we can derive that the DR does not improve the robustness against such attack. It confirms the result that the DR does not contribute to the generalization much. The SR, instead, works better than the others, contrary to what happens for both affine transformations and PGD, showing lower fooling rate values.

Table 6.2: Robustness results against the CW attack.

	MNIST		GTSRB		CIFAR10	
Network	Mean Distortion	Fooling Rate	Mean Distortion	Fooling Rate	Mean Distortion	Fooling Rate
DeepCaps	1.24	86.8%	1.16	98.8%	0.34	100%
DeepCaps without DR	1.62	74.0%	1.27	84.11%	0.46	100%
DeepCaps with SR	**2.28**	**48.6%**	1.02	**54.4%**	**0.52**	**99.2%**

6.4 Conclusions

With our analysis we compared three DeepCaps architectures by modifing or removing the routing algorithm between two capsule layers.

- From the first analysis *against affine transformations*, we derived that the architecture without DR responds slightly better than the others. Accordingly to our results, the architecture with the SR in the last layer outperforms the model without DR only in the case of the simpler MNIST dataset.
- *Against the PGD attack*, only for the simple MNIST, the DR seems to improve the robustness, increasing the magnitude of the perturbation. The DeepCaps with SR outperforms all the others, but when already the accuracies are too low. Overall, going toward more complex datasets, the DR does not improve the classification capability, especially for high level of perturbation.
- *Against the CW*, the SR seems to confer robustness, even if, again, the architecture with DR is outperformed also by the one without DR.

To go into detail, we also analyzed the behaviour of the coupling coefficients and the output activations for the architectures with and whitout DR, concluding that when the input is perturbed or transformed, the coupling coefficients do not help, or in some cases could make the things worst. This could happens because, when the input image becomes too noisy, the transformation matrix could wrongly recognize some relationships between the inputs and a wrong output label, which the DR amplifies, together with the correct agreements. This, with the consequence of increasing the confidence of the incorrect label.
We can conclude that the relationship between objects, is learned by the transformation matrix during the training, and performing more routing iterations during inference does not help to recognize spatial input transformations or brittle features. Furthermore, it is possible to learn the coupling coefficients implicitly, considering them included in the weights of the transformation matrix during the training process.

Chapter 7

Conclusions and Future works

In this work, we systematically analyzed the contributions of the CapsNets to their robustness against affine transformations and adversarial attacks. Comparing different types of CapsNets and CNNs, we investigated which model quality plays an important role for our purpose. The ShallowCaps are more robust than comparable CNNs, as already confirmed by other works [8][34]. However, not so deep to correctly generalize more complex datasets, with the further cost to train a high number of parameters. The results prove that they are more robust against adversarial attacks but show their limits against affine transformations. Going deeper, the DeepCaps reduce this problem, decreasing the gap between the transformed and untransformed version of the datasets, despite the lower number of parameters. Against the adversarial attacks, specifically the PGD, the DeepCaps does not reach the same robustness of ShallowCaps for a simple task like the MNIST or GTSRB classification, but for a more complex dataset like the CIFAR10, their performances overcome not only a CNN with a similar architecture, but also the ResNet20. The same conclusion can be done on affine transformations, where the DeepCaps reaches a higher accuracy than the ResNet20 with the affCIFAR dataset. Moreover, our results show that the dynamic routing does not contribute much for improving the CapsNets robustness.

Additionally, accordingly to our results, it turns out that by applying defenses with a spercific attacker algorithm, i.e the Projected Gradient Descend (PGD), the robustness of the models increases also against a different attack, i.e. the Carlini Wagner (CW).

Regarding future works, could be interesting to exploit the adversarial examples generated to fool a more robust model, i.e. a Capsule Network, to train less robust Neural Networks, i.e. ResNet20, analysing whether its robustness improves.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 4, 12).

[2] Peter Jeffcock. «What's the Difference Between AI, Machine Learning, and Deep Learning?» In: (2018). URL: https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning (cit. on p. 4).

[3] *Introduction to Deep Learning: What do I need to know...?* 2018. URL: https://medium.com/@srnghn/introduction-to-deep-learning-what-do-i-need-to-know-75794ebc4a62 (cit. on p. 6).

[4] Yann LeCun and Corinna Cortes. «MNIST handwritten digit database». In: (2010). URL: http://yann.lecun.com/exdb/mnist/ (cit. on p. 7).

[5] Jan Kukacka, Vladimir Golkov, and Daniel Cremers. «Regularization for Deep Learning: A Taxonomy». In: *ArXiv* abs/1710.10686 (2018) (cit. on p. 10).

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: (2014) (cit. on p. 11).

[7] A. Mikołajczyk and M. Grochowski. «Data augmentation for improving deep learning in image classification problem». In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. May 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338 (cit. on p. 11).

[8] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. «Dynamic Routing Between Capsules». In: *CoRR* abs/1710.09829 (2017) (cit. on pp. 13–15, 19–23, 26, 27, 34, 35, 38, 58).

[9] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. *Intriguing properties of neural networks*. 2013 (cit. on p. 17).

[10] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. «Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples». In: *CoRR* abs/1602.02697 (2016) (cit. on p. 17).

[11] *Capsule Network (CapsNets -Tutorial*. 2018. URL: https://youtu.be/pPN8d0E3900 (cit. on p. 21).

[12] Amara Dinesh Kumar. «Novel Deep Learning Model for Traffic Sign Detection Using Capsule Networks». In: *CoRR* abs/1805.04424 (2018) (cit. on p. 23).

[13] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. «Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark». In: Aug. 2013. DOI: 10.1109/IJCNN.2013.6706807 (cit. on p. 23).

[14] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. «DeepCaps: Going Deeper With Capsule Networks». In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019 (cit. on pp. 23, 25, 32, 34, 35).

[15] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. «Matrix capsules with EM routing». In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=HJWLfGWRb (cit. on p. 26).

[16] Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Soufei Zhang, and Zhou Zhao. «Investigating Capsule Networks with Dynamic Routing for Text Classification». In: *CoRR* abs/1804.00538 (2018). arXiv: 1804.00538. URL: http://arxiv.org/abs/1804.00538 (cit. on p. 26).

[17] Jaewoong Choi, Hyun Seo, Suii Im, and Myungju Kang. «Attention routing between capsules». In: *CoRR* abs/1907.01750 (2019). arXiv: 1907.01750. URL: http://arxiv.org/abs/1907.01750 (cit. on p. 26).

[18] Hongyang Li, Xiaoyang Guo, Bo Dai, Wanli Ouyang, and Xiaogang Wang. «Neural Network Encapsulation». In: *CoRR* abs/1808.03749 (2018). arXiv: 1808.03749. URL: http://arxiv.org/abs/1808.03749 (cit. on p. 26).

[19] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim. «Self-Routing Capsule Networks». In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 7658–7667 (cit. on p. 26).

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386 (cit. on p. 27).

[21] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV] (cit. on p. 27).

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *CoRR* abs/1512.03385 (2015) (cit. on pp. 27, 28, 32, 34).

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014 (cit. on pp. 28–31).

[24] Nicholas Carlini and David A. Wagner. «Towards Evaluating the Robustness of Neural Networks». In: *CoRR* abs/1608.04644 (2016) (cit. on pp. 28, 29, 45).

[25] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2017 (cit. on pp. 28, 31, 46).

[26] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. «Adversarial examples in the physical world». In: *CoRR* abs/1607.02533 (2016) (cit. on p. 28).

[27] Nicholas Frosst, Sara Sabour, and Geoffrey E. Hinton. «DARCCC: Detecting Adversaries by Reconstruction from Class Conditional Capsules». In: *CoRR* abs/1811.06969 (2018) (cit. on p. 31).

[28] David Peer, Sebastian Stabinger, and Antonio Jose Rodrıguez-Sánchez. «Training Deep Capsule Networks». In: *CoRR* abs/1812.09707 (2018) (cit. on p. 31).

[29] Alberto Marchisio, Giorgio Nanfa, Faiq Khalid, Muhammad Abdullah Hanif, Maurizio Martina, and Muhammad Shafique. «CapsAttacks: Robust and Imperceptible Adversarial Attacks on Capsule Networks». In: *CoRR* abs/1901.09878 (2019) (cit. on p. 31).

[30] Felix Michels, Tobias Uelwer, Eric Upschulte, and Stefan Harmeling. «On the Vulnerability of Capsule Networks to Adversarial Attacks». In: *CoRR* abs/1906.03612 (2019) (cit. on pp. 31, 32).

[31] Nicolas Papernot et al. «Technical Report on the CleverHans v2.1.0 Adversarial Examples Library». In: *arXiv preprint arXiv:1610.00768* (2018) (cit. on p. 35).

[32] François Chollet et al. *Keras*. 2015 (cit. on p. 35).

[33] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015 (cit. on p. 35).

[34] Jindong Gu and Volker Tresp. *Improving the Robustness of Capsule Networks to Image Affine Transformations*. 2019 (cit. on p. 58).