UNIVERSITÀ DEGLI STUDI DI TORINO
Dipartimento di Informatica

Corso di Laurea in Informatica

Tesi di Laurea Magistrale in Informatica

A computational model for word learning on real world data through Deep Neural Networks.

Curriculum: Realtà Virtuale e Multimedialità

Relatore:
Valentina Gliozzi
Roberto Esposito

Candidato:
Giorgia Fenoglio

ANNO ACCADEMICO 2015/2016

Ringraziamenti

Al termine di questo percorso universitario e della stesura di questa tesi desidero fermarmi a ringraziare tutti coloro che, a vario titolo, hanno reso possibile tutto questo.

In primo luogo un sentito grazie ai relatori Valentina Gliozzi e Roberto Esposito, per avermi guidato in questo tortuoso percorso, lasciandomi libera di compiere le mie scelte e fornendomi tutto l'aiuto di cui avevo bisogno.

Un doveroso grazie ai professori della magistrale per avermi interessato, stimolato, illuminato e aver fatto crescere in me la passione per questa meravigliosa scienza.

Un grazie immenso va ai miei genitori e a mio fratello, perchè mi avete sostenuto in ogni momento di questo percorso universitario, discutendo con me ogni problema e festeggiando ogni successo. Grazie perché avete creduto in me in questi anni senza mai farmi mancare il vostro appoggio, senza il quale non sarei dove sono oggi.

Grazie davvero ai miei coinquilini Angela, Simone, Davide, Serena e Carla: siete stati la mia seconda famiglia, ridendo con me nei momenti migliori e soprattutto in quelli peggiori, condividendo le disgrazie universitarie e i dubbi verso il futuro con la spensieratezza che ci contraddistingue.

Un grazie di cuore a Leonardo, che ha affrontato con me ogni istante di questo di percorso, ascoltandomi e consigliandomi in ogni momento: grazie perchè sei la mia roccia e il mio trampolino.

Grazie a Debora perchè, che ci separi un muretto o qualche chilometro, nessuna carente conoscenza dell'informatica ti ha mai fermato dall'essere la mia spalla.

Un doveroso grazie ai miei compagni di università, in particolare ad Andrea (il valdostano), Aureliano e Andrea (Perets), perchè anche se in certi momenti mi avete fatto dannare, siete stati la migliore compagnia in cui potevo sperare in questi due anni.

Un ultimo grazie alla famiglia e agli amici: ciascuno di voi ha dato il proprio contributo per rendermi ciò che sono oggi, quindi questo traguardo è anche merito vostro.

Contents

Chapter 1

Introduction

The purpose of this thesis is the simulation of the process of infant word learning, through a system based on neural networks.

Word learning is the process through which the infants learn the words and their meanings.

In particular, children, in the first years of their life, learn to segment the visual field and the auditory stream in the parts that compose them (the objects and the words), to assign each part to a specific category and to create an association between the object categories and the word categories. Through this process, infants acquire new vocabulary, learning the words and their meanings.

At the end of this process, when an infant view an object he can retrieve the word associated to the category of the object and, at the same time, when he hears a word he can retrieve a representation of the referred object.

The problem of lexical acquisition view from a developmental and psychological point of view and the findings about infants' object knowledge are fully described in the first part of this thesis.

The core of the thesis is the creation of the model that simulates the child cognitive process, therefore it is composed of several neural networks that are a useful means to simulate a cognitive process.

The model is based on that proposed by Mayor and Plunkett [14]: two Self Organizing Maps are used in order to create the correct associations between objects and their names, when they are presented as inputs to the two maps; that model uses inputs artificially created, which limits its application in real situations, in particular for what concerns the visual part.

The model developed in this thesis extends the one by Mayor and Plunkett [14], introducing the use of real images: after the evaluation of the state of the art deep neural networks developed in the recent years as tool for the image recognition and classification, we focused on the Deep Convolutional Neural Networks which are the most capable, among all the studied Neural Network, to extract a distributed representation of the image provided as input.

In particular, we use the Deep Convolutional Neural Network called InceptionNet presented by Szegedy et al. [24], which is the state of art in the task of image recognition and classification on real data images, in order to extract abstract representations that catch the most important features of the input images.

The extracted representations have been studied in order to understand their contents and their properties, through different experiments among which the most important are the clustering experiments.

We defined a concept of *goodness of the representations*, which allows us to evaluate the representations, comparing the results achieved with InceptionNet with the results achieved using the reference Deep Convolutional Neural Network (AlexNet).

The extracted representations have been used to create the training set for training a Self Organizing Map (also called SOM): this type of network produces a low-dimensional discretized representation of the input space of the training samples, organizing the inputs in a topological way.

The Self Organizing Map learns to identify in its different areas the different available categories of objects (assuming that the representations provided as inputs are good enough), recognizing similarities and differences between the presented inputs. At the end of training, the SOM is able to categorize new representations provided as inputs, on the basis of the most activated area.

It can be argued that the Deep Convolutional Networks together with the SOMs work as the infants' neural networks, which recognize an object in the visual field as belonging to a specific category if, after the correct processing, it activates a specific area of the neurons. This happens whatever the point of view on the object is and whatever its irrelevant features (for example color and size) are. InceptionNet is able to create good representations that are almost uniform for all the examples in a specific category, without being distracted by the point of view or by irrelevant features; then the Self Organizing Maps is able to categorize these representations into different areas on the basis of the similarities recognized inside of each category.

On the other side, for what concern the auditory inputs, they are still artificially created and they are used as training set for an auditory SOM. This SOM, as the previous one, learns to categorize the presented inputs in different categories on the basis of the produced activation. This part simulates the segmentation of the speech stream and the identification of the words done by the infants.

The last step of the word learning process is the creation of an association between the word and its meaning: this process is simulated creating connections between the visual and the auditory SOM already trained, linking areas that are related to the same category with strong connections. In particular, similarly to what happens during the childhood learning, two stimuli (visual and auditory) are presented together to the model, which processes them up to the SOMs and then update the connections between the two maps on the basis of produced activations. This process simulates what happens when an infant looks at an object and hears its name.

At the end, the performances of the model are evaluated with different methods, considering the capacity of the system of recover an information (visual or auditory) when the other is presented as input.

Chapter 2

The problem of lexical acquisition

2.1 Constraints for lexical acquisition

In order to provide solid psychological bases to our model that simulates the word learning process, we present in this first chapter studies by different authors that try to explain how this complex but automatic mechanism occurs.

Children acquire the vocabulary of natural language, learning words and their meanings, in the early years of their lives.

This learning process happens at remarkable speed: studies by Dromi [3] show that children acquire new vocabulary at the rate of 45 words a week from about 18 months on.

How can they correctly learn at this speed?

One of the most important problems of this learning is pointed out by Quine [18], who introduced a philosophical conundrum which is usually called the *indeterminacy of reference*: he suggests that for any set of data perceived by an infant, there will be an infinite number of logically possible hypothesis that are consistent with it.

For example, suppose a child looking at a dog: if he hears someone label it as "dog", what association word-meaning can he make? The infant could think that the label refers to a feature of the dog (its color, shape, size, substance or other features), to one of its parts (tail, head or other) or to a specific breed (Labrador) or individual (Fido).

We suppose that a child cannot approach the problem by considering and ruling out every logically possible hypothesis, because this approach would be too complex. Then we need to understand how children find the correct hypothesis so fast.

The solution proposed by Quine [18] is that humans consider only some of these hypothesis, giving them priority over others. In order to set these priorities, and therefore to narrow the hypothesis space, they use linguistic constraints.

Markman [13] suggests that three constraints have proved particularly influential in informing cognitive approaches to lexical development:

Whole object constraint: with this constraint Markman supposes that children initially assume that a novel label refers to the whole object and not to its parts or to its properties. Using again the example of the dog, infants associate the term "dog" to the whole dog, not to its color, its shape, its tail or its head.

Furthermore, once infants associate a term to its referred object, they can reuse this information when they see again the whole object or one of its parts. For example if they learn the meaning of "dog" they can identify a dog even if they see only its head or its tail when the dog is occluded by another object.

Taxonomic constraint: once infants decide that the label is associated with the whole object, they need to decide how to extend that term to other objects. Considering the term "dog", children could associate that term to a specific individual or breed, or to all dogs.

As solution of this problem, Markman propose the taxonomic constraint: labels refer to objects of the same kind (taxonomically related) rather than to objects that are thematically related. Thematic relations are often used by young children to categorize objects, for example organizing them in groups that represent temporal, spatial, causal or other relations among objects (see Markman [12]). Despite this, single nouns rarely encode thematic relations and when infants learn new words they don't know yet enough language to define those new words using relations with other words. For these reasons Markman suppose that infants extend the term to objects that are taxonomically related.

Mutual exclusivity: as we know, infants can learn terms referred to features or parts of objects, but looking at previous constraints this seems impossible: suppose a child looking at a dog, if he hears the term "dog" and then the term "head" he could associate both terms to the whole object.

Mutual exclusivity can help remedy this limitation, assuming that children expect each object to have only one label.

This constraint implies that if children have already found a label for an object, new labels can't be associated with that object (it narrow the hypothesis space); furthermore it motivates children to acquire terms other than object labels, such as those concerning features or parts of object, as the term "head" in our example.

If we assume that these three constraints govern lexical acquisition, we can ask ourself if they are an emergent property of linguistic and cognitive development or they are innate.

In order to answer this question we explore studies by Plunkett [17], who attempt to demonstrate that these constraints can emerge from associative learning processes, suggesting possible models that bring out these constraints.

For each constraint Plunkett suggests a possible solution: for the whole object constraint he suggests to use an auto-associator, a neural network which is able to bring out the significant features that are shared between the images of a certain category presented as input. For the taxonomic constraint Plunkett suggests the use of an auto-encoder or of a Self Organizing Maps: the first is a neural network which reproduce at the output the pattern presented as input, the second is a neural network able to categorize the inputs on the basis of their similarities; both the nets, when are trained with enough examples, are able to give as output the same informations both for the examples of a certain category and for the examples in a taxonomically related category, bringing out the taxonomic constraint. Instead, for the mutual exclusivity, Plunkett presents experiments made with infants, without suggesting a possible implementation.

2.2 Developmental findings in object representation

In this thesis, we are interested in building a model that, simulating the word learning process, brings out the whole object constraint. For this reason we faced the problem of how objects are represented and detected by infants.

Many studies has been done in order to explain how infants can build a conceptual representation of what they see, in particular of objects that they can detect from the visual field.

Cognitive science studies, in particular, give great support in understanding how the human mind works. Two possible views of human nature dominate this science: on one view, the human mind is a flexible and adaptable mechanism for discovering regularities in experience, with a single learning system that copes with all the diversity of life. On the competing view, it is a collection of special-purpose mechanisms, each shaped by evolution to perform a specific function.

These theories explain human behaviour in divergent but consistent way, therefore it seems impossible to find a single correct solution.

Developmental science, born in order to solve this and related problems, has shown that both these views are false: in fact, human mind has neither a single learning system, nor a multitude of small special-purpose systems, but humans are endowed with a small number of separable systems of core knowledge.

Studies of human infants and animals provide evidence for five core knowledge systems

(Spelke and Kinzler [22]). Each of these core knowledge individuates the entities in its domain using a set of principles, used also to support inferences about the entities behaviour. Each system, moreover, is identifiable across tasks, ages, species and human cultures thanks to a set of signature limits that characterizes it.

These core systems are believed to be universals and constants over human development.

Core knowledge systems are used to represent inanimate objects and their mechanical interactions, agents and their goal-directed actions, sets and their numerical relationships, places in the spatial layout and their geometric relationships, and social interactions.

These systems are not set in stone: these foundations can be overcome acquiring knowledge and experience over the life. For example preschool children change their idea of numbers when they learn to count, and their idea of agents when they learn about biological processes like breathing and eating. In the same way, the system under social interactions can lead to harmful conflicts, but these behaviours can be moderate by intergroup contacts during the development.

In this thesis, where we try to bring out the whole object constraint from our model, we are interested mainly in the representation of objects.

The core system of objects representation, which aims to identify and represent inanimate objects, uses the principles of cohesion (objects are connected and bounded wholes), continuity (objects move on continuous paths), and contact (objects interact when they make contact) in order to detect objects out of a visual stimulus deriving from a visual scene.

These principles allow infants to identify objects and their interactions, even when they are fully or partially occluded, and to predict where and when an object will move.

This core system has an important signature limit: the number of objects that infants are able to represent at time (about three).

Understanding inanimate objects representation is very useful in word learning process where we try to associate an object representation with a word.

2.3 Object knowledge

In the previous we gave an idea of the complexity of lexical acquisition and of the constraints that lead this process.

From now we focus on the whole object constraint, in order to implement a neural network that reproduces the behaviour of this constraint.

The first problem that we encounter is to understand the kind of knowledge that children acquire about objects during the development.

How can they understand what is an object? How can they distinguish that two adjacent objects are not a single object? How can they understand that an object exists even when it is occluded by another object?

In section 2.2 we showed principles underlying the inanimate object representation that can help to answer some of these questions.

We present now some results both psychological and computational about how children represent an object and about their capacity of understanding that an object exists and of predict its position even when it isn't visible.

We start talking about results reached by experiments on infants and then we present some models that try to replicate these results.

2.3.1 Psychological findings

The first theory about how infants can understand hidden objects has been proposed by Piaget, who, in the early 1920s, supposed that this knowledge develops through six stages that span the infancy, until the age of two.

This idea was enforced by experiments in which children are required to retrieve known objects moving behind other objects.

These experiments depend on the infant's object knowledge, but depend also on the motor maturity of the infant and on his ability of planning actions to achieve the task.

In the following years, other psychologists tried to design new experiments in which the object knowledge is recognizable independently from the motor maturity of the infant.

The most important studies rely on the finding that infants direct more attention (look longer) at novel or unexpected stimuli. This finding can be used in order to understand what object knowledge is built in their minds: supposing that they attend to an event from which they can learn some object knowledge, if they look longer at a second event with respect to a third one this means that that second event is unexpected, indeed we can deduce that they built some object knowledge that agrees more with the third event (the one that does not surprise them) than with the second one.

This approach is at the origin of the following methods:

Habituation-dishabituation: infants attend to an event involving an object, then the object is occluded. After this they attend to two other events, which are related to the original, but differ along some object-relevant dimension. One of this event is consistent with the idea that the original object exist even if it is occluded, the other is consistent with the opposite idea. Then psychologists study how infants transfer what they learn in the original event to new events, in order to understand what they believe about hidden objects.

Violation-of-expectation: infants attend to a familiar event, for example a ball is moved behind a screen. Then they attend a possible event (when the screen is removed the ball is still here) and an impossible one (when the screen is removed the ball is missing) related with the familiar one. Infants look longer the unexpected event, which is the impossible one, and this behaviour allows psychologists to understand what infants know about hidden objects.

With these methods, we can understand infants' object knowledge independently from their motor maturity.

Studies demonstrate that 3.5 months infants understand that hidden objects continue to exist and that they maintain their property of solidity.

Under the age of 8 months they understand the height and location of a hidden object, and at 13 months they can use a protuberance to infer the size of an object under a cloth.

For what concerns the number of hidden objects, since 2.5 months infants can keep track of the number of objects that are behind a screen.

Furthermore, studies show that from 3.5 to 8 months the infant's ability of comparing the size of objects and occluding screen matures, allowing him to understand that if the screen is smaller than the object but he can't see the object something is wrong.

We now may ask ourselves what features infants can use to recognize an object, that is how can they represent an object in their mind? The representation grows with infant development: at 3.5 months infants can use size and shape to recognize an object, at 7.5 months they can use texture pattern information, and only at 11.5 months they can use colour information to individuate an object.

2.3.2 Computational model

Building a computational model is very useful, because researchers are forced to be explicit about the nature, scope and power of the knowledge representations that they posit, and about the processes that work over those representations.

We now show some models that try to explain infants behaviour about hidden objects, in order to understand how these models build an object knowledge representation.

These models work with a simplified input that represents the position of an object and a screen in a specific moment. Consecutive inputs show different positions of the object simulating its motion in the scene and behind the screen. The task is to predict the next position of the object, even when it is occluded by the screen, predicting at the same time, the time and position of the reappearance of the object at the end of the screen.

Rule-based model

Early models, created to explain infants behaviour about hidden objects, are rule-based models: researchers write a set of rules that explain infant behaviour when they interact with hidden objects. These models don't provide a mechanistic account of development, but they propose different sets of rules in order to explain behaviour at different ages. Rules are written statically by researchers and there aren't way to dynamically create a new set of rules starting from another, or to transform a set into another in order to catch different behaviour.

These models are based on the Bower's Identity Theory of object, which gives conceptual rules used to understand what is and what isn't an object. Conceptual rules are used by researchers as a foundation to build the models and can represent the object knowledge.

Attention-based model

Another kind of model is the attention-based model, in which each object is associated with an index, a mental token that works like a pointer. Indexes are in a limited number, are distinct for each object and they stay with the object over spatial transformation.

This model reproduces some infant behaviour, but it fails to implement any account of how development might occur. Indexes can contain some information about the object knowledge.

Connectionist models

As difference with respect to the previous models, connectionist models try to predict the reappearance of the object using a multi layer network.

The first connectionist model we present has been proposed by Munakata et al. [15]. Its architecture is made by three layers: the input layer, the hidden layer and the output layer. This model takes as input a stimulus containing 7 units with object informations and 7 with screen informations which can be the location of the object and screen (each unit represent an allowed position), or some features bounded to them. The layer of hidden units represents the object knowledge of the network at a specific time.

This net is trained using backpropagation and tries to predict the next input, that is the next position of the object.

This model can also explain the lag between infants' ability to predict the location of a hidden object and their ability to respond on that knowledge (for example to take the object): indeed the output layer can be duplicated in order to represent these two abilities. Then a different learning rate (lower for the second ability) is used in the learning phase and the training of the second output layer is delayed, waiting that the first has partially learned its task.

In this way the first output layer learns to predict the next position of the object and after a while the second output shows the ability to respond on that knowledge.

Another connectionist model proposed by Mareschal et al. [11] tries to separate the prediction about the trajectory of the object from the object knowledge, represented in the hidden layer in the previous network. A grid with four feature detector units per cell is used as input of the network; this grid is called *input retina*.

This architecture is built of three main modules:

object recognition module: the object recognition module is made of five complex cells, associated with a particular feature combination wherever it appears on the retina; it is used in order to recognize the object on the retina and contains the object representation.

trajectory prediction module: the trajectory prediction module uses a layer of hidden units to predict the next position of the object; it uses informations acquired with previous inputs, associating to the object its motion informations, visible near the object as the tail of a comet. This module is used in order to accomplish involuntary behaviour, like surprise as effect of a wrong prediction.

response integration module: this last module combines informations extract from the two previous modules in order to show the infant's ability to co-ordinate and to use informations that it has about object position and object identity in order to accomplish voluntary task. When the object is visible this module uses mostly object recognition module informations; when it's hidden it uses the informations contained in the trajectory prediction module.

The two models showed so far try to explain some infants' behaviours and their knowledge about hidden objects, and they demonstrate that an internal representation of hidden objects independent from any spatial transformation can be acquired through learning; despite this, they arise a lot of other challenges, for example understand how the models cope with being extended to the multitude of tasks in which infants show hidden objects knowledge.

Furthermore these models work with simplified input: the input retina used in the last model, for example, is a grid with cells set to 0 to represent the background, to 1 to represent the screen and to some value between 0 and 1 to represent the object.

This last point is a big limit of all these models, especially when we try to build a realistic model.

In the next chapter we show some models that try to build a conceptual representation of the object, independent from spatial transformations, starting from a more realistic input.

Chapter 3

Object representation

3.1 Good object representation

In this chapter we introduce several models that create a conceptual object representation.

What can be defined as an admissible representation?

First of all, the representation needs to be highly similar for each input that represents the same object and each different object needs to be associated with a different representation; for example, if we give as inputs some dog pictures we expect that the model produces as output a single representation (or highly similar representations) for the object dog. This output has to be different from the one created using pictures of other objects as for instance tables or trees.

At the same time we can expect that different modifications of the same object, like different positions and sizes have to lead to the same representation (or highly similar representation). Furthermore, even if the object is not completely visible, for example we see the head and body of a dog but not its tail, the model has to output the whole object representation as before.

Moreover, the conceptual object representation needs to be independent from spatial transformation: rotation, scaling and translation of the picture. For example, if we take a dog photo and we rotate it of 180 degrees we expect the representation produced by the model to be the same.

Models that we show in next sections use deep neural networks. This technology is of particular interest because it has been showed to cope well with the complexity of realistic inputs in which the object can have different positions, sizes, hidden parts, colors, luminosities, locations in the photo and in which the background can divert the object identification process.

For these reasons models sometimes try to solve problems with multiple steps, first detecting the object in the image, then cutting out the related part and finally identifying the object.

We will show, in the following sections, many models that we can use to extract a conceptual representation; the target of these models is usually image classification, based on the main object contained in the image. Classification, usually, passes through the building of a representation for the main object.

We can divide models into two categories: cognitively plausible models, that try to explain architecture and behaviour of the model in a cognitive way, and models that are not plausible for cognition.

We list some interesting models, starting from the cognitive ones, identifying them with name and authors.

3.2 Rolls et al., VisNet

One of the firsts studies about object recognition in a cognitive plausible way is the one of Wallis and Rolls [26].

In particular the paper is based on neurophysiological studies of single neuron activity in primates. These studies give evidence on how information about visual stimuli is represented in temporal cortical visual areas, and how representations that are invariant with respect to the position, size and even view of objects are formed.

We give a brief sketch about the neurophysiology of the temporal cortical visual areas, in order to understand the plausibility of the model presented.

Visual pathways are extend through a number of cortical stages from the primary visual cortex until they reach the temporal lobe visual cortical areas. In each stage neurons are specialized to respond to specific stimuli: for example first levels respond to simple stimuli like bars, edges or angles, other levels respond to moving visual stimuli. Usually each level combines informations from previous levels in order to give new informations.

In temporal cortical area we can find neurons responsive primarily to faces, giving also informations about which face has been seen.

An important question for understanding brain computation is whether a particular object (or face) is represented in the brain by the firing of a single (or a few) cell, or whether instead the firing of a group of cells, each with somewhat different responsiveness, provides the representation.

Studies in this direction by Rolls and Tovee [20] show that the representation for an object is distributed on an ensemble of cells in a sparse way: distributed representations help to maximize the number of different memories that can be stored, and sparse distribution gives advantages as generalization to similar stimuli, fault tolerance (graceful degradation) and some locality to the representation.

Another problem which must be solved is the building of a representation which allows recognition to occur independently from size, contrast, spatial frequency, position on the retina, angle of view or other object variations.

This is possible because a subpart of neurons respond to a stimulus independently from these variations (Rolls [19] shows this property in experiments on monkeys); these neurons are responsible for the object (or face) representation. Other neurons react in different ways based on the variation with which the object is proposed; these neurons are useful to discriminate facial expressions and help in social interactions.

Furthermore visual cortex gets as input not the entire image, but the subpart catch by the fovea, that is central region of the retina with the maximum visual acuity; in this way it eliminates a lot of informations that could lead to a wrong representation.

At the end of this description of visual cortex neurophysiology we focus on how and how fast neurons can learn a new representation.

When a new face is proposed multiple times, neuron activations are updated in order to store a representation of the new face. Older representations (of previously seen faces) are not disrupted when the new representation becomes stable. Neurons need few presentations of the new face to update their activations in order to create a new stable representation. This remarkable speed characterize each cortical stage involved in object recognition.

VisNet, a computational mechanism that simulates the cortex work presented so far, has been proposed by Wallis and Rolls [26]: it's a four-layer network, designed as a sequence of hierarchical, convergent, competitive networks (Self Organising Maps [1]).

Like the visual cortex, it is organized as a set of hierarchically connected cortical regions, with convergence from each small part of one region to the succeeding region (or layer in the hierarchy).

Each layer is considered to act partly as a set of local self-organizing competitive neuronal networks (SOM) with overlapping inputs, where synaptic modifications follow a modified Hebb rule: the activation of the output is proportional to the activations of previous SOM that works as input.

Each layer operates to detect correlations between the activity of the input neurons, get from the previous layer, and to allocate output neurons to respond to each cluster of such correlated inputs. Competitive scheme result in a small ensemble of active neurons in a defined region representing a category (or cluster of correlations) recognized in the input.

[1]For details on Self Organising Maps see chapter 7

The distributed representation (as said before) have utility in maximizing the number of memories that can be stored.

Each level of the network is trained in hierarchical order, from the bottom; activations in each level work as input for the next level.

In order to create a net that can recognize objects independently from variation to the input image (size, position, rotation...), in the training phase different views of the same object are shown in succession.

The net is tested with simple stimuli, like T, L and + stimuli, in order to test its invariant capacity, as well as on face images. In both cases the network obtained satisfying performances.

The net presented is one of the most important of the cognitively plausible networks of object recognition.

We spend the rest of the chapter showing other models, which have better performances often at the price of cognitive plausibility.

3.3 Lee et al., Sparse deep belief net model for visual area V2

The focus of the work presented by Lee et al. [9] is an unsupervised learning model that faithfully mimics certain properties of visual area V2; in particular they develop a sparse deep belief network whose first layer simulates the work done by V1 cell receptive fields, and second layer encodes correlation of the first layer responses as done by visual area V2.

Area V1 neurons show a selectivity for oriented bar, working as localized oriented edge filters. Results of area V1 are well known, so it's easy to measure the efficacy and efficiency of the proposed algorithm.

Instead, quantitative accounts of responses in area V2 are few in numbers: in the literature we can identify two sets of quantitative data that give us a good starting point for making measurements to determine whether an algorithm may be computing similar functions as area V2. The first one describes how V2 neurons respond to angular stimuli, with no difference for angle width or orientation; the second one describes how they respond to complex contour and reticular stimuli.

The algorithm proposed is based on the Lee et al. [9] algorithm for deep belief networks learning: each layer is treated as a Restricted Boltzmann Machine (RBM) and the network is trained one layer at a time from the bottom. The algorithm is modified in order to learn sparse representations.

The result is a two layers network that simulates V1 and V2 areas: when weights are displayed we can see V1 filters for oriented bar recognition and V2 filters for edge and contour recognition.

When layers are tested with standard images (specific set of angles and complex shaped stimuli) their neurons respond as described in studies on real V1 and V2 area.

Given the uncertainty about how following levels of the visual cortex work and the complexity of handling deep net made by RBM, it's difficult to extend this model in order to handle the entire process of visual analysis.

3.4 Lee et al., CDBN for scalable unsupervised learning of hierarchical representations

This second work extends the previous: the target is the building of a hierarchical generative model, called Convolutional Deep Belief Network, that can cope with the complexity of high-dimensional images.

Lee et al. [10] present a deep architecture consisting in feature detector units arranged in layers: lower layers detect simple features (as bars in the previous work) and feed into higher layers, which in turn detect more complex features. In particular, in the architecture proposed, the first layer learns edge detectors, the second layer learns object parts and the third layer learns the whole object.

Each layer of the net is a CRBM (Convolutional Restricted Boltzmann Machine) which is made of three levels: the first one is the input layer, also called the visible layer; the second one is the hidden layer, also called the detection layer, where hidden units are gather into groups that are fully connected with the visible layer so that these weights are shared among all locations in the image; the third one is a max pooling layer.

The pooling layer shrinks the representation of the previous layer by a constant factor, allowing higher-layer representations to be invariant to small variations of the input and reducing the computational burden. More specifically each unit in a pooling layer computes the maximum activation of the units in a small region of the detection layer.

The final network is a three levels network, each one is a CRBM made by three levels (the pooling layer at one level is the visible layer in the consecutive level).

This net can be used for classification: in order to classify an image we can use the net to get a representation of the object (seen as concatenation of the first and second pooling layer activations) and then use this representation as input for a SVM classifier. Performances are comparable to the state-of-the-art.

An important advantage of this net is that it can learn object parts in an unsupervised way: each level learn to combine informations of the previous level without any supervision, that is without any label. In particular, when the net is learned with a single object category we can see how the second layer learns features corresponding to object parts and the third learns to combine these parts representations into more complex higher-level features.

Furthermore, once the net knowns an object and its representation, if we give a corrupted input (for example an incomplete one) the net can create the representation associated with the original (complete) input, based on what it learned about that object on the training phase.

The only problem is that this model requires a certain degree of supervision during dataset construction: images used for training must be aligned, homogeneous and belong to one selected category.

3.5 Le et al., Deep autoencoder to build high-level features in unsupervised learning

The idea to use deep networks built by staking simple few-level networks, used in the previous work, is the base of this work.

The target is, again, building high-level feature detectors from unlabeled data.

The solution proposed by Le et al. [8] is a 3-layered locally connected sparse autoencoder with pooling, with which they trained a face detector. In particular each layer is made by an input layer, a local filtering layer (the hidden level), a local pooling layer and a local contrast normalization layer.

Unlike the model proposed by Lee et al. [10] in this model there is no convolution: weights aren't shared across different locations in the image, but hidden, pooling and normalization layers work locally, selecting a small part of input informations from the previous layer.

The idea is that at the last level we can consider each neuron as a feature selector associated with a specific high-level feature. In particular we can identify a neuron whose activation allows us to classify the input image.

The approach is inspired by the neuroscientific conjecture that there exist highly class-specific neurons in the human brain, selective for object categories such as faces or hands. Surprisingly, this kind of neurons, informally known as "grand-mother neurons", exists in the last layer of the proposed network: we can find a single neuron that performs very well in recognizing faces, despite the fact that no supervisory signals were given during training.

In order to understand what the grand-mother neuron catches, Le et al. build the image that maximize its activation. The resulting image is an average of faces used during training.

If training is done with faces in different position and rotation the grand-mother neuron activation is independent from these variations.

As in other works, once we have a representation of the object (the one given by the last layer) we can build a supervised classifier that uses that representation as input and that outputs the corresponding class.

3.6 Krizhevsky et al., Classification with CNN

Target of this work by Krizhevsky et al. [7] is the classification of the ImageNet dataset (1.2 million high-resolution images) into 1.000 different classes using a Deep Convolutional Neural Network[2].

The main difference from previous work is the amount of classes and images in the dataset.

By using a large dataset the network copes with very different images, learning to identify objects in very different situations.

In order to cope with the complexity of real images we need more complex networks: the proposed one has 650.000 neurons, organised in five convolutional layers (some of which are followed by max-pooling layers) and three fully connected layers with a final 1000-way softmax. Training parameters are 60 milion.

Problems related to a large CNN like the one proposed are many: first of all, in order to train a large CNN we need enormous datasets and a lot of time; even if we have a model with a large learning capacity and enough images for the training, to learn about thousands of objects from millions of images the model needs to have lots of prior knowledge. This happens because they have to cope with high resolution real images.

When we use high-resolution images (like in this case) the amount of images and time necessary for the training increases, making the problem intractable with standard feed-forward neural networks.

In these cases CNNs can be a solution: they have much fewer connections and parameters and they are easier to train; moreover current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to support the training of large CNNs.

In particular, in the model proposed by Krizhevsky et al. [7], they train one of the largest

[2]For a complete description of a Deep Convolutional Neural Network functioning we refer you to chapter 5.

convolutional neural networks achieving the best results ever reported on the ImageNet dataset.

The improvements presented in the work are due to a highly-optimized GPU implementation of 2D convolution, to the employment of REctified linear Units as the non linear operator, to parallelization of the training on multiple GPUs, to the usage of overlapping adjacent pooling units, and others.

Despite the large dataset, the net suffers of overfitting. In order to reduce it, authors present two different methods: data augmentation and dropout.

In the first one dataset is artificially enlarged using label-preserving transformations, in particular translations, horizontal reflections and generation of new images altering the intensities of the RGB channels of training images.

In the second one the output of each neuron has a 50% chance of being suppressed; with this method the net is forced to learn more robust features since it can't rely on the presence of particular neurons output.

Optimizations presented are useful to handle the large number of parameters in order to simplify training and are described in detail in section 5.5.

Result is a net that reaches the best performance on ImageNet dataset. This network is one of the networks we studied in this thesis. In facts, we will show that studying hidden levels (in particular the last one) we can find a distributed representation of objects as defined at the start of this chapter: the representation is similar for images related to the same object and it's different for images related to different objects.

3.7 Szegedy et al., DNN for Object Detection

An important problem in object recognition process, so far neglected, is the position of the object in the image.

Standard models, like those seen so far, are able to identify a single object in a complex image, when the background is nearly uniform and the object covers most of the image.

The solution proposed by Szegedy et al. [23] tries to detect an object of any size and position in the given image, building a Deep Neural Network which outputs a binary mask of the object bounding box.

The model is made by two main parts: the first one is a DCNN, inspired by the architecture defined by Krizhevsky et al. [7], in which the last layer is replaced by a regression layer. The regression layer extracts from the net spatial informations (instead of the object representation), used to lead the building of the bounding box. In particular it outputs a mask, where pixels are set to 1 if they are in the predicted bounding box of the object, 0 otherwise.

The second part is a multi-scale box inference, followed by a refinement step, with which the net can predict a more accurate mask starting from the initial mask.

We don't go into details with this architecture, while it is useful to show one of the problems that we have to face when we use real images, it is nevertheless outside our interest because it does not improve our knowledge about object representation.

3.8 Szegedy et al., DNN for Scalable Object Detection

The work proposed by Erhan et al. [4] is an extension of the previous model in order to detect multiple objects in the same image: they proposed a saliency-inspired neural network model for detection, which predicts a set of class-agnostic bounding boxes along with a single score for each box, corresponding to its likelihood of containing any object of interest.

The first step is to train a detector, called *DeepMultiBox*, which generates a small number of bounding boxes as object candidates. The important idea is that boxes are generated by a single Deep Neural Network, in a class agnostic manner. Furthermore, for each predicted box the net outputs a confidence score on how likely is the box that contains an object.

The second step is to use the sub-images extracted from the original one as input for an object classifier in order to identify the object. This step is not described in the paper, because we can use one of the existent classifier to reach the task. For experiments the classifier used is similar to the one presented by Krizhevsky et al. [7].

In order to predict bounding boxes and their confidence scores, the DNN is trained using training images with ground truth boxes: the DNN predicts for each image a large number of bounding boxes each with a confidence score; then the training algorithm updates the score for each box such that if the box matches well one of the ground truth object boxes the score is increased, but is lowered if vice versa happens.

We don't go into training algorithm details, but we report some experimental result.

The model has been tested on two different tasks: detection and classification; the second task starts from the sub-images generated in the first task and tries to classify them.

First of all, as we'll see for other models, DNN needs large training sets: in the detection task they use 110.000 images, cropped in order to obtain 10 million crops overlapping some object (positive boxes) and 20 million crops not overlapping or overlapping minimum parts of objects (negative boxes). In classification task they use 544.545 training images uniformly distributed among 1.000 classes.

In both tasks the net is competitive, reaching state of art performances. In particular, performances increase if we increase the number of boxes that we consider: specifically, the detector predicts a large number of boxes associated with a confidence score and we select the best K boxes. Increasing K we allow the network to reach more competitive performances.

This model is very interesting because it allows us to understand how detection and classification can be integrated in order to reach better results.

The successive models follow this road, trying to integrate these and other tasks.

3.9 Sermanet et al., Overfeat

As said before, the classification (and the training task) could be easier if each image contains exactly one object, keeping out unnecessary informations as background or other objects.

But if images aren't so well defined, we need a more sophisticated net as the one proposed by Sermanet et al. [21], where they integrate classification, localization and detection in order to extract and classify objects from the images.

They indicate with classification the task of identifying the object in the image (with 5 guesses to find the correct answer), with localization the task of finding a box that contains the entire main object of the image, with detection the task of finding many boxes, one for each object of the image.

They propose the net as a features extractor (named *OverFeat*), which supports a multi-scale classifier and a bounding box extractor. The model uses convolutional methods to study subparts of the images in order to identify features or objects.

The features extractor is similar to the one presented by Krizhevsky et al. [7], with some differences like that pooling regions are non-overlapping and 1st and 2nd layer feature maps are larger. Furthermore the last layer, the one intended to support classification, is removed.

For classification purposes they add bark the class layer to the OverFeat network.

For detection, they complete OverFeat with a regression network, training it to predict object bounding boxes at each spatial location and scale.

We note that since the classifier and the regression networks share the same features extraction layer they need to recompute only final layers.

Predicted boxes are combined with classification results at each location in order to identify useful boxes and to reject useless ones. In the localization task the net outputs the top 5 boxes with the best classification scores.

In the detection task, in order to discern multiple objects from background, the net can use negative examples sampled from background images.

This model obtained very competitive results for detection and classification, and it was the best in localization in ImageNet Large Scale Visual Recognition Challenge 2013.

As in other convolutional model we can study hidden levels to find an object representation; in particular, in this model, the representation is given by OverFeat, the features extractor, which produces as output a distributed representation.

3.10 Girshick et al., Regional CNN

The idea of integrating object detection, feature extraction and classification is used also in the work presented by Girshick et al. [5], where the object classification process is built on three steps: object detection, feature extraction and finally classification.

The object detection process generates category-independent regions; for each input image they extract 2.000 regions in which an object can be.

Then the use the CNN presented by Krizhevsky et al. [7] in order to extract features from each region: the region is first resized to the required size and then given as input to the net. The last hidden layer contains (as seen before) a distributed representation of the object that can be seen as a set of features associated with the region.

Last step of the process is to use a set of class-specific linear Support Vector Machine (SVMs) in order to classify the representation of the region (that is the set of features associated with it). By combining results from the classifiers, the process outputs the location of the main object and its class.

Associated with the net description, authors give an idea for understanding what the net (in particular the features extractor) is learning.

First-level filters can be visualized directly and are easy to understand: they capture oriented edges and opponent colors.

Following levels aren't so easy to understand, so we need to use different methods: we select a particular unit (feature) and we run the net with a lot of regions (about 10 millions); we sort regions from highest to lower activations and we display the top-scoring. In this way we can understand what features is captured by a specific neuron.

This process is long and difficult, but can be used in order to understand what features are extracted at the last level from an image.

3.11 Deep network internal representation

The material discussed in the previous sections supports the conclusion that Deep Convolutional Neural Networks can be a valid solution for detection and classification tasks. Even when models deal with complex high-dimension real images, with more objects in the same image, or objects in different sizes, positions, views or other variances, DCNNs achieve remarkable performances.

It may seem the DCNNs are the perfect solution, if we don't consider the fact that they are not cognitively plausible.

Indeed, in deep networks it remains unclear what the nature of the learned representation is and why it works so well. We can use them as black boxes that given an image as input, output a distributed representation of the object (or objects) inside the image. But we aren't sure that the output is exactly the one that we thought.

A number of works have focused on understanding the representation learned by CNNs.

An important problem is that we don't know what characterizes a good representation: for example we can think to a part-based representation, where parts can be shared across objects; but we don't know what can be a part: for a face we can suppose mouth, eyes and nose, but the object parts that are important for visual recognition might be different from these.

In fact each algorithm can potentially find different and arbitrary part configurations, all giving similar recognition performances.

Zhou et al. [28] try to understand the internal representation that a CNN builds to classify scenes.

Scene classification can be compared to object classification if we think at each object in the scene as a part of an object. The representation of the category depends on the objects in the scene and on the spatial configuration of those objects.

In order to understand what a network learns, the authors trained two CNN with ImageNet images and Places Database images respectively: the first one categorizes better single objects, the second one scenes. Then they created a dataset with both kind of images (200.000 images) and they studied the activations of the layers in the nets. In particular, they extract for each layer of each net the top K images that produce the largest activation.

In the first layers images are the same (or highly similar) in the two networks, and are images with repetitive patterns (bars or net). In successive layers images are different, specific objects or places respectively.

This supports the idea that informations are hierarchically learned by the net: from simple features in the first levels to more complex in the lasts.

Another way to examine what a net has learned is to compare the performances achieved with standard images with the one achieved with simplified images.

The idea is to simplify an image that is correctly classified: they remove from the image one segment at time and they see if the image is still correctly classified; the segment removed is the one that produces the smallest decrease of the correct classification score. They iterate this process until the image is misclassified: the objective is to end up with an image that contains the minimum amount of informations that the network needs to correctly classify it.

For instance, in the case of bedroom the minimal image representation usually contains the region of the bed.

This experiment show us that in order to recognize a scene, the network must have few characterizing objects, so it uses a part-based representation finding the most salient parts.

A more difficult task is to estimate what a net learns in its internal layers: the only way in which we can interact with those units is as a black box, where we give an image as input and we study the activation of a certain unit.

Experiments made by Zhou et al. [28] follow this direction: for each interesting unit (or set of units) they extract from a mixed database the top K images that produce the largest activation. Then from each of these images they generate a set of 5.000 new images occluding a different small area in each new image. At the end they reprocess each new image and compare the new activation with the original: if they are similar the occluded area isn't important for that unit, if they are different we can deduce that the unit was trained to recognize that detail.

This approach works but it's impossible to think to use it for each unit of the network, when a CNN has usually millions of units. Anyway it is useful to understand that each unit usually focus on a small detail of the image.

Other experiments are focused on identifying the semantics of internal units: for each layer they extract the top K images that produced the large activations and they ask workers on Amazon Mechanical Turk to identify the common theme and concept that exists between those images. They had to label the set of images and to choose one category between scene, region, object, object part, texture or material and simple elements or color. The result is that first level images are associated mostly with simple elements, colors and texture materials, whereas middle level images are associated with object parts and last level images are associated mostly with object and scene.

This is another proof of the hierarchical distribution of information in deep networks.

Even if these studies show some interesting informations about what is represented inside a deep network, they also show us how difficult is to handle these networks and what they learned.

Chapter 4

Introduction to the model

4.1 Lexical acquisition process

In previous chapters we provided a high level overview of the lexical learning problem and we presented some computational models that can help building a system for objects identification and lexical acquisition.

We focus now more specifically on the problem that we address in this work: building a computational model that simulates the lexical acquisition, bringing out the whole object constraint.

As said before, lexical acquisition is the process with which an infant learns new terms: in particular, during the process, infants learn to associate each new term with its meaning.

Given a natural language we can find a lot of different kinds of words: nouns, that identify concrete or abstract entities, verbs, that identify activities or processes, adverbs and adjectives, that identify verbs or nouns modifications, conjunctions and others.

When we speak about lexical acquisition we refer, in particular, to acquisition of nouns, that is the acquisition of associations between entities names and their meanings (the entities itself). In other words we can say that during this process infants create an association between an object (or its representation) and its name.

Usually an infant has a lot of informations about objects and words acquired observing the world and hearing its sounds. Initially these pieces of information are separated; only in a second phase infants learn to associate them.

In particular we can suppose that after looking at many instances of the same object they are able to represent somehow that object in their minds. In the same way, after hearing the same word many times, even with modifications depending on who is speaking, they understand that that sound is a word and they are able to represent it in their minds as a single word.

Acquisition of this informations starts in the first days of life and it continues the whole life.

Instead, lexical acquisition starts when there is an adequate amount of informations about objects and words. At this point infants start to create associations between object representations and their names when these informations are presented together to them: for example they look at a dog and they hear someone says "That is a dog."

After the association is created when they see a dog they can retrieve its name and, in the same way, when they hear the word "dog" they can understand its meaning, retrieving a representation of the dog.

As seen before, there are three main constraints that lead the association between an object and its name: whole object constraint, taxonomic constraint and mutual exclusivity. In this work we focus on the whole object constraint.

This constraint allows infants to associate a word to the entire object that they are looking at, unless they have already created an association word-meaning for that object (in this case the mutual exclusivity constraint come into play).

Furthermore, even if the object is not completely visible, if they are able to attribute the visible part to the correct object, they can recover the association between the whole object and its name and in this way they can recover its right name.

4.2 System overview

When we try to build a system that simulates lexical acquisition we need to focus on each phase of the process.

Therefore we distinguish four main phases that we have to handle:

- the building of a representation for each object given as visual input, that corresponds to visual informations acquisition by the infants;
- the building of a representation for each word given as auditory input, that corresponds to auditory informations acquisition by the infants;
- the identification of a category for each group of similar representations (visual or auditory) previously created, that corresponds to the infants' capacity of understand that similar objects belong to the same category;
- the building of correct associations between visual and auditory representations that belong to the same category, that corresponds to the lexical acquisition process.

In order to simulate the real process, these phases are not simultaneous: construction of representations must be independent, since the building of a visual representation does not influence the building of an auditory one and vice versa; identification of category can happen after that we have acquired enough informations from the previous phases; connections between visual and auditory representations must be created in the last phase of the process, at the end of others training phases.

Each phase has its critical points that we have to face. We describe now each phase of the process, postponing implementation details to following chapters.

4.2.1 Building of a visual representation

In this first phase we simulate the process of visual informations acquisition, with which children learn to segment the scene that they see in objects and they learn to create a conceptual representation for each object.

To build correct visual representations we need a system that takes as input object images belonging to different classes and it outputs representations that (as said in section 3.1) are similar (or identical) for objects in the same class and different for objects in different classes.

The visual stimuli that we can give to the system can be a simplified image (for example a binary one or an image with white background), or a real one; this last choice, more complex to handle but more realistic, can have many levels of complexity: the two extremities are images in which there are multiple objects in a large scene, and images in which there is a single object that cover most part of each image. Furthermore, the object can be of different size, color, location in the picture, point of view or other variations, or we can allow only a kind of object, for example, if our object is a bottle, each bottle in each image must be vertical, in the center of the picture and must cover the most part of the picture.

In our experiments we choose images with only one object (or one main object) in the picture, allowing every possible variations of it.

Images come from ImageNet, a hierarchical database organized according to WordNet, with 21.841 synsets (only the noun section) and 14.197.122 images.

Our task, in this first phase, is to create visual representations starting from 10.000 images split in 100 categories; categories are selected randomly from the 1.000 provided in the ILSVRC2014 (ImageNet Large Scale Visual Recognition Challenge 2014).

As suggested in the previous chapter, we can use a deep network in order to extract an object representation, despite the complexity of real images. In particular, convolutional deep networks perform well on large datasets (as see in section 3.6). Our choice falls on the InceptionV3 net presented by Szegedy et al. [25], which has reached remarkable performances in classification, demonstrating substantial gains over the state of the art in the ILSVRC2012 classification challenge.

Given an image to the trained net, we can suppose to find a distributed representation of the main object of the input image in the penultimate layer of the net: considering that the last layer is a fully connected softmax layer used for classification, its input we assume contains a representation that leads to the classification. Indeed, we can suppose that at that layer similar activations are associated with inputs that belong to the same category and different categories are recognised because we find enough differences in activations.

In order to verify the presence of a correct representation in that layer we conducted clustering experiments, aiming to find similarities or differences in the activation states of this layer; activations are produced using different input images chosen from different categories. In chapter 6 we'll focus on these experiments.

We must underline that the building of a representation is an unsupervised process: even if we, in our experiments, know to what category an image (and the object contained) belong, CDNNs use this information only in the training of the classification layer. This means that during the building of the representation the net uses only informations that are present in the image, discovering features in an unsupervised way.

Unsupervised learning is exactly what happens in infants when they discover the surrounding world: they discover regularities in objects they see and they do that without the need of any supervision.

4.2.2 Building of an auditory representation

Similar to the previous one, this phase simulates the process of auditory informations acquisition in which infants learn to segment the audio stream in words and to build a corresponding representation.

Trying to build an auditory representation we cope with very similar problems to those that arise in the learning of the visual representation.

First of all we have to decide if the auditory input is realistic or simplified: in the first case we need a system that, given an audio file containing the sound of a word, builds a representation of that word; furthermore, the representation built must respect all the constraints defined for a visual representation, for example the fact that the system has to associate identical (or highly similar) representations to every sound containing the same word, and vice versa representations must be different with sounds referred to different words.

In this work we use a simplified input: we defined a binary array with length equal to the number of categories, in which each cell represents a category. Conceptual representations for each category are the inputs themselves. In summary we assume to be given a perfect auditory representation.

4.2.3 Category identification

This third phase simulates how infants learn categories: when an infant sees different dogs, he's able (after a while) to understand that all those objects belong to a singular category, because they are similar between them. In the same way, when they see a single dog from different points of view (for example the dog is moving) they can understand that it is always the same object. On the other side, if they look at a table, they understand that it isn't the same object seen previously, because it's very different.

In the same way, the system needs to identify that some objects are similar and belong to the same category, but others are different and belong to different categories.

At this point of the process we can produce a set of visual representations and a set of auditory representations associated with different categories; in order to recognize those categories, we chose to locate these representations into two multidimensional spaces, one for each kind of input (visual or auditory).

We use these spaces to organize representations in a topological way: similar representations are located in close areas whereas different representations are located in areas that are distant from each other; therefore, for the way in which we built representations, those derived from the same class are located in close areas (because they are similar) and vice versa happens for those derived from different classes.

To reach this target, we use a Self Organizing Map (SOM), a neural network that produces a two-dimensional discretized representation of the input space of the training samples, via competitive learning.

The training is unsupervised, because the net identifies regularities in inputs in order to find similarities and differences between them and to locate them correctly in the map. We will analyze the SOM mode of operation in chapter 7.

For each SOM training, we used a set of representations (visual or auditory) resulting from the previous phases.

At the end of the training we expect that each SOM has correctly clusterized inputs, defining for each input category a subspace associated with it. Each category subspace contains neurons that respond with maximum activation to inputs of that category.

In this way, when we give as input a new representation (associated to a specific category), SOM activation is greater in the area designed for the identification of that category. This happens for both kinds of inputs, visual and auditory, in the respective SOMs.

Indeed, SOM activation, given a specific input, allows us to understand what category the input is associated, through the identification of the area in which we find the most active neuron.

4.2.4 Word learning

At this point we have a system that, given a real image and a simplified auditory input, is able to produce corresponding representations and to map them on two two-dimensional spaces, in order to identify in each space the correct input category.

The last step is to build connections between the two maps in order to associate each area in the visual map with the corresponding area in the auditory map, where two areas correspond if they identify the same category.

This is done with Hebbian connections, which connect in a strong way neurons that are frequently active at the same time, and in a weak way neurons that aren't. The connection strength is proportional to the activation levels of the two neurons involved.

How can we handle these connections in order to simulate word learning?

If we present simultaneously to the system an image and an auditory input belonging to the same category, it produces the corresponding representations and it maps them on the two SOMs. In this way SOMs are activated in different areas with different activation levels: in particular the two areas trained to recognize the input category are strongly activated, differently from the others. At this point we update connections between maps to connect highly activated areas with strong connections. This process is done using several pairs of inputs (auditory and visual) belonging to different classes.

The same process happens in children: when they look at a dog and they hear someone saying the word "dog", they create an association between what they see and what they hear, that means between the dog category (its visual representation) and the representation of the word "dog". Repeating this event several times, associations became stronger.

At the end of the training we can test what the net has learned: if we present an object image to the system, it produces the corresponding representation and maps it on the visual SOM. Visual SOM activation is propagated on the auditory SOM through Hebbian connections, producing a specific activation of that second map.

At this point we can find the highly activated neuron in the auditory map and we can see in which area is located. Starting from this information we can recreate an auditory information, for example the sound of the correct word.

We can also use the system in the opposite direction, presenting an auditory input and extracting the visual information associated, for example by building an image representing that category.

This is the way in which people use their vocabulary: when they hear an object noun they understand what that object is building a representation in their mind, and when they look at an object they can retrieve its name thanks to the learned associations.

Image 4.2.1 shows the complete architecture of the model proposed.

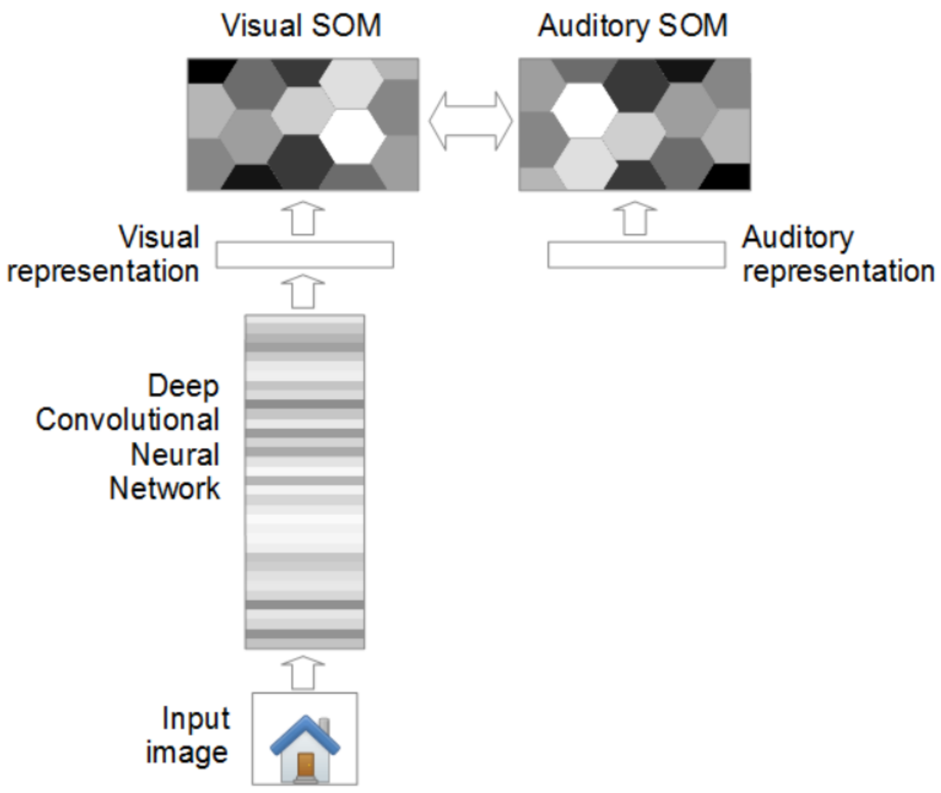


Figure 4.2.1: Complete architecture of the model proposed.

From the next chapter we start describing in details each block of the presented system, showing models used and experiments conducted.

Chapter 5

Building of a visual representation

5.1 Deep Convolutional Neural Network

In this chapter we present the deep convolutional neural network we used to build a conceptual representation of the input image.

As mentioned, current state-of-art approaches to object recognition are based on deep convolutional neural networks. Indeed this kind of network can achieve impressive performances using realistic high resolution images.

In particular, convolutional networks, after an adequate learning phase, are able to identify with good accuracy the presence of one or more objects in the input image and to recognize to which category those objects belong to.

This happens because their layers are trained in order to identify more and more complex features, to create some kind of distributed representation of the image and of the objects contained in it. This representation leads to the final classification process for the image categorization.

We present now some relations that exist between this kind of networks and the visual cortex.

5.2 Relations with the visual cortex

For what we know about the architecture of the visual cortex, we cannot say that the architecture of a Deep Convolutional Neural Network is its correct transposition. What we can do, is to identify some principles that characterize the visual cortex and that inspired the construction of this kind of models.

First of all, like in the architecture of the visual cortex, the net is made by several layers and each layer is connected with the previous and the successive ones. Furthermore, each layer uses informations from the previous layer to build more complex features.

The second principle followed by convolutional networks is the local connectivity: each neuron at a specific layer produces the strongest response to a specific input pattern present in a small region of the previous layer; the same happens in the visual cortex, where an individual sensory neuron refers to a small part of the received image. This particular region is called receptive field of a neuron. This technique brings two main advantages: the first one is that at each level not only we are able to identify the presence of a specific pattern in the input, but also to keep track of the position of this pattern; the second one is that the network is able to create a good representation of small parts of the input and then to assemble them to build more complex representations using also spatial informations.

Another principle is that of shared weights: since each neuron is trained to recognize a specific pattern in a specific location of the input, we can use replicated units across the entire visual field; these units share the same parametrization, that is same weights and bias. They form a feature map, that is a map of neurons that are able to identify the same pattern in any position of the input. In some levels (those fully connected) this principle is abandoned in order to recognize position dependent features. For example if an image of a window is presented to the net, in its first levels it tries to find bars or angles all over the input using replicated units, and then it tries to identify the entire window recognizing the presence of bars and angles in certain positions.

Last, the architecture of the visual cortex is arranged in such a way that neurons respond to overlapping regions of the input; this process can be mathematically described by a convolutional operation.

Now that we know what principles characterize a deep convolutional neural network we describe the general architecture of this kind of networks.

5.3 Standard architecture

A single deep net is a stack of multiple layers, each of them has specific characteristics. The order and the content of the stacked layers determines what the net is able to do.

In our work we are interested in particular in deep convolutionl networks and we start this section by describing the kinds of layers that characterize such networks.

During the years, various authors proposed different architectures of the deep convolutional networks, some of which are extremely complex and significantly different from what is described in the rest of this section. Despite this, the kinds of layers that we present can be considered as the basis from which also the complex models are derived.

In the convolutional networks, each layer (of any kind) has neurons organized in three dimensions (width, height and depth) and it is locally or fully connected to the previous layer. Apart from that, each type of layer is characterized by a specific organization of its neurons and by a specific operation, or a group of specific operations, that are executed in that layer.

5.3.1 Convolutional layer

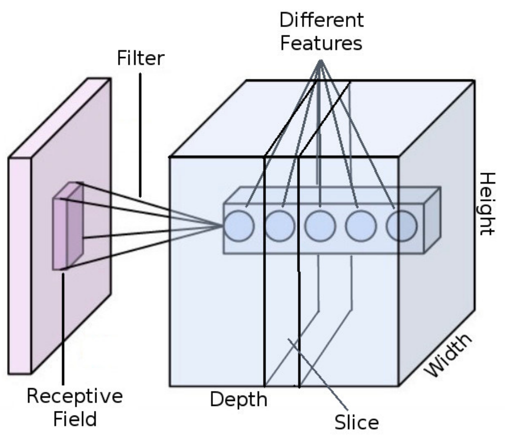


Figure 5.3.1: Each neuron of each slice of the convolutional layer gets the presence of a specific feature in the region defined by the perceptive field.

The core layer is the convolutional (showed in fig. 5.3.1), in which each neuron is locally connected to a receptive field that is small in width and height, but that extends through the full depth of the input volume.

Each slice of the output, along the depth dimension, represents a feature map, that is all neurons in a specific slice are trained to identify a specific pattern and therefore they share equal weights; for this reason, each slice can be associated to a filter, represented by the set of weights that neurons in that slice share.

Furthermore, each neuron in a specific slice refers to a specific region in the input (its receptive field) and all the neurons in the input are included in at least one receptive field.

During the forward step, each filter associated with a slice of the output, is convolved across the width and height of the input volume: for each position of the convolution, the net defines a receptive field, it computes the dot product between the neurons in the receptive field and the filter and stores the result in the output neuron correspondent to that receptive field.

To explain clearly how this layer works we make an example: the input is an image and we want to extract simple features as the presence of vertical and horizontal bars. Supposing that to extract these two kind of features we need only two filters (one for each feature) the depth of the output is 2. Indeed, for each filter we define a slice of the output, which represents a feature map that catches the presence of the feature identified by the filter in different input regions.

In our example, we use the first output slice identify the horizontal bars and the second one to identify the vertical bars.

We have to define, now, other two parameters: the dimension of each filter (that defines the dimension of each receptive field) and the convolutional stride. The first parameter allows us to control the level of detail to which we want to analyze the image: in our example, if we use small regions we are able to recognize vertical or horizontal bars (by the strong activation of the correspondent output neuron) even when the pattern covers small parts of the input image; instead if we use bigger regions, even a lot bigger than the small area that contains bars, the filters will produce less activation in the correspondent neuron.

The second parameter allows us to control the overlap of receptive fields. If stride is equals to 1 each receptive field is located one spatial unit apart to its neighbour, so they overlap almost entirely; if stride is equal to the width or height of the receptive field they don't overlap.

Furthermore, since we need to convolute the filter over the entire input we may have the problem of deciding how to compute the filter near the edge of the previous layer. To simplify this matter it is convenient to pad the input with zeros.

Dimensions of receptive fields, convolutional stride and size of zero-padding are the parameters responsible of the dimensions of the output: for example, if we use a receptive field as big as the input we have only one neuron in each slice of the output, but if we use little receptive fields and stride equal to 1 we produce a large number of output neurons in each slice.

Given an input of size D (with D neurons) and a filter of size K (with K neurons for each receptive field), with a convolutional stride set to S and a zero-padding set to P on each border, we can calculate the dimension of the output as (D - K + 2P)/(S + 1).

5.3.2 Pooling layer

The convolutional layer represents the core of a convolutional network, but it introduces two main problems: complexity and overfitting.

Trying to extract a lot of useful features, analyzing small parts of the input (that is using small perceptive fields) with a lot of overlapping between fields, the net produces an output volume that is bigger than the input. Seen that this volume becomes the input of the next layer, we risk that, layer after layer, the output becomes of intractable dimension. The complexity introduced by this kind of layer is useful in order to extract a lot of information, but it transforms the net in an unmanageable system.

The second problem is that, even if we use a large set of training data, the convolutional layer suffers of overfitting, generating informations that are too much related to the images presented as inputs.

These two problems can be solved with the introduction of a pooling layer, which is a form of non-linear down-sampling. This type of layer is usually located after each convolutional layer in order to reduce the dimension of the convolutional output, trying to maintain the important information contained in it.

The idea is to partition each feature map of the input volume (the convolutional volume) into a set of non-overlapping rectangles and to output, for each of these regions, a single value (that is a single neuron with a specific activation).

To explain how a pooling layer solves the previous problems without losing important informations, we need to keep in mind that nearby neurons in a single feature map refer to nearby perceptual fields (even overlapped), and, consequently, they refer to nearby areas of the input. When we substitute a region of a feature map with a single neuron we loose information about the presence of that feature in the precise position defined by each neuron in that region, but we maintain the information about the presence of that feature in the entire region.

By varying the size of the pool we control the level of generalization that we introduce, remembering that more generalization is useful to limit the complexity and the overfitting, but too much is counterproductive because it introduces errors due to approximation.

Usually the size of each region is small, for example rectangles of width 2 and height 2, size that allows to replace four neurons with only one, discarding the 75% of each region.

To decide the level of activation of the new neuron starting from the levels of the neurons in input we can apply different operations, showed in figure 5.3.2: an idea is to use the average-pooling, with which the level of activation of the new neuron is calculated averaging the levels of activation of each neuron in the input region; this operation leads to loss of information when only a single neuron in the region is strongly activated. For this reason the max-pooling is usually used, where the level of activation of the new neuron is calculated as the maximum level of activation of the neurons in the input region; in this way, even if the feature cover a small part of the original input and therefore it is identified by a single neuron in the region, the pooling layer preserves the information. The two operations are visible in fig. 5.3.2.

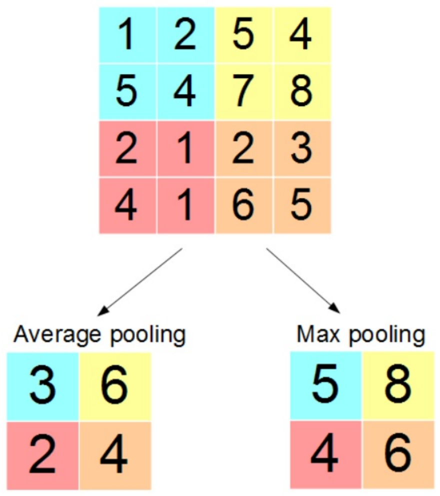


Figure 5.3.2: Application of average pooling and a max pooling on a feature maps with regions of dimension 2x2.

5.3.3 Fully connected layer

The last type of layer used in convolutional networks is the fully connected. As seen before, the convolutional layer is able to identify the presence of different patterns in each location of the input. By trying to build simple features that type of layer is very performing because in addition to the information about the presence of the feature, it exploits also the information about its context.

Instead, when it tries to build more complex features, the net needs to combine the features in the previous layers: complex features emerge by combining a set of simple features spatially organized on the basis of a specific criterion.

In section 5.2 we proposed the example of the object “window” that is recognized when the net observes the presence of bars and angles located following a specific structure.

To control spatial relations between features located in different positions of the input, the net can not use a locally connected layer. It uses instead a fully connected layer (figure 5.3.3), similar to the layers used in standard neural networks: all the neurons in the input influence the activation of each neuron of the output.

This type of layer is usually used only in the final layers of a convolutional network.

5.3.4 Complete architecture for image recognition

In order to solve the problem of object recognition, many authors proposed different architectures. Each of these architectures is based on a standard model made basically of

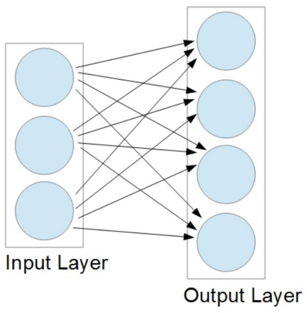


Figure 5.3.3: Connections between input and output in a fully connected layer.

three parts:

extraction of simple features: in the first part the net is composed of an alternation of convolutional and pooling layers; these types of layers allow to extract simple features from the input image in volumes of limited size.

extraction of complex features: the second part is composed by fully connected layers, which try to identify more complex features until it builds a representation of the complete input image or of the objects contained in it.

classification: the last part is a classifier that takes as input the output of the deep convolutional network and it gives as output predictions about the class of the object contained in the input image.

An example of the complete standard architecture is visible in fig. 5.3.4.

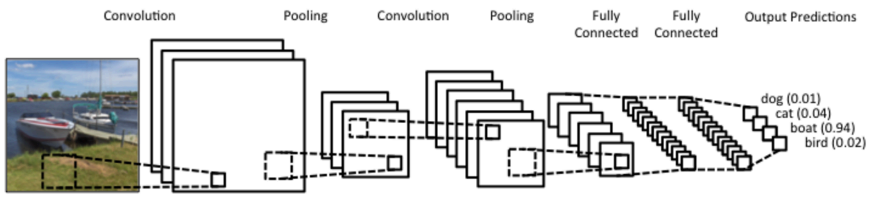


Figure 5.3.4: Standard cnn architecture. Source https://www.clarifai.com/technology.

5.4 AlexNet

The complete architecture for image recognition presented is the one used in AlexNet: the net proposed by Krizhevsky et al. [7] was the first Deep Convolutional Network that achieved impressive results in the task of image classification.

The net, that takes its name from the main author Alex Krizhevsky, became for this reason the net of reference for the successive architectures.

The authors train a large deep convolutional network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 and 2012 contest (see sec. 5.7) into the 100 different classes.

Using real images, AlexNet is the first that reaches the top-5 error rate of 15.3%, definitely better with respect to 26.2% achieved by the state-of-the-art in 2012, becoming the new state-of-the-art.

The network architecture is made of five convolutional layers (that extract simple features), some of which are followed by max-pooling layers, three fully-connected layers (that extract complex features) and a final 1000-way softmax (that classify the input in the 1000 available classes). The architecture is visible in image 5.4.1.

The firsts layers are split between two GPUs: the first one runs the layer parts at the top of the input image, while the other runs the bottom parts. The GPUs work independently in the first layers and communicate only from the fully-connected layers.

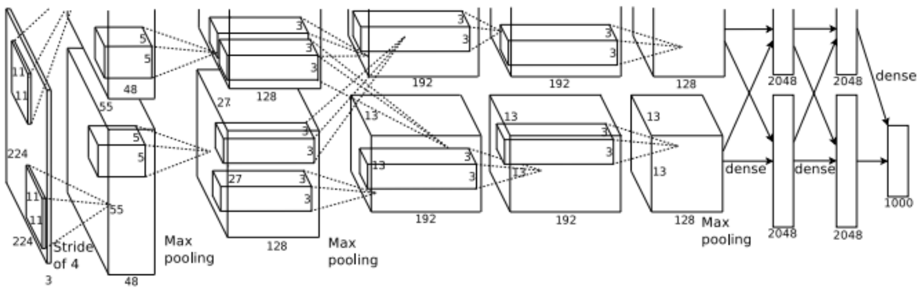


Figure 5.4.1: The AlexNet architecture, with 5 convolutional layers, 3 fully connected layers and the final softmax layer. Source [7]

The architecture suffer of the many problems related to the large number of parameters that must be trained.

In order to solve these problems, the authors propose a set of standard optimizations which influence the performance of the net, and which are used in all the successive architectures.

5.5 Standard optimizations

The problems that usually affect a complex net like AlexNet are many: first of all it has a large number of parameters that has to be set, so it needs large datasets and a lot of time; as a matter of fact, when the algorithms are run on multiple GPUs a standard CNN needs weeks to complete the training phase. Obviously practitioners prefer more limited times, to evaluate variations of performances in a reasonable time.

Another problem is overfitting: despite the large set of training data and the robustness introduced by the pooling layer, standard CNNs still suffer from overfitting.

To solve these two problems in Krizhevsky et al. [7] the authors introduce the remedies that are the state of art for these kind of problems: Rectified Linear Units (ReLU) for improving training time; data augmentation and dropout for reducing overfitting.

5.5.1 Rectified Linear Units

The Rectified Linear Units (also called ReLUs), introduced by Nair and Hinton [16], substitute the traditional neurons present in the neural networks.

Each neuron usually produces its output on the basis of the relative input, by following a function that is modelled as $f(x) = tanh(x)$, that is a sigmoid function whose output is included between -1 and 1. Instead, this new type of neurons during the calculation of their outputs follow the function $f(x) = max(0, x)$, a ramp function always greater than 0.

The use of the Rectified Linear Units brings various advantages, among which the most important is an improvement in the velocity of the computation: instead of the complex calculus of the sigmoid function, the neurons need only a comparison between the input and the value 0 in order to produce the output.

Therefore, with these neurons the training of deep convolutional networks is several time faster then that with the standard neurons; for example, Krizhevsky et al. [7] demonstrated that a four-layer convolutional neural network that works on real images with Rectified Linear Units reaches a 25% training error rate six time faster than an equivalent network with *tanh* neurons.

Seen the improvement that brings during the training phase, this optimization is used in almost all the implemented deep convolutional neural networks.

5.5.2 Techniques to reduce overfitting

For what concerns overfitting, we report two of the most used techniques which have been proposed for the first time in AlexNet.

Data augmentation

The first idea to avoid overfitting in deep networks is data augmentation, that is a technique to automatically increase the number of images in the training set. As said before, the training phase uses a large set of images, for example the ImageNet dataset contains about 14 millions of images.

Building from scratch new images for each class can be a very long process, as well as finding images from other datasets that can be easily attributed to the correct classes.

The idea behind the data augmentation technique is to artificially enlarge the dataset using label-preserving transformations. Two forms of data augmentation are in use, both of which allow transformed images to be produced from the original images with very little computation allowing the system to work without the need to store them on the disk.

The first form consists in generating image translations and horizontal reflections: this is done by extracting random patches from each image, where each patch is a subimage of the image that is a little smaller than the original image[1]. Many patches can be extracted from each image, and these patches and other their horizontal reflection can be used as new images to train the net.

Even if the produced images are strongly dependent from the original images, this technique allows to greatly increase the size of the training dataset and, consequently, to greatly reduce the phenomenon of overfitting.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. For each set of RGB pixels extracted from a subset of training images, the principal component analysis is performed. This process aims to identify principal components, which are used as parameters during the generation of the new images. In particular, to each training image several of these components are added in order to build new images.

This technique brings out an important property of natural images, namely, that object identity is robust to changes in the intensity and color of the illumination.

[1]In Krizhevsky et al. [7] original images are 256x256 and patches are 224x224.

Dropout

Another technique to reduce overfitting is to combine predictions generated from different models: if models are enough different each of them is characterized by a specific overfitting whose effects are limited by the other models.

When we use deep networks of these sizes, to build different models (with different parameters) that achieve the same performance is difficult and to combine their results appears to be too expensive.

Despite this, an efficient version of model combination exists, and it is based on the technique introduced by Hinton et al. [6] called *dropout*. This technique consists of setting to zero with probability 0.5 the output of each hidden neuron. Every time that an input is presented, the neural network samples a different architecture, choosing what neurons are *dropped out*; an important fact is that all these architectures share weights.

With the dropout technique co-adaptations of neurons is reduced, because each neuron cannot rely on the presence of particular other neurons, but it is forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

This technique reduces overfitting but, to reach the same error rate that the net can reach without *dropout*, the model needs to double the number of iterations.

5.6 Inception

We present now the architecture of the Inception deep network (also called GoogLeNet), which is the model that we chose to produce the object representation.

The model presented by Szegedy et al. [24] set the state of the art for classification (and detection) in the ILSVRC14. This improvement was not due only to more powerful hardware, larger datasets or bigger models, but it was mainly a consequence of new algorithms and improved network architectures.

Indeed, this model uses 12x fewer parameters than the architecture of Krizhevsky et al. [7] while being significantly more accurate. Furthermore, the model uses 1.5 billion of operations at inference time; this number of operations is still well inside what modern hardwar can process in reasonable time, so taht it can be used in the real world at reasonable cost, instead of keeping it only in the academic world.

The main idea of this network is to use an approach called Network-in-Network: the net is divided in *Inception modules* that are complex layers made by more simple layers. This approach is proposed as solution to two computational problems:

Sparse structure: using the standard architecture that we reported above, the net is forced to create sparse structures when optimizations like dropout are used. This is a problem because numerical calculation on non-uniform sparse data structures is very inefficient with the todays computing infrastructures.

Large number of parameters: in order to make the network work, a large number of parameters need to be set. This introduces two problems: the requirement of large datasets for the learning process and some optimization to avoid overfitting; this last point brings us back to the previous one, since that the standard technique to avoid overfitting is the dropout technique.

The use of Inception modules is useful because, as we shall show, they avoid the use of sparse matrix multiplication, clustering these structures into relatively dense submatrices, and they limit the number of parameters.

In order to understand how an Inception module works we report architectural details as presented in [24].

First of all, simple blocks in each Inception modules are convolutional or pooling, because the authors want to maintain the properties of any standard DCNN (translation, rotation and scaling invariance).

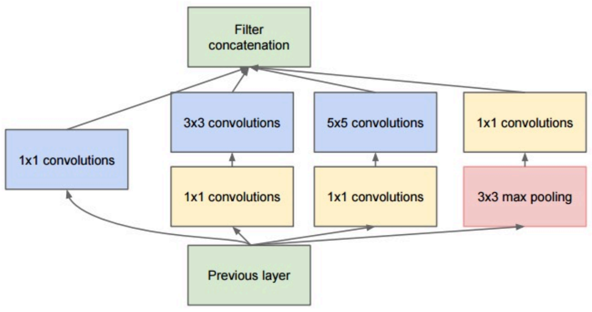
Each Inception module (as each layer in a standard architecture) tries to extract features from its input but, using only convolutional or pooling filters, it does it in a local way. In particular, the module tries to maximize the amount of information that it can extract from its input, using at the same time convolutional filters of different sizes. Sizes are usually small (1x1, 3x3 or 5x5) and allow the net to extract features of different complexity.

Additionally to these convolutional filters, in each module we can find a pooling operator. The pooling layer is not stacked on top of the convolutional layers, because this approach, which target is the reduction of the volumes, tends to create bottlenecks, by summarizing the information extracted by the convolutional layers , with the risk of losing information. For this reason it is preferred to use convolutional layers with a higher stride to reduce the volume of the output, limiting in this way the loss of informations (as described in [25]). Despite this, since that the pooling layer extracts significant information it is maintained in the architecture, and it is located in each module in parallel with the convolutional filters.

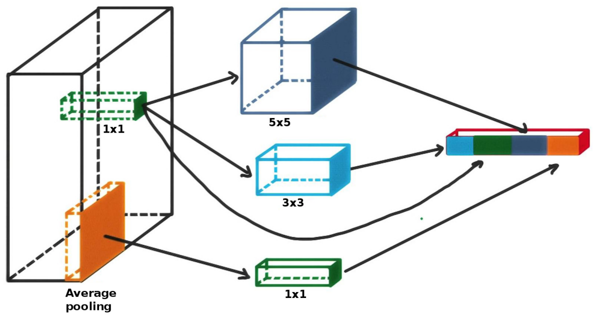
In figure 5.6.1 we can see the architecture of an Inception module in 2D and 3D, with the different filters applied on the input.

Convolutional filters of 1x1 (in yellow), are introduced to reduce the dimension before the expensive 3x3 and 5x5 convolutions: the input layer is a volume of size WxHxD (Width x Height x Depth) and each 3x3 (or 5x5) filters should work on a volume of dimension 3x3xD; using the 1x1 convolutional filters we produce a new input of size WxHx1, allowing each of the successive filters to work with a volume of size 3x3x1. This dimension reduction increases the efficiency of the net.

At the end of the module, results produced by each block of the module are concatenated in order to create a new volume (see figure 5.6.1b).



(a) 2D representation, input at the bottom. Source [24].



(b) 3D representation, input on the left. Source Google online course on Udacity and YouTube

Figure 5.6.1: Representation of an inception module in 2D (a) and 3D (b).

The standard Inception architecture (figure 5.6.2) consists of modules like the ones just described stacked upon each other, with an occasional max-pooling layer with stride 2 to halve the dimension of the volumes. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion (alternating convolutional and pooling layer).

Inception is a net made by 27 layers, where each layer contains multiple filters. To train such a complex network the authors propose an expedient: seen that features produced by the layers in the middle of the network are very discriminative, they added auxiliary classifiers connected to these intermediate layers, encouraging the extraction of characterizing features. Indeed, classifiers increase the gradient signal that gets propagated back in order to train those layer in a better way.

Even with this expedient the network can be trained in a week using few high-end GPUs, using data augmentation model like those proposed in section 5.5.2.

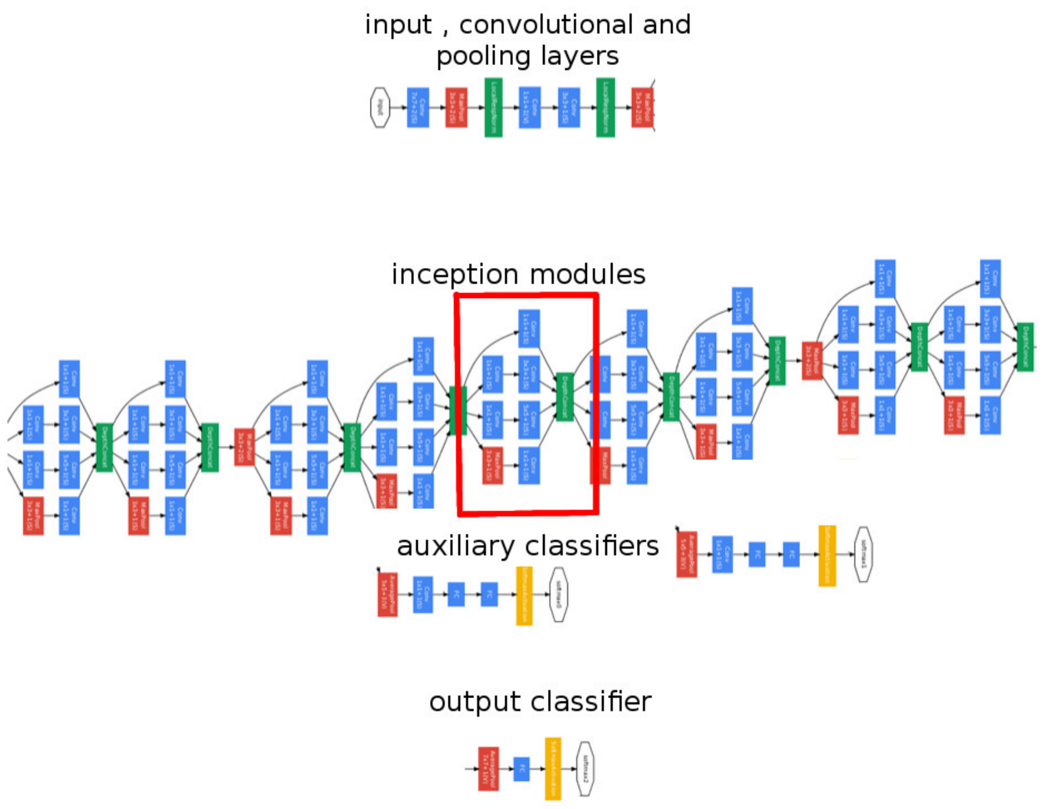


Figure 5.6.2: Architecture of the Inception network. Source [24].

5.6.1 Successive optimizations

The architecture presented so far has been improved in successive versions with the addition of optimizations affecting efficiency and efficacy of the net.

The net that we used in this thesis, in order to extract representations of the objects contained in each input image, is the InceptionV3; this version (the latest at the moment of writing) is based on the standard architecture, but it presents some optimizations which are fully described in Szegedy et al. [25].

For completing the description of the Deep Convolutional Neural Network that we used in this work, we report two of these optimization techniques that aim to reduce the number of parameters, in order to accelerate the training.

The first technique concerns the factorization of large convolutional filters into smaller convolutions. Considering a single filter of size 5x5 Szegedy et al. [25] propose to replace it with a set of sequential filters (a mini-network), with the same input and output of the original filter. The mini-network is made by a 3x3 convolutional filter, that slides over the input and produce an output of dimension 3x3, followed by a fully connected layer which output is a single neuron. This mechanism allows to reduce the number of parameters used and, therefore, to accelerate the training, without losing information.

Another similar technique to reduce the number of parameters (also proposed by Szegedy et al. [25]) is factorizing 3x3 convolutional filters into a mini-network made by an asymmetric filter of size 1x3, with 3 output units, and a fully connected layer, with a single output unit. This decomposition has the same advantages of the previous.

5.7 External resources: ImageNet and TensorFlow

In this section we present two external resources used during the work: ImageNet and TensorFlow.

ImageNet (that can be found at `http://www.image-net.org/index`) is an image database organized according to the WordNet hierarchy, in which each node of the hierarchy is associated to hundreds and thousands of images.

WordNet (which can be found at `http://wordnet.princeton.edu/`) is a large lexical resource of English. We report only the structure of the section relating to nouns, that is the one replicated in ImageNet. The information is structured in sets of synonyms (synsets), each of them is referred to a distinct concept: each synset is made of a set of synonyms, a brief definition of the concept and, usually, one ore more short sentences illustrating the use of the synset members. Synsets are interlinked by means of conceptual-semantic and lexical relations, like relations of hyperonymy or hyponymy.

ImageNet reuses the same structure: each set of images that is related to a single concept is grouped into a synset, which is identified by a name. Each synset is linked to others by means of conceptual-semantic and lexical relations. Each synset contains about 1.000 images related to the concept. Images are different for size, color, size of the object in the image, point of view on the object and other variances.

Each year, the authors of ImageNet propose a set of challenges based on the images contained in the database; this challenges, called generically ImageNet Large Scale Visual Recognition Challenge (ILSVRC) are related to the different tasks of visual recognition and they are used by researchers in order to evaluate the performances of their models in an objective way. For example, challenges proposed in last years are image classification, object localization, object detection, object detection from video and scene classification. From now on, when we report performances, we refer to results reached in these challenges, in particular in the image classification task. In this task models have as target the classification of images into 1000 predefined classes.

The second external resource that we present is TensorFlow: it is an open source software library for machine intelligence that can be found at `https://www.tensorflow.org/`.

The idea behind TensorFlow is to use a data flow graph for numerical computation. The structure used is a directed graph: nodes typically implement mathematical operations, but can also represent endpoints to feed in data, push out results, or read/write persistent variables; edges describe the input/output relationships between nodes.

The basic data structure used by the library is the tensor: a dynamically-sized multidimensional data array that flows through the graph according to the path described by the edges. Hence the name of the library TensorFlow.

The graph structure is useful because allows one to assign the nodes to different devices (e.g. CPU and GPU) and to execute them asynchronously and in a parallel way once all the tensors on their incoming edges become available.

TensorFlow provides a lot of machine learning resources, useful to solve problems related to image and language processing. In particular, we use different implementations of neural networks models (deep convolutional models and SOM models).

5.8 Use of the deep net

After the presentation of the architecture of the model used, we present in this section the work done with the deep convolutional network GoogLeNet. First of all we want recapitulate the motivations that led us to use this type of network: during the first phase of the word learning process, infants build a representation of the objects that they perceive in their visual field; to model this phenomenon we need a computational tool able to transform realistic images into some abstract representation. Our working assumption is that Deep Convolutional Neural Network can be such tool allowing us to extract "good" representations.

Deep Convolutional Neural Networks are very efficient to classify objects present in real images and we hypothesize that this ability is the direct consequence of the fact that there exists a fixed layer in the network containing a good representation of the object in the input image. Among all the convolutional networks we studied the one that achieves better results in the classification task, and therefore the one we suppose that build better representations, is InceptionV3: this net reaches 3.46% top-5 error rate[2] in the classification task of ILSVRC2012, where is required to correctly classify a set of images into 1000 predefined classes; to understand how good is this model one can compare it with the reference network, that is AlexNet (described in section 5.4), that achieves 15.3% top-5 error rate in the same task.

[2]The top-5 error rate examines how often the model fails to predict the correct answer as one of their top 5 guesses.

We used the pre-trained Inception model provided by TensorFlow[3], which implements the InceptionV3 architecture we introduced earlier in this chapter.

In particular, the provided network gets as input images of any size and then reshapes them into tensors of dimensions 227x227; after this process, each tensor flows through the net, passing through all the inception modules. At the end of the last pooling layer, the vector is of dimension (2048,1), that is an array of length 2048. This tensor contains all the informations that the net has succeeded in extracting from the input image. The last level is a classifier, which takes as input the tensor of length 2048 and returns an array of length 1000, containing the probability of each class being the correct one for the image in input.

The model that we use is a pre-trained model: that means that all the weights that are present in the net have already been evaluated through a process of training conducted by the authors of the net; in particular, the net was originally trained with training images extracted from the ImageNet dataset. We choose to use a pre-trained model following the idea of *transfer learning*, that is to use informations provided by a reliable source, instead of recalculate the same informations from scratch. The reason of this choice is tied to the fact that, as we have said many times, complex networks require a lot of time and resources as well as large datasets for their training.

Using a pre-trained net has different advantages, for example it can be used as part of a more complex model, without losing a lot of time and computational resources to re-build and retrain it from scratch. In particular, given a certain image, the pre-trained model used is already able to recognize, layer after layer, always more complex features; our assumption is that at a certain level, it is able to build a feature complex enough to capture the distributed representation of the main object of the input image.

In the Inception architecture, we believe that this distributed representation is located in the last pooling layer of the pre-trained model (the one of length 2048): this layer conveys all the information acquired by the network about the input object as it represents the only input of the final classifier.

For the purposes of this thesis we need to extract representations related to a finite set of classes of objects; in particular, we test the model using first a set of 10 different classes of objects and then a set of 100.

The first 10 classes were chosen following a criterion of similarity and difference assessed by human operators: in order to understand what is the bond between the object presented as input and the representation built by the net, we chose classes that are similar and other deeply different; for example we chose classes like *cats* and *dogs* as similar, and classes like *mugs* and *tables* as deeply different.

The 100 classes were chosen in a random way between the 1000 provided in the ILSVRC2012

[3]The model can be found at [1].

Table 5.8.1: List of the 10 classes used.

10 classes	
Birds	Guitars
Books	Houses
Cats	Mugs
Dogs	Tables
Fishes	Tree

challenge.

The lists of all the classes used are shown in figure 5.8.1 and 5.8.2.

The images of the cited classes are extracted from the ImageNet dataset and each class of interest correspond to a synset of ImageNet.

In this first phase, the goal is to extract good representations for the objects presented as input, where "good" means that representations are similar for objects in the same class and different for objects in different classes.

In order to evaluate the capacity of the network to build good representations, we present to the net 100 images for each class, we extract the relative activations at the last pooling layer and we extract a clustering of the resulting representations.

For maximizing the probability of finding a good representation inside the last pooling layer we extract only the representations of the first 100 images for class that are correctly classified by the Inception model. The idea is that if the classifier is able to correctly identify the class of the object presented as input, then we can assume that in the last pooling layer (that is the input of the classifier) there is enough information to assign the image to the correct class; vice versa, if the classifier mis-classify then the same assumption cannot be made.

Supposing that informations sufficiently characterizing build a good representation, we extract only the representations related to the correct classified inputs, where an input is considered as correctly classified if the first answer given by the classifier is exactly the right class of the input.

One problem we need to solve is that the classifier provided with the pretrained model we used, classifies the images into 1000 classes,(the 1000 used in the ILSVRC2012 challenge). In order to focus on the classes of interest, we retrain the classifier (that is the last layer of the net) to identify only the 10 or 100 classes of our interest.

The retraining is done using for each class all the images that we have available for that class, that is all the images of the correspondent synset in ImageNet, (each synset contains about 1 thousand images).

Table 5.8.2: List of the 100 classes used, referred through their synset ID.

100 classes				
n03642806	n03000684	n03017168	n03001627	n03545150
n02342885	n01677366	n02787622	n02113712	n02408429
n01503061	n04254680	n01704323	n02701002	n02974003
n04379243	n04389033	n03131574	n02423022	n04004767
n02012849	n03187595	n03871628	n04465501	n03742115
n03797390	n03733281	n03124043	n03840681	n03773504
n07747607	n03095699	n01843383	n02317335	n02109961
n02777927	n03661043	n03791053	n04263257	n02870880
n04326547	n03179701	n02480495	n02129165	n04557648
n04026417	n02119789	n03344393	n07614500	n02128925
n03452741	n07753592	n02951358	n07742313	n02739550
n02174001	n02512053	n03995372	n02510455	n03495258
n03481172	n04256520	n01667114	n02835271	n04467665
n04507155	n04228054	n02011460	n07693725	n02128385
n03467517	n03977966	n02980441	n07749582	n04429376
n04355933	n01689811	n13104059	n03792782	n04086273
n04380533	n07745940	n02121620	n09332890	n04266014
n02086079	n02084071	n03445924	n04347754	n07753275
n06785654	n02814860	n02655020	n09472597	n02206856
n02870526	n02930766	n06267145	n03444034	n02437312

At the end of this process, we use the net to extract 100 good representations for each class, that is the 100 activations of the last pooling layer relative to 100 correctly classified images for each class. These representations will be the input of the next phase of the word learning process.

5.9 Use of AlexNet as reference network

In order to evaluate the performances achieved by InceptionNet inside our model, we compare the results achieved by this net work with the results achieved by AlexNet. The comparison is made in particular for the experiments done on 10 classes, supposing that the differences in the results can be extended to a larger number of classes.

The model used is a pre-trained model of AlexNet, provided by TensorFlow.

The model classifies each input image into one of the 1000 available classes. In order to compare the results, we modify the model following the modification done on InceptionNet: the classifier is retrained to classify the input images into one of the 10 available classes. The retraining is done on the same training set used for the retraining of the Inception classifier.

The representations are extracted from the last fully connected layer of the network, of length 1000 (for comparision: the Inception model has 2048 values in this layer). Each representations, as before, is an array of values included between 0 and 1.

For each class we consider the 100 examples correctly classified by the Inception classifier and we present those 100 examples to AlexNet in order to evaluate the capacity of the network of correctly classify the same examples. For each class, the network is able to correctly classify about 95% of the examples provided. Therefore, we have about 95 examples for each class that are correctly classified by both networks.

When we conduct experiments using InceptionNet, we use all the 100 examples (since they are correctly classified by the net); instead, when we use AlexNet, we use only the 95 examples that are correctly classified by both networks. As for InceptionNet, this choice is guided by the idea that the representations that are correctly classified, are those able to catch all the significant features of the input image and therefore, they contains a good representations of the image presented to the net.

Given the different dimensions of the representations and the different architectures that produce those representations, we can not compare directly the representations provided by the two Deep Convolutional Network; instead, we can compare the results reached through some evaluation method.

Understanding what is contained in each representation, both if it is produced by AlexNet and by InceptionNet, is difficult, because each representation is a large array with values contained between 0 and 1. In the next chapter we describe the experiments that we conducted in order to understand the goodness of these representations, and to compare the performances reached by the two net works.

Chapter 6

Evaluation of the representations

As said before, we extract from Deep Convolutional Neural NetworkS a good representations for the objects presented as inputs, where the representations can be considered "good" if they are similar for inputs in the same class and different for inputs in different classes.

The assumption concerning the presence of good representations, is driven from the fact that the representation that we consider is the only input of the classifier: if the classifier is accurate, then these representation needs to contain enough information to discriminate between the classes.

In order to investigate the validity of this hypothesis, we conduct experiments focused to evaluate the presence of a certain level of similarity between the representations extracted.

Formal concepts of similarity are presented in the next sections, but we present now an informal concept, useful to understand the approaches used.

The similarity concept used, that takes advantage of the idea of distributed representation, is based on the fact that all the representations that belong to a certain class are characterized by the presence of high values in some positions and low values in other positions; two representations that belong to different classes are different because they have high and low values in different positions.

This concept derives from the fact that the arrays used are the representations of the objects presented as inputs, from which the deep network has extracted features of increasing complexity: at the considered level all the features previously extracted are funneled into the array given as output, therefore we think that different positions of the array catch different features.

One of the most important difficulties encountered trying to find similarity between the representations is the structure of the representations themselves: given an array of length 2048 with values included between 0 and 1 is difficult evaluate the distribution of values inside of the array and the possible similarities between two arrays.

Usually each attempt of approach is guided by an intuitive idea, matured observing the set of available data. In this case, the observation of the data is not sufficient to chose an approach with respect to another, therefore we grope, trying to find these similarities through different methods.

The methods used are two, the first one based on the concept of prototype and the second one on the clustering.

6.1 Similarity to the prototype

The first method used in order to find similarities passes through the construction of a prototype for each class and the evaluation of the distance between the prototypes and the representations.

Definition 1. *The prototype P of a class C is a artificial exemplar of the class C, calculated on the basis of the representations of that class in order to be the most representative of that class; it is an vector of the same length as the representations provided by the Deep Convolutional Neural Network, and it is calculated as the mean of all the representations* r *that belong to the class C.*

$$Prot(C)[i] = \frac{\sum\limits_{r \in C} r[i]}{|C|} \tag{6.1.1}$$

If the representations that belong to a certain class share the distribution of values inside them, that is all the representations of objects member of a certain class present high values in certain positions and low values in other positions, then the prototype of that class catch this distribution.

Ideally, each prototype, in order to correctly discriminate each class, must be enough different from the others. Furthermore, the representations that are members of a certain class, must be very similar to the prototype of that class, or, at least, more similar to that prototype than to the others.

In order to catch this similarity, we calculate for each class the distance between its prototype and the members of each class: the result that we expect is that this distance is small for the members of the same class of the prototype and big (or at least bigger) for the members of the other classes.

In order to calculate the distance between the representations that belong to a class C

and a prototype $Prot(C')$ of a class C' (where C' can also be equal to C), we follow these steps:

1. we calculate the prototype of the class $Prot(C')$ as described in equation 6.1.1;
2. we consider each example of the class C and we calculate the distance from the prototype as the absolute value of the difference point-to-point between the two arrays; the distance at this point is an array of the same length a the representations;
3. for each example, we transform the array into a number, that is the mean of the values contained in the array: if the representation considered is similar to the prototype, this value of mean distance is small, if the representation is not similar the value is bigger;
4. at the end, we average all the mean distance calculated from the representations of the class C in order to represent the distance between the class C and the prototype of the class $Prot(C')$ as a single number: if all the example are similar to the input, the value calculated is small, but if they are different the value is big[1]

Indeed, the similarity can be described formally as follow:

$$Similarity(Prot(C'), C) = 1 - \frac{\sum_{r \in C} \frac{\sum_{i=0}^{length(r)} |Prot(C')[i] - r[i]|}{length(r)}}{|C|} \tag{6.1.2}$$

The results are described in table 6.1.1.

[1]The values contained in the representations are included between 0 and 1, therefore the distance calculated between the representations of the class C and the prototype of the class $Prot(C')$ is also included between 0 and 1. For this reason, when we said the the value of distance is small we mean that is near 0, and the more it goes closer 1 the more is considered as big.

Table 6.1.1: Mean distance from the examples of each class (on the columns) and each prototype (on the row). For each row the smallest value is highlight.

		Classes									
		tables	mugs	fishes	trees	dogs	houses	books	guitars	cats	birds
P	tables	0,06287	0,10131	0,12756	0,11718	0,12241	0,11084	0,11900	0,10234	0,11590	0,08423
r	mugs	0,10959	0,05459	0,13666	0,14012	0,13070	0,12471	0,13700	0,11164	0,12486	0,10749
o	fishes	0,10324	0,10406	0,08719	0,12581	0,12275	0,11417	0,11247	0,11313	0,11460	0,09827
t	trees	0,09065	0,10531	0,12360	0,08941	0,12216	0,09899	0,11867	0,10921	0,11905	0,07926
o	dogs	0,08950	0,08950	0,11416	0,11578	0,09579	0,09871	0,11465	0,09277	0,09903	0,08732
t	houses	0,10574	0,11133	0,13338	0,12041	0,12652	0,06798	0,13446	0,11770	0,12175	0,10561
y	books	0,08911	0,09883	0,10690	0,11531	0,11768	0,10968	0,09276	0,10205	0,11138	0,08263
p	guitars	0,12334	0,12436	0,15846	0,15674	0,14669	0,14382	0,15295	0,04187	0,14384	0,12130
e	cats	0,10750	0,10818	0,13052	0,13718	0,12354	0,11845	0,13287	0,11444	0,07127	0,10872
s	birds	0,08609	0,10107	0,12446	0,10766	0,12210	0,11258	0,11439	0,10216	0,11899	0,06101

Ideally we can expect that for each prototype the class with the smallest distance is the same to which the prototype belongs.

The experiment proves this hypothesis only for 7 classes on the 10 available: for the prototypes of the classes *trees*, *dogs* and *books*, the nearest class is *birds*.

This means that the examples of the class *birds* are on average nearer to the prototypes of other classes.

The behaviour can be a consequence of the fact that the classes *trees*, *dogs* and *books* have some examples that are a lot different from their prototypes: if these examples are enough in number, they increase too much of distance between objects in the class and its prototype.

Another cause can be found in the computation through which we calculate the distance: by averaging the distances calculated in a single array (or representation) we do not consider how the differences are placed inside of the array. The positions of the differences are important because, as we said before, we think that each position inside the array can be considered as a feature extracted by the Deep Convolutional Neural Networks. For this reason we think that the differences between a representation that belongs to a certain class and the prototype of that class should be small and well distributed: however this is not the only way through which we can produce a small mean distance: if the differences are strong but are concentrated in little portions of the array, given the length of the array (2048), we can still produce a small value of distance.

Therefore, we experiment with another method through which we can determine the similarities between the representations, taking into account also the positions of the values inside of the representations themselves, and, consequently, the distribution of the differences between two representations.

6.2 Clustering

Clustering is defined as the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar to each other than to those in other groups.

Given its definition, clustering is arguably a good fit for our problem of finding similarities and differences between the representations generated by the Deep Convolutional Networks.

With the previous method, the most problem was the definition of similarity, whose, by averaging the distances calculated inside of a single representation, ignored the positions of the differences inside the array.

The clustering solves this problem defining a new concept of similarity: the similarity between two representations is based on the Euclidean Distance between the two representations. Each representation can be consider as a point in a vector space of dimensions equal to the length of the representation. The similarity is then calculated as a function of the distance between points in this vector space.

To better understand we can think to representations of length three, that is with three features, with values included between 0 and 1. We can create a vector space in three dimensions and locate each representation in this space through the values of its features; each feature represents the value assigned to one of the three coordinates in the space. At the end, we can calculate the similarity between two representations as the distance between two points in this space.

The Euclidean Distance between the representations q and p is calculated as:

$$ED(q,p) = ||q-p||_2 = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \tag{6.2.1}$$

where n is the length of vectors q and p.

The equation 6.2.1 shows two advantages of this measure: the distance can be calculated between representations of any length, and we do not need to build a real vector space of large dimensions (which can be very difficult to build) in order to calculate the distances.

Furthermore, this concept of distance take into account the positions of the values inside of a representation, considering them as different features into the vector space. For this reason, if the differences between two arrays are small and distributed along all the representation then the distance is small, but if we have strong differences even concentrated into a small portion of the array then the final distance will be enough big.

Indeed, the Euclidean Distance has all the characteristics useful to find significant similarities and differences between the representations.

Once we define the correct measure of distance, we use it inside two different clustering algorithms[2] and to compare their results:

K-means clustering: the K-means algorithm aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean; the mean is considered as the prototype of that cluster. The algorithm follow three main steps: the initialization, in which it chooses k observations that are considered as the firsts prototype; the assignment, in which the clusters are created assigning each observation to the nearest prototype, where the distance is calculated as Euclidean Distance; the update, in which for each cluster it chose a new mean on the basis of the members of the cluster. The second and third step are repeated

[2]In order to maintain the terminology usually used in the clustering processes, we refer to the representations with the name of observations.

until the clusters become stable, that is until the differences in the partitioning are small or null between an iteration and the next.

Agglomerative clustering: the agglomerative algorithm aim to create clusters using a bottom-up approach: initially each observations is considered as a cluster with a single element; then the algorithm gradually put together couples of clusters to create bigger clusters. The two clusters chosen to be put together in each iteration, are the clusters whose members are nearest according to the Euclidean Distance. The clusters are agglomerated until the algorithm reach the predetermined number of clusters.

For both the algorithms we set to 10 the number of clusters that the process has to produce, in order to force the building of clusters that replicate the division in classes of the representations.

In order to evaluate the results reached, we introduce different measures:

1. distribution of the member of a single class in different clusters;
2. uniformity of the clusters;
3. intra-cluster and inter-cluster distances;

For what concerns the distribution of the members of a single class in different clusters, we consider for each class the clusters in which we can find the members of that class. The results can stress four different situations: if all the members of a class are grouped into a single cluster this means that the similarity between them is very strong. If they are mainly grouped into a single cluster, with few element distributed in other clusters, this means that the main part of the members are similar, but that we can identify some exceptions. If they are equally divided into a few number of clusters, it can be argued that inside of a single class, the images can be divided in sub-groups that are characterized by a set of specific features: for each sub-group the Deep Convolutional Network produce a set of representations that are similar between them but different from those that belong to other sub-groups; in this case forcing the number of clusters, the algorithm groups together sub-groups of different classes. In the last case, if the representations are distributed into a large number of clusters, this means that for that class, we can not identify a strong similarity between the members, because they are very different.

The charts in figures 6.2.1 and 6.2.2 show the results reached using respectively the K-Means and the agglomerative algorithms.

We can see that for 7 classes on the 10 available the members of the class are distribuited mainly in one cluster (the main cluster is colored in blue).

The members of the classes *books*, *dogs* and *fishes* are instead distributed in more main clusters (two or three), denoting a rift between the members of these classes: as said before, this can happen because the Deep Convolutional Network extracts features from the input images and, if images that belong to a class are enough divisible in two or three types (or sub-groups) of images, the equivalent representations will replicate this division.

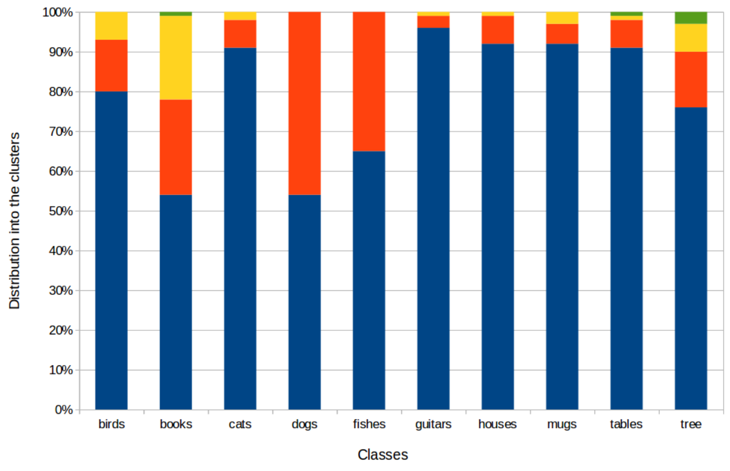


Figure 6.2.1: Distribution of the members of the classes in the different clusters using the K-Means algorithm.

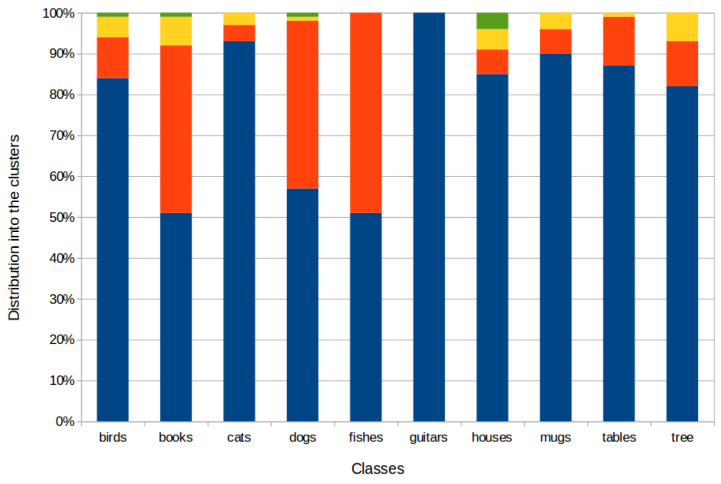


Figure 6.2.2: Distribution of the members of the classes in the different clusters using the agglomerative algorithm.

The second evaluation concerns the uniformity of the clusters: the more the clusters are uniform, the more the members of the same class are characterized by a strong similarity between them and by a strong difference with the members of the other classes; the less the clusters are uniform, the more there is confusion in the representations, that is the observations that belong to different classes are similar between them.

The charts in figures 6.2.3 and 6.2.4 show respectively the structures of the clusters created using the K-means and the agglomerative algorithm.

Results from both the algorithms show that 8 clusters out of ten are completely or almost completely uniform, whereas the second and the fourth clusters (that are also the largest one) are mixed. In particular, the classes *fishes* and *dogs* are divided among these two clusters.

We can notice that even the mixed clusters (the first and the fourth) do not contain all the classes in equal parts: the main classes are *fishes* and *dogs*, which are not present (or are little present) in the other clusters. This means that those classes are similar between them but enough different from the other classes. As explained for the previous measure, if the images in input for the classes *fishes* and *dogs* are separable in sub-groups (on the basis of the features extracted by the Deep Convolutional Network), the algorithm of clustering can group together sub-groups that belongs to different classes, if these are more similar then the sub-groups that belong to the same class.

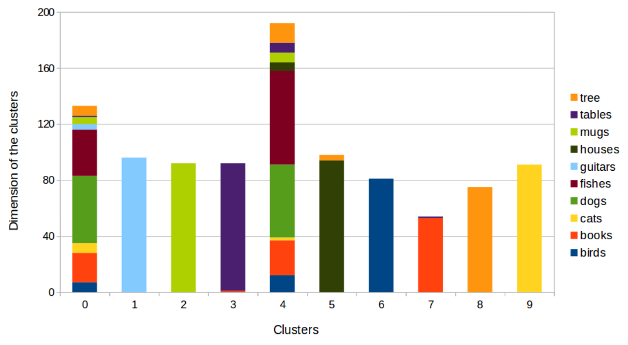


Figure 6.2.3: Structure of the clusters created with K-means algorithm.

The creation of mixed clusters can also be motivated by the fact that we set the number of clusters that we want: if we do not set the number of clusters as a predefined number, the algorithm can recognize the different sub-groups and divide them into different clusters, but forcing this number we force the algorithm to find similarity even when they are not significant. Other experiments in this direction have not been done.

This firsts two measures give us an important information: the idea that the arrays that we extract from the Deep Convolutional Network are good representations[3] of the images presented as input to the net is true. In most cases, the representations that belong to a certain class are enough similar to create a uniform cluster, with few exceptions.

[3]A representation is considered as good if more similar to other representations that belong to the same class that to representations that belong to different classes.

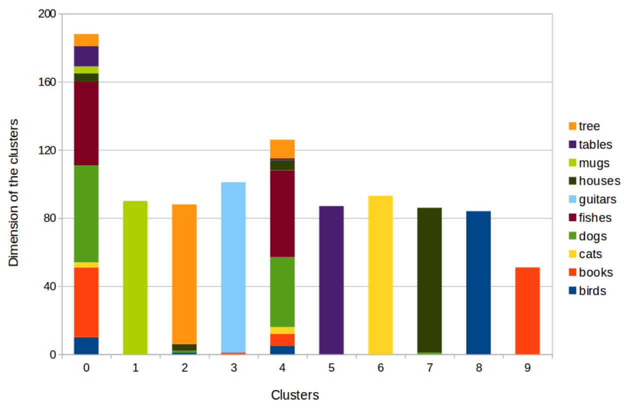


Figure 6.2.4: Structure of the clusters created with agglomerative algorithm.

The third measure that we present has as purpose the evaluations of the compactness of each cluster: the more the clusters are compact, the more the representations in the clusters are similar.

The compactness of each cluster is calculated as the ratio between the intra-cluster and the inter-clusters distances, defined as follow:

Definition 2. *Given a cluster C, the intra-cluster distance is measured as the mean distance between each couple of representations (i, j) that belong to C. The distance is measured as Euclidean Distance in the vector space, considering each object as a point in that space. Let*

$$I_c = \{ED(i,j) | i,j \in C, i \neq j\} \tag{6.2.2}$$

then

$$D_{intra}(C) = avg(I_c) = \frac{\sum\limits_{d \in I_c} d}{|I_c|} \tag{6.2.3}$$

Definition 3. *The inter-clusters distance is measured as the mean distance between each couple of representations (i, j). The distance is measured as Euclidean distance in the vector space, considering each object as a point in that space. Let*

$$E = \{ED(i,j) | i,j \in \bigcup_k C_k, i \neq j\} \tag{6.2.4}$$

then

$$D_{inter} = avg(E) = \frac{\sum\limits_{s \in E} s}{|E|} \tag{6.2.5}$$

The intra-cluster distance indicates the degree of proximity of the representations that belong to the same cluster: if all the representations are near to each other, the intra-cluster distance is small, but, if the cluster covers an area that is big (for example with representations on the border of the vector space) the relative intra-cluster distance is big. Considering this measure alone is of little significance: we do not know the maximum dimension of the vector space occupied by the representations, therefore we can not understand if the number expressed by this measure is big or small.

The second measure of distance (the inter-clusters distance) is useful in this regards: if all the representations available are sparse and located in a large area, then the measure is big, but if they are concentrated into a small area the number is small. Comparing this measure with the previous one we can have a more complete picture about the organization of the clusters.

Indeed, starting from the previous measures of distance, we can calculate the compactness of each cluster as the ratio between the two measures:

Definition 4. *The compactness of a cluster C is the ratio between the intra-cluster distance of C and the inter-clusters distance.*

$$compactness(C) = \frac{D_{intra}(C)}{D_{inter}} \tag{6.2.6}$$

The compactness is always a positive number, hopefully less than one. Indeed, if the compactness for a cluster is near 0 that means that the cluster is compact with respect to the distribution of all points, because the average distance between representations that belong to that cluster is much smaller then the average distance of two generic objects. Vice versa, if the number is near 1 or bigger than one, that means that the representations in the cluster are sparse, that is the average distance between the representations that belong to the considered cluster is close to the average distance of any pair of representations.

The results of the measurement are reported in Table 6.2.1 and in Figure 6.2.5.

The results show us that the clusters are not much compact: in particular, all the clusters have compactness above the value of 0.6 and one of the cluster compactness is larger than 1. This means that each cluster contains representations that are sparse and located in large area, without a clear separation from the other clusters.

The cause can be found also in the structure of the representations used: arrays of length 2048 and with values included between 0 and 1 generate a vector space with many dimensions (2048) and ranges of values for each dimension very small (fro 0 to 1). This factors influence the arrangement of the representations in the vector space, influencing, consequently, the values of compactness for each cluster.

The classes *dogs* and *fishes* are again the classes that bring out an anomalous behavior: as before, if we suppose that in each of these classes the representations are grouped into

Table 6.2.1: Compactness of the clusters and main class for each class, using the two different clustering algorithms.

Cluster	Main class	K-means	Agglomerative
1	birds	0,77675	0,78600
2	books	0,86713	0,85587
3	cats	0,84716	0,85773
4	dogs	1,09442	0,95213
5	fishes	0,92385	1,07369
6	guitars	0,63546	0,65270
7	houses	0,85416	0,83554
8	mugs	0,73200	0,72332
9	tables	0,81171	0,81022
10	tree	0,98071	0,99610

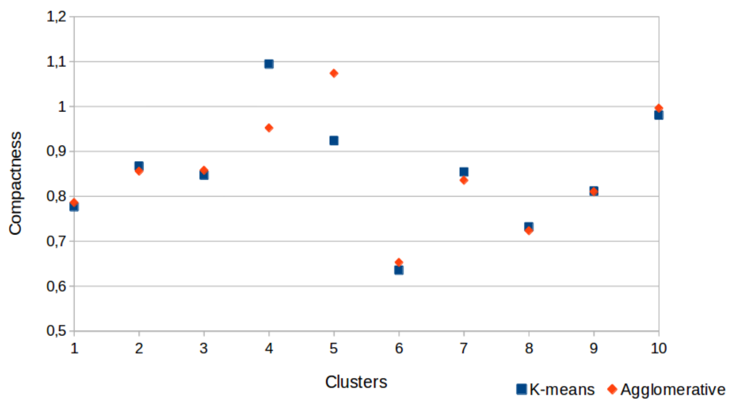


Figure 6.2.5: Compactness of the clusters using two different clustering algorithms.

more little distinct groups and that the clustering algorithms group them into mixed clusters, we can understand the reached results. The clusters here indicated as 4 and 5 are the mixed clusters, one with more *dogs* and the other with more *fishes*. The presence of different classes, whose members are near but distinct, increase the space covered by the clusters, increasing, consequently, the resulting values of compactness.

Unfortunately, the dimension of the vector space impedes us to visualize directly the representations in the space, therefore the explanations given so far can be considered as hypothesis conform to the evaluations done.

In the next chapter we continue the simulation of the word learning process, and, in particular, we will introduce a neural network that is extremely useful to confirm the hypothesis done so far about the representations.

In the last section of this chapter we compare the results presented in this section with the analogous results achieved using AlexNet instead of InceptionNet as network for the extraction of the representations. This comparison is useful because AlexNet is the reference network for studies about Deep Convolutional Networks and because we want to understand if InceptionNet is able to build better representations.

6.3 Comparison with AlexNet

In order to understand the goodness of the representations provided by InceptionNet and to evaluate the increasing in performance produced using this net, we compare its clustering results with the results produced using AlexNet.

The inputs of the cluster algorithms are, in this case, the representations extracted from the last fully connected layer in the AlexNet architecture: it's an array of 1000 elements, taking values in [0,1].

As those extracted from the InceptionNet, we expect that the representations from AlexNet respect the characteristics of a good representation: they should be similar for objects in the same class and different for objects in different classes.

Also, since that the performance of AlexNet in the classification task are worse than those of InceptionNet, we expect also that the capacity of correctly creating distinct representations for each class is worse.

For what concerns the distribution of the members of the classes in the different clusters, we can see in fig. 6.3.1 and 6.3.2 that they are distributed in a larger number of clusters with respect to what happens using the representations extracted from InceptionNet. In particular, the members are in large part concentrated into a single cluster, but the remaining part is divided into many cluster.

If we focus on the classes that were considered as problematic in the previous section, we can see that, in this case, the classes *dogs*, *books* and *fishes* are not as problematic as before: this change suggests that the two different architecture used for the extraction of the representations are able to focus on different features, producing different results for different classes.

Instead, if we focus on the structures of the clusters produced by the K-means and the agglomerative algorithms (respectively images 6.3.3 and 6.3.4), we can see that the clusters created are more mixed: there are fewer uniform clusters (one or two) and even those uniform clusters do not contains all the members of the classes they aim to represent; indeed, they contain less than half of their members. See, for example the clusters related to the class *guitars* (colored in cyan) and that related to the class *mugs* (that colored in light green).

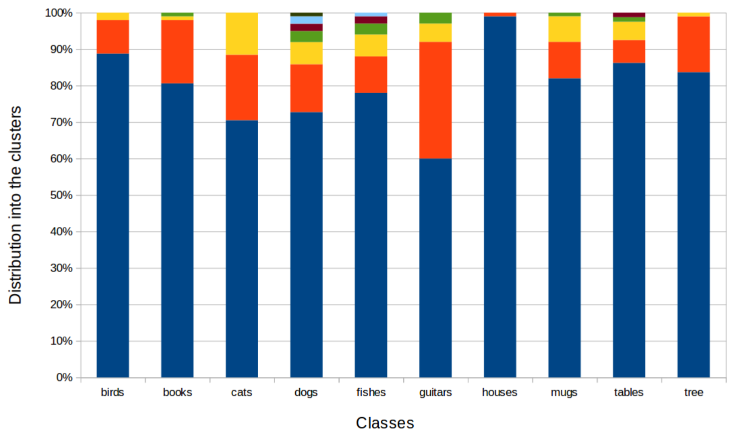


Figure 6.3.1: Distribution of the members of the classes in the different clusters using the K-Means algorithm

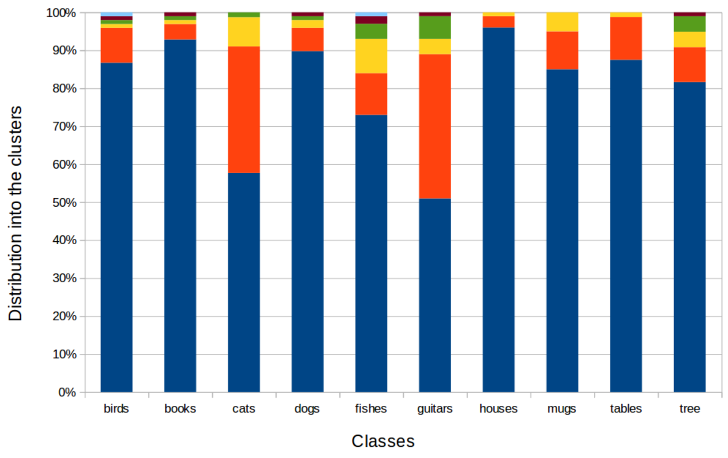


Figure 6.3.2: Distribution of the members of the classes in the different clusters using the Agglomerative algorithm

This confusion between the clusters suggests that the representations extracted by AlexNet are not as good as those extracted by InceptionNet. In addition, we have to consider the fact that in both the networks we extract only the representations that are well classified by the networks and that the dimensions of the representations are different: in InceptionNet the representations have length of 2048, while in AlexNet they are only of length 1000. For this reason, we can expect that the clustering algorithms have an easier time in trying to discriminate between objects described using the first type of representations, just because the vector space is simpler. In facts, the representations that are clusterized in a better way are those from InceptionNet.

In order to complete the comparison between the two types of representations, we calculate the compactness of the clusters.

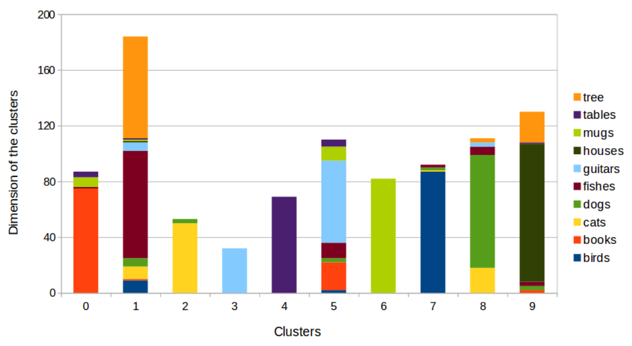


Figure 6.3.3: Structure of the clusters created with K-means algorithm.

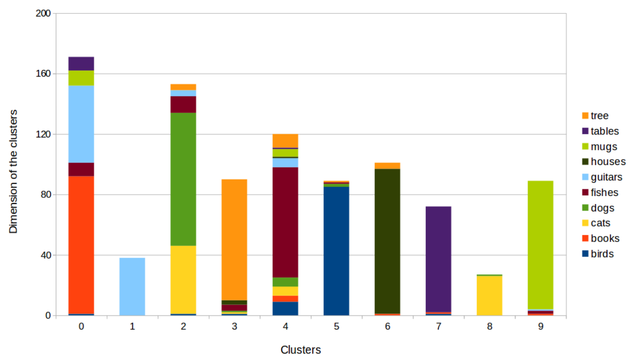


Figure 6.3.4: Structure of the clusters created with agglomerative algorithm.

The chart in Figure 6.3.5 shows that the clusters on average are more compact then those produced by the Inception network. This can be considered as a good result, but, for the purposes of this thesis, this can not be considered as an advantage of AlexNet on InceptionNet, because the clusters, even if they are more compact, are too much mixed.

For our purposes the uniformity of the clusters is more important than their compactness and, indeed, the mixed clusters generated from the representations extracted from AlexNet compromise excessively the final performance of the model. This reason leads us to prefer InceptionNet.

With this comparison between the representations extracted from the two Deep Convolutional Network we can understand the motivations behind the increasing of the performances produced by InceptionNet: given the complexity of its modules, InceptionNet is able to produce better representations, which lead to a more accurate classifier. The more the representations are uniform inside of each class and distinct between different classes, the more the classifier is able to achieve better performances.

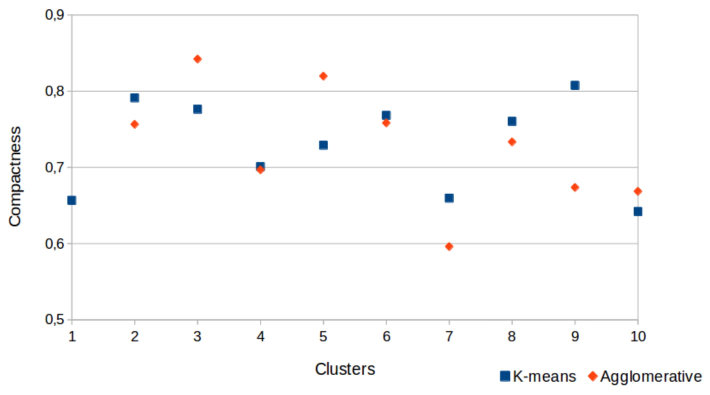


Figure 6.3.5: Compactness of the clusters using two different clustering algorithms.

Chapter 7

Self Organizing Maps and their training

7.1 The standard architecture

As we described in chapter 4, children learn to segment the visual field into the objects that compose it. Infants are able to associate a representation to each object that brings out its more representative features. In our system we simulated this process with a Deep Convolutional Neural Network.

Next step in the word learning process is to group these representations into sets that represent categories, in order to associate to each of these visual categories the correct name (or sound). In order to create these sets, in this work we explore the idea that representations can be grouped on the basis of their similarities and their differences. As we said many times, representations generated by the Deep Convolutional Neural Network can be considered as good if are similar for inputs that belong to the same class and different for inputs that belong to different classes.

If the generated representations follow this rule, we can easily map them into a vector space, using a Self Organizing Map.

A Self Organizing Map is a neural network that produces a low-dimensional discretized representation of the input space of the training samples. In particular, the network is a lattice, organized so that neighbour neurons respond to similar inputs or, also, similar inputs are categorized in neighbour areas of the map.

The Self Organizing Maps have many features useful for the purpose of this thesis:

1. the network works in an unsupervised way: it identifies the similarities in the inputs and it uses them in order to organize the inputs in a topological way;

2. the network, through the process of dimensional reduction, allows one to visualize in an understandable way the similarities between the inputs;

3. the network, organizing the inputs, preserves the topological relations that exist between the inputs.

This kind of network can be considered as psychologically plausible: with the organization in areas and the fact that they are not supervised, the Self Organizing Maps simulate well how the brain works.

In the training phase, we present to the network the set of inputs that it shall learn to categorize: the set of inputs is presented many times to the network which updates its weights until it is well organized. The number of epochs (that is the number of times that the training set is presented to the net) has to be well evaluated in order to reach better performances.

To update the weights the net does not use the classic loss based learning approach, but, instead, it applies competitive learning: when an input is presented to the net, its neurons compete to become the Best Matching Unit (usually called BMU), that is the unit that respond with the highest activation.

Given an input i $(i_0, i_1, ...i_k)$ and a neuron j with weights $(j_0, j_1, ...j_k)$ the activation of the neuron j when the input i is presented can be calculated as

$$\sum_{l=1}^{k} i_l j_l \tag{7.1.1}$$

When the BMU is identified the weights that connect that unit to the input (each neuron of the net is fully connected to the input) are updated in order to improve the capacity of that neuron to identify that specific input or highly similar inputs. These are not the only weights that are updated: the weights associated to neurons near the BMU are updated in a measure that is proportional to the distance from the BMU; neurons that are near the BMU are strongly updated, whereas neurons that are far from the BMU are weakly updated or not updated.

Mathematically the weights $\mathrm{w_j}$ associated to a neuron j are updated as reported in the equation 7.1.2. We can see that if η and H are set to 1 the weights become identical to the input presented, which guarantees the maximum activation for that neuron.

$$\mathrm{w_j} = \mathrm{w_j} + \eta H(j, BMU)(\mathrm{i} - \mathrm{w_j}) \tag{7.1.2}$$

- $\mathrm{w_j}$ are the weights associated to the neuron j;

- i is the input presented to the SOM;

- η is the learning rate. It is usually decreased during the training, starting from 0.1 and decreasing to 0.01 in the first phase of the training (auto-organization); in the second phase (convergence), the learning rate decreases even further.
- $H(\text{j}, BMU)$ is a measure inversely proportional to the distance between the neuron j and the BMU: when j is the BMU, $H(\text{j}, BMU)$ is 1, when j is far from the BMU $H(\text{j}, BMU)$ becomes 0. This measure allows one to update the weights of the neurons in a way that is proportional to the distance between the neurons and the BMU. More formally, $H(\text{j}, BMU)$ is defined as in the formula 7.1.3, which is a Gaussian with parameters $\mu = d(\text{j}, BMU)$ and $\sigma^2 = \sigma^2(n)$ (7.1.4) : $d(\text{j}, BMU)$ is the distance between the neurons, n is the iteration of the training, σ and τ are parameters used to modify the function during learning.

$$H(\text{j}, BMU)_n = exp\left(-\frac{d(\text{j}, BMU)^2}{2\sigma^2(n)}\right) \tag{7.1.3}$$

$$\sigma(n) = \sigma * exp\left(-\frac{n}{\tau}\right) \tag{7.1.4}$$

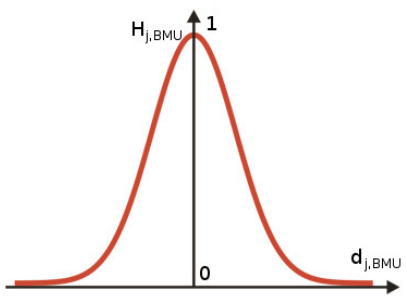


Figure 7.1.1: Neighbourhood function.

At the end of the training the network is expected to be organized in different areas: each area contains neurons that are able to recognize similar inputs, responding with a strong activation when that type of input is presented.

The network allows us to categorize new inputs: when the new input is presented its neurons are activated in different way and, finding the BMU, we can find the area of maximum activation that indicates the category which the input belongs to.

7.2 Training and training dataset

In this thesis, Self Organizing Maps are used to find similarities between the representations produced by the Deep Convolutional Neural Network InceptionNet. These representations ought to be similar for object in the same category and different for object that belong to different categories. The Self Organizing Maps shall catch these similarities and differences and, with the adequate training, learn to organize correctly the representations according to the network topology.

The network used is bi-dimensional, with neurons disposed in a lattice.

The parameters that we can use to modify the learning capacity of the SOM are basically three: the neighbourhood function (that is its parameters τ and σ), the dimensions of the map (that is the number of available neurons) and the number of iterations. We conduct different experiments, building different SOMs through the variation of these parameters.

For the training of the SOM we use all the representations provided by InceptionNet. The representations are 100 per class, with 10 classes available, for a total of 1000 examples. Each representation is an array of length 2048 with values included between 0 and 1.

All the SOMs built have a dimension of 20x30 with a total of 600 neurons. With this number of neurons and 10 classes available the net can create areas of maximum activation per class with about 60 neurons, that is the net specializes on average of 60 neurons to identify each specific class. The dimension of these areas, with adequately characterized inputs, allows one to discriminate well the set of inputs presented. Indeed, each cluster can be defined as the set of neurons that are BMU of the inputs that belongs to a specific class.

The above approach can be justified as follows: it is not possible to identify a single neuron per class that represents the BMU of all the inputs of that class, in fact the representations are distributed on arrays of length 2048 and those associated to objects in the same class are similar but not identical. In other words, considering all the inputs that belongs to a specific class, we can not find a single neuron whose activation is maximum for each input. Despite this, when we present a set of representations that belong to the same class, all the BMUs that are activated are located in a specific area, usually a compact cluster[1]. This fact derives from the properties of the SOM, in which, at the end of the training, neighboring neurons recognize similar inputs, that is they respond with the maximum activation to similar inputs.

During the training, once we set the dimension of the map, we change the neighbourhood function and the number of iterations in order to reach better performances.

[1]The measure of compactness used is fully described in the section 7.3.

7.3 Evaluation methods

We say that a SOM A is more performing than an other SOM B if presenting all the 1000 inputs to each network and finding all the BMUs associated with the inputs, the BMUs of the SOM A are organized in clusters that are more compact than those in SOM B.

To evaluate the compactness of the created clusters we compare the intra-cluster distance with the inter-cluster distance. The following definitions are the transposition for the SOMs of those in equations 6.2.3, 6.2.3 and 6.2.6.

Definition 5. *The intra-cluster distance for a class C is the mean distance between each couple of neurons that are BMUs of different inputs (i, j) that belong to the class C. In the bi-dimensional space the distance is computed as Euclidean distance.*

$$D_{intra}(C) = \underset{i,j \in C}{\text{avg}} \left(||BMU(i) - BMU(j)||_2 \right) \tag{7.3.1}$$

Definition 6. *The inter-cluster distance is the mean distance between each couple of neurons that are BMUs of a couple of different inputs (i, j). In the bi-dimensional space the distance is computed as Euclidean distance.*

$$D_{inter} = \underset{i,j \in \bigcup\limits_{k} C_k}{\text{avg}} \left(||BMU(i) - BMU(j)||_2 \right) \tag{7.3.2}$$

These two measures are compared to produce the following measure of cluster compactness:

Definition 7. *The compactness of a class C is the ratio between the intra-cluster distance of C and the inter-cluster distance.*

$$compactness(C) = \frac{D_{intra}(C)}{D_{inter}} \tag{7.3.3}$$

Given this measure of compactness, we can evaluate the compactness of the entire SOM as follow:

Definition 8. *The compactness of a SOM is the average between the compactness of each class in the SOM.*

$$compactness(SOM) = \frac{\sum\limits_{C \in SOM} compactness(C)}{|SOM|} \tag{7.3.4}$$

The compactness of the SOM can be evaluated as the average of the compactness of its clusters because each cluster contains the same number of neurons, or, better, contains the neurons that are BMUs of a set of inputs of established dimensions: each cluster contains the BMUs related to 100 inputs of a specific class.

Another important measure is the variance calculated on the compactness of the SOM. It gives us information about the structure of the clusters: if all the clusters in the SOM are equally compact, the variance is small, but if there are some clusters that are highly compact and others less compact, the mean can be small, but the variance is high.

The perfect SOM has a small value for compactness and a small variance, that is a SOM where all the clusters are well compacted.

The evaluation in terms of compactness is not the only evaluation done on the generated SOMs. A visual evaluation about the distribution of the BMUs associated with the training examples is useful in order to highlight some phenomenons that happen during the training. To these phenomenons we dedicate a section at the end of this chapter.

7.4 Experiments conducted increasing the number of iterations

We present now the results that we obtained with different iterations and different neighbourhood functions. For generating different neighbourhood functions, we change the σ value that denotes the radius of influence of the BMU while training on its neighbours.

We start setting $sigma$ to 15 (half of the map), and then reducing it to 10 (1/3 of the map) and 7 (about 1/4 of the map).

After setting σ to 15 and changing the number of iterations, we evaluate the resulting SOM through the measure of compactness introduced before. We changed the number of iterations from 30 to 70 with step 10; results are shown in tables from 7.4.1 to 7.4.5

Once we found the best number of iterations we change the value of $sigma$ in order to find the SOM with the minimum value of compactness.

A summary of the results is provided in Figure 7.4.1.

It is interesting to notice that all the values of compactness are included between 0.35 and 0.45. These figures show us that in all the SOMs the clusters are compact. We also notice that 0.35 is a very good compactness value and that it can not decrease further since examples, although similar, are not identical; as we said before, this fact cause the activation of many different BMU, that are near, but that cover an area of a non-zero measure.

Another interesting point is that the variance is small for the first four cases, demonstrating that the SOMs (if trained correctly) work well with examples that belongs to different classes, without much variation in the performance in the different classes.

Table 7.4.1: 30 iterations, neighbourhood 15 neurons

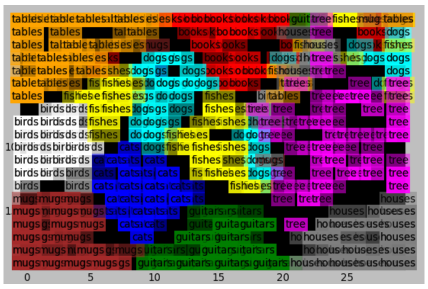
Class or SOM	Value	Image
Compactness(SOM)	0,4186097802	
Variance	0,0182804735	
Compactness(birds)	0,414669520403	
Compactness(books)	0,41233507486	
Compactness(cats)	0,272873668666	
Compactness(dogs)	0,639983705295	
Compactness(fishes)	0,681509985873	
Compactness(guitars)	0,309717573402	
Compactness(houses)	0,348907713127	
Compactness(mugs)	0,35087404157	
Compactness(tables)	0,362747762895	
Compactness(tree)	0,392478755708	

Table 7.4.2: 40 iterations, neighbourhood 15 eurons

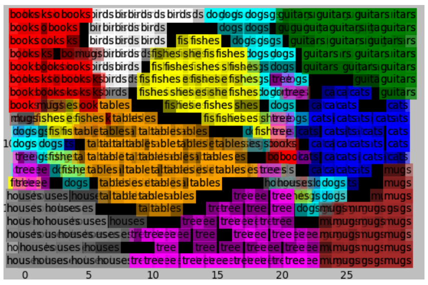
Class or SOM	Value	Image
Compactness(SOM)	0,3974437591	
Variance	0,016128844	
Compactness(birds)	0,371819928411	
Compactness(books)	0,440091233363	
Compactness(cats)	0,285452648165	
Compactness(dogs)	0,703315318334	
Compactness(fishes)	0,430913351691	
Compactness(guitars)	0,260357450472	
Compactness(houses)	0,354138068417	
Compactness(mugs)	0,34512493776	
Compactness(tables)	0,314180853801	
Compactness(tree)	0,469043800666	

Table 7.4.3: 50 iterations, neighbourhood 15 neurons

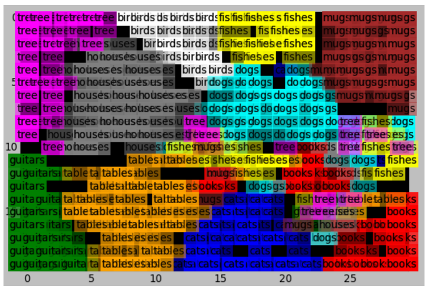
Class or SOM	Value	Image
Compactness(SOM)	0,4046378579	
Variance	0,0117312552	
Compactness(birds)	0,424341292795	
Compactness(books)	0,463187946188	
Compactness(cats)	0,291502843871	
Compactness(dogs)	0,426359424111	
Compactness(fishes)	0,530168854449	
Compactness(guitars)	0,279833838149	
Compactness(houses)	0,338638621954	
Compactness(mugs)	0,322595954946	
Compactness(tables)	0,356040148089	
Compactness(tree)	0,613709654501	

Table 7.4.4: 60 iterations, neighbourhood 15 neurons

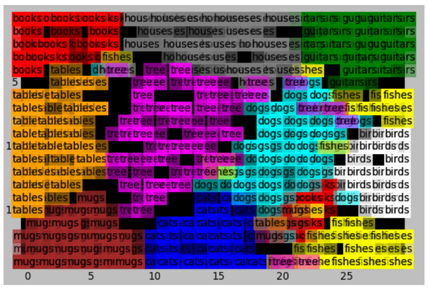
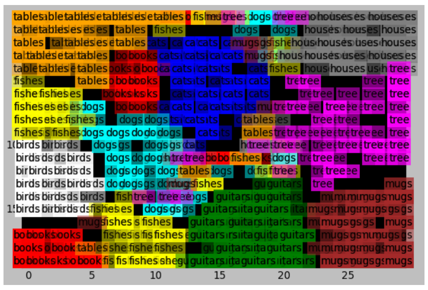
Class or SOM	Value	Image
Compactness(SOM)	0,3789768453	
Variance	0,0202175045	
Compactness(birds)	0,277967127718	
Compactness(books)	0,707085174696	
Compactness(cats)	0,275356392485	
Compactness(dogs)	0,373017639487	
Compactness(fishes)	0,526457253954	
Compactness(guitars)	0,231864799114	
Compactness(houses)	0,331638091976	
Compactness(mugs)	0,344267294807	
Compactness(tables)	0,306990535471	
Compactness(tree)	0,415124143329	

Table 7.4.5: 70 iterations, neighbourhood 15 neurons

Class or SOM	Value	Image
Compactness(SOM)	0,4376153727	
Variance	0,0287964717	
Compactness(birds)	0,364484287296	
Compactness(books)	0,723478863435	
Compactness(cats)	0,271718786213	
Compactness(dogs)	0,586176201511	
Compactness(fishes)	0,69316118601 8	
Compactness(guitars)	0,260085790569	
Compactness(houses)	0,309189115038	
Compactness(mugs)	0,361853124943	
Compactness(tables)	0,379975884657	
Compactness(tree)	0,426030487197	

We can see that the compactness decreases up to the 60th iteration and, after that, it raises again to a high value, with a high variance. Over the 60th iteration the SOM builds worse clusters.

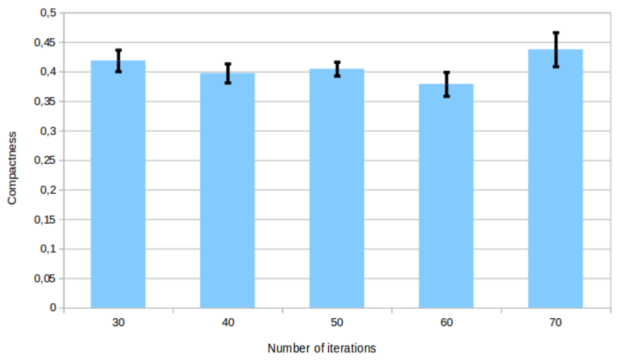


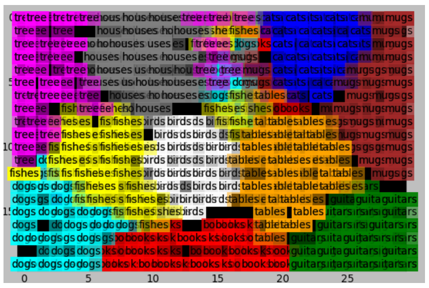
Figure 7.4.1: Compactness of the SOM generated using a neighbourhood function of radius 15 and an increasing number of iterations.

7.4.1 Experiments changing the value of sigma

Once established good values for the number of iterations, we conducted experiments changing the value of sigma, reducing the radius of influence of the BMU while training on its neighbours.

We chose to conduct these experiments only on the SOMs built with 50 and 60 iterations,

Table 7.4.6: 50 iterations, neighbourhood 10 neurons

Class or SOM	Value	Image
Compactness(SOM)	0,3771502593	
Variance	0,0032306859	
Compactness(birds)	0,361317072	
Compactness(books)	0,4581300188	
Compactness(cats)	0,3067065095	
Compactness(dogs)	0,3945602222	
Compactness(fishes)	0,435786119	
Compactness(guitars)	0,2893137268	
Compactness(houses)	0,3719960874	
Compactness(mugs)	0,3293897126	
Compactness(tables)	0,4355085186	
Compactness(tree)	0,3887946062	

because they provide the better performances considering their values of compactness and variance.

The results achieved show that the reduction of the σ leads usually to better performance, even if the results are not too distant from those reached using larger radiuses.

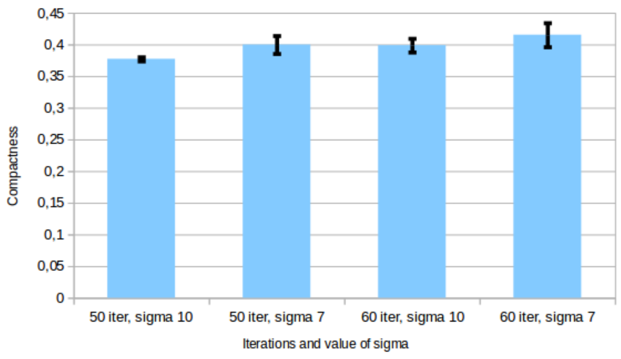


Figure 7.4.2: Compactness of the SOM generated using a neighbourhood function of radius 10 and 7, and 50 and 60 iterations.

7.5 Anomalous input

We define now the concept of anomalous input: an input can be consider as anomalous if, even if the classifier of the Deep Convolutional Network classified it correctly, its representation is very different from the representations of the other inputs that belong to

Table 7.4.7: 50 iterations, neighbourhood 7 neurons

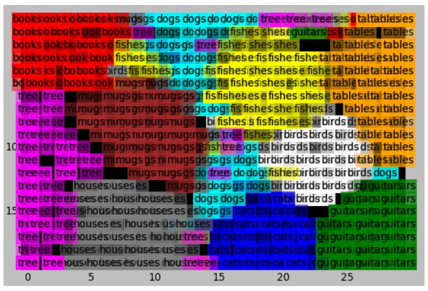
Class or SOM	Value	Image
Compactness(SOM)	0,399842415	
Variance	0,0141563338	
Compactness(birds)	0,346222342	
Compactness(books)	0,5297642465	
Compactness(cats)	0,3293184811	
Compactness(dogs)	0,5802840755	
Compactness(fishes)	0,3906031173	
Compactness(guitars)	0,2698167533	
Compactness(houses)	0,3652600269	
Compactness(mugs)	0,2949144468	
Compactness(tables)	0,3107721802	
Compactness(tree)	0,5814684802	

Table 7.4.8: 60 iterations, neighbourhood 10 neurons

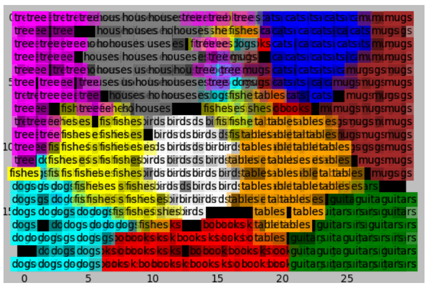
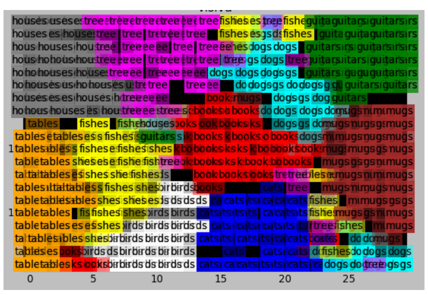
Class or SOM	Value	Image
Compactness(SOM)	0,3987739117	
Variance	0,0107765257	
Compactness(birds)	0,3486473029	
Compactness(books)	0,5879083622	
Compactness(cats)	0,3365473651	
Compactness(dogs)	0,5706448914	
Compactness(fishes)	0,4132484014	
Compactness(guitars)	0,3153214127	
Compactness(houses)	0,3091969301	
Compactness(mugs)	0,3670986206	
Compactness(tables)	0,3087982765	
Compactness(tree)	0,4303275545	

Table 7.4.9: 60 iterations, neighbourhood 7 neurons

Class or SOM	Value	Image
Compactness(SOM)	0,4152591434	
Variance	0,0189502565	
Compactness(birds)	0,4132393729	
Compactness(books)	0,5014098222	
Compactness(cats)	0,3173586197	
Compactness(dogs)	0,6445681667	
Compactness(fishes)	0,644833135	
Compactness(guitars)	0,2906406406	
Compactness(houses)	0,31989821	
Compactness(mugs)	0,3055926043	
Compactness(tables)	0,3078346907	
Compactness(tree)	0,4072161718	

the same class; this anomaly emerges in the cluster experiments and it is present again in the SOMs.

Differently from what happens with the other inputs, when an anomalous input of a specific class is presented, the neuron that produces the maximum activation (its BMU) is located in an area that is different from the area associated with the class of the input; these anomalous inputs will create problems in the next phase of our system, in which it is created the association between the category of the object (or, better, its representation) and its name.

These anomalous representations can be view as exceptions that the classifier built in the last layer of the Deep Convolutional Network is able to identify; unfortunately, the Self Organizing Maps are not able to identify those representations as exceptions and they locate those inputs in wrong positions; indeed, neurons that respond with the maximum activation in these cases are located in areas that are far from the correct cluster of neurons.

We must stress that an insufficient training can cause the wrong positioning of new inputs, whose representations produce the maximum activation in neurons that are far from the correct areas. This is not related to the problem of the anomalous inputs, which are identified by an unexpected neuron even if the net has been correctly trained.

7.6 Visual evaluation

At the end of this chapter we propose some visual evaluation that can be useful to understand better the SOMs view so far.

When we look at SOMs as showed previously, we can see that, if the SOM is correctly trained, a compact cluster is created for each class; this cluster contains most of the inputs that belong to that class.

The clusters have not the same shape: this depends from the diversity of the inputs in each class and from the training phase that, on the basis of its parameters, organizes in different ways the different classes of input.

As expected, clusters that are very compact, maintain this property of compactness even if the training phase changes (that is, even if the parameters used in the training phase change). This happens because if a cluster is compact then the inputs associated to that cluster (or that class) are much similar and, therefore, the similarity is catched in the form of compactness in each SOM.

Despite this, we can notice that some clusters are less compact, to the point that each of them could be considered as a conjunction of more little distinct clusters. For example, if we consider the SOM in table 7.4.7, we can see that the cluster associated with the class *dogs* (the cyan one) can be considered as the union of two distinct clusters. The same can be seen for the clusters associated with the class *fishes* (the yellow one) in almost all the SOMs, and in particular in that in table 7.4.4. This phenomenon does not happens for all the classes: for example, if we consider the classes *mugs* (brown), *tables* (orange) or *guitars* (green) we can see that those classes are well clusterized in each SOM.

Before giving an explanation to this phenomenon, it is worth recalling that all the representations used as inputs of the SOMs are provided by the Deep Convolutional Network, and, in particular, are only those that have been correctly classified by the classifier of that network. For this reason we can exclude that those little subclusters are the consequence of wrong representations.

We attribute this phenomenon to the structure of the representations built by the Convolutional Network: in classes like *mugs*, *tables* or *guitars* all the representations of objects in the same class are similar to a single prototype.

In other cases, for example when we consider the class *fishes*, the network identifies several prototypes. Indeed, the Deep Convolutional Network produces for that class different groups of representations, related to different groups of input images that share some features. During the training of the classifier, the network learns to identify these different groups of similar inputs as belonging to the same class and, consequently, at the end of the training it is able to correctly classify those inputs even if they are different between them.

When these different representations are presented to the SOM, the network is able to identify the similarities inside of each of these subgroups but it fails to find similarities between different groups that belong to the same class. For this reason the SOM creates a set of small distinct clusters associated to a single class.

On the one hand, understanding this phenomenon is useful to better understand the kind of representations that the Deep Convolutional Network produces, highlighting the fact that it is able to create different types of representations inside of a single class, when the images presented are different enough.

On the other hand, for the purposes of this thesis we prefer to have a SOM where all the representations that belong to a single class are well clusterized or where the subclusters associated to a single class are near enough to be considered as a single cluster. In chapter 9 we will explain in detail why we need a compact cluster for each class of objects.

7.7 Comparison with SOMs trained with AlexNet representations

As in the previous chapter, in this section we compare the results obtained using representations generated by InceptionNet with those obtained using representations generated by AlexNet.

In the previous chapter, conducting clustering experiments, we noticed that AlexNet builds worse representations with respect to InceptionNet, because the representations in a single class are less compact and there is more confusion between the representations that belong to different classes.

In order to visually evaluate the confusion between the representations provided by AlexNet and, consequently to evaluate the improvement produced by InceptionNet during the built of the SOMs, we built different SOMs with the representations provided by AlexNet.

To build these SOMs we maintain the dimension of each SOM to 20x30 neurons and we set different values for the two parameters presented in the previous section: σ (the value that denotes the radius of influence of the BMU while training on its neighbours) and the number of iterations.

In particular, we combine the value of σ 7, 10 and 15 with two different limits of iterations: 50 and 60. These parameters have been chosen on the basis of the parameters used and the result reached in the SOMs produced using the representations that belong to InceptionNet.

The training set contains all the representations extracted from AlexNet that have been correctly classified by this network and by InceptionNet; as mentioned previously, these representations are about 95 per class.

The chart in fig. 7.7.1 shows the values of compactness produced with different parametrizations. We can see that the best result is reached using a value for σ equal to 7 and 60 iterations. The value of compactness reached in this SOM is higher than that reached in most of the SOMs produced with the representations that belong to InceptionNet.

Indeed, in most cases, the SOMs produced so far have values of compactness under the value of about 0.42. All the SOMs produced with the representations provided by AlexNet produce values of compactness beyond that limit.

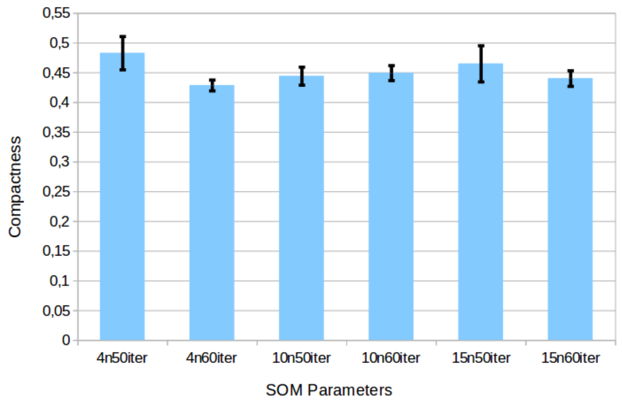


Figure 7.7.1: Compactness of the SOM generated using different value of σ and different number of iterations and the representations provided by AlexNet.

The cause of those values can be found in the confusion between the representations that belong to different classes. The inability of AlexNet to produce much similar representations for the objects in the same class and much different representations for those in different classes compromises the capacity of the SOM of grouping correctly the representations.

As said before, for the model of word learning simulation that we want to build, we need that the representations that belong to a certain class are correctly grouped in a restricted area of the SOM. For this reason, the use of InceptionNet to create the representations is preferable than the use of AlexNet.

In order to visually evaluate the capacity of the SOMs to correctly group the representations, we report the SOM produced with σ equals to 10 and 50 iterations, which reaches the best value of compactness between the SOMs produced.

We can notice that, with respect to the SOMs produced in the previous sections, the representations are more sparse: the clusters are defined by a core that contains enough representations, but for each class we can find a lot of representations that are sparse in the other areas of the SOM, without forming other little clusters like in the previous SOMs.

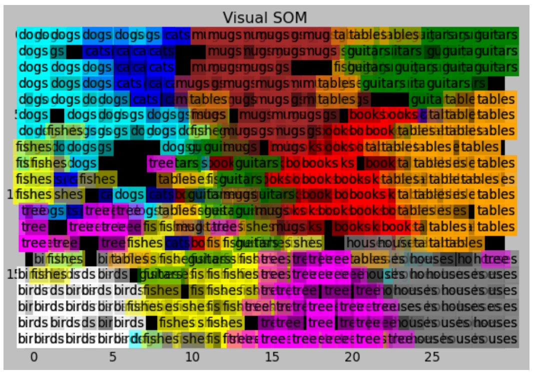


Figure 7.7.2: SOM generated using the representations provided by AlexNet, with σ equals to 10 and 50 iterations.

From these SOMs we can bring out an other characteristic of the representations generated by AlexNet: representations that belong to classes that are topological related are much similar between them, and are difficult to distinguish.

Indeed, if we consider the classes *cats* (blue) and *dogs* (cyan) we can see that they are placed in near areas and, more important, they are overlapped. This means that AlexNet struggles to built distinct representations when the topological similarity between two classes is strong.

Because of the size and the sparseness of these SOMs and of the overlap between topologically related classes, we can not use them in our model.

From the next chapter we will use the SOMs built with the InceptionNet representations in order to simulate the last step of the word learning process: the association between each class of objects and its name.

Chapter 8

Auditory representations

So far we dedicated our attention to the building of a good visual representation, starting from a real image.

On the other side, during the process of word learning we need a good auditory representation: indeed, the correct association between an object and its name concerns the correct association between the visual representation of an object (that is related not to a single object but to the entire category which the object belongs) and the auditory representation of an object.

We can imagine that the auditory inputs that lead the word learning process are a set of sounds that contains words and, in particular, the names of the objects that the infants is learning. Each sound is segmented by the infants into its components, theorically the words that compose the sentence in the sound, and each components is associated to an auditory representation.

At this point the infant learns to associate each auditory representation to the correct meaning, in this case, the visual representation presented together with the sound.

The problems in the process of extraction of the representations are mainly three:

- the first problem is that we need a system able to segment the auditory stream into its components, and to extract the sounds of a single word;
- the second problem is that the system must be able to understand which of the words that compose the auditory stream is the one that we want to associate to the visual representation;
- the third problem is that the system must be able to create representations that are highly similar when the sounds presented contain the same word, even if the word is pronounced by different persons; for example if the same word is pronounced by people of different sex or age or by people with strong accents.

The first two problems are easily solvable if we suppose that the input stream contains only the word of interest. This is a realistic assumption: when a person indicates an object to an infant in order to teach him the word associated to the object, she says only the name of the object or few other words.

The third problem is more complex and we need a system of speech recognition in order to create an unambiguous representation for each object category.

The focus of this thesis is to simulate the word learning process using real images. Unfortunately implementing word acquisition through realistic audio is all but trivial, in fact a speech recognition system needs a large database of sounds of the words of interest and implementing recurrent neural models that are not simpler than the deep networks studied in this thesis. For all these reasons we decided to simulate this part of the system with artificial representations.

For each object category we build a prototype of the auditory representation: each prototype has length equals to the number of used categories, and contains all zeros except for a single one whose position indicates the class of membership.

After the evaluation of the difficulties encountered during the building of a visual representation that must be similar for images that belong to the same class and enough different for objects that belong to different classes, we know that we are introducing a great simplification.

In practical terms, we shall assume that a good speech recognition system, is able to create a unique representation for each class (where a class is a word).

As done for the visual representations, we need to locate the auditory representations into a SOM. Indeed, for the final step of the word learning process we need that the two types of representations (visual and auditory) are arranged into two similar architectures. The association between two representations is realized through the construction of correct associations between the different areas of the SOMs.

The SOM built for 10 classes has dimension 20x30 as the analogous visual SOM.

The training set of the SOM is created artificially starting from the prototype defined above: for each class we build a set of 100 representations that are equal to the prototype of that class. The training process iterates for 70 times on the training set, with a value of σ equal to 7, and organize the representations in the different areas of the SOM.

Given that all the representations that belong to a specific class are equal, the SOM is able to recognize a single BMU for the entire class. For this reason, the final SOM for the auditory inputs is similar to that showed in figure 8.0.1, where the areas that contains all the BMU associated with single class are actually single neurons, that are the BMU trained to recognize all the inputs that belong to those classes.

We test different value of iterations but, as it should be obvious, results are similar for

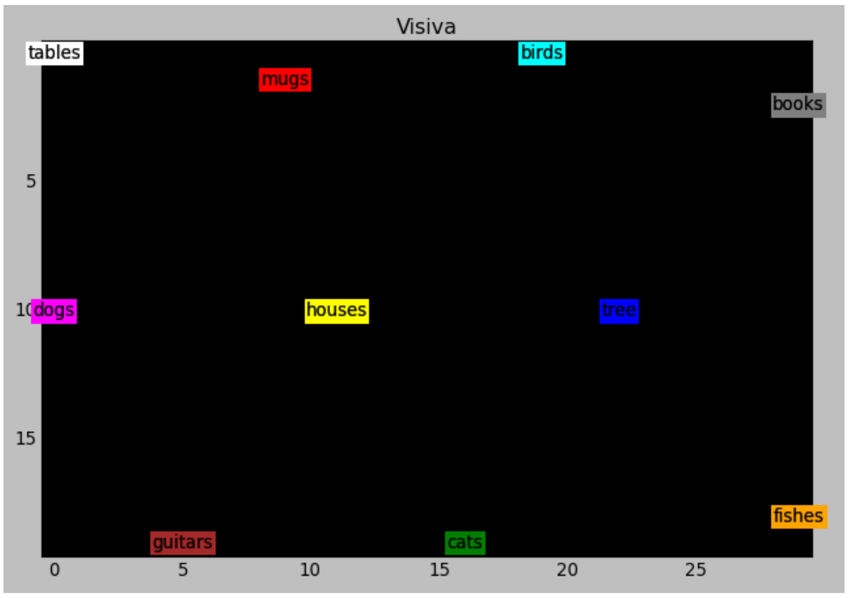


Figure 8.0.1: Auditory SOM.

each value, seen that all the representations that belong to a single class are identical.

Chapter 9

Word learning

During the first year of life, infants develop joint attentional skills that are important for promoting language development. Many studies demonstrate that by 12-18 months, infants turn their heads to the correct side when following an adults' direction of gaze.

The amount of joint attentional activities between infants and their caregivers is a predictor of word learning abilities: for example pointing behaviour has been shown to influence positively the process of words acquisition.

Furthermore, infants' vocabulary acquisition proceeds faster when parents are sensitive to their infant's own focus of attention when they speak.

For these reasons we can say that joint attention plays a crucial role in early word learning since it enables the infant to match words and their referents through gaze following or pointing behaviour.

In particular, the joint attention allows a supervised learning process: first the infants learn to group independently objects (the visual stimuli) and sounds (the auditory stimuli) into categories; then, in the association phase, the joint attention allows the infants to correctly associate an object with its name when they are presented together, that is when the infant focus on the object and in the same time hears the sound of the word associated with the name. In this case the object category and the name category work as labels one for the other.

9.1 The reference model

The model proposed in this thesis is based on that proposed by Mayor and Plunkett [14].

Mayor and Plunkett [14] proposed a neuro-computational model using self-organising maps that accounts for the emergence of taxonomic response in early word learning.

The model consists of two separate SOMs that categorize respectively the inputs received from the visual field and the auditory stimuli. The inputs are artificially built, indeed the visual stimuli are random dot patterns that are similar for objects in the same category and different for objects in different categories (they maintain a family resemblance), and the auditory stimuli are extracted from the acoustic signatures extracted from sounds produced by a female native speaker; the acoustic signatures are manipulated in order to create simple inputs, which maintain a high level of similarity for sounds that belong to the same class (low intra-class variation) and a high level of difference for sounds that belong to different classes (high inter-classes variation).

In the model proposed by Mayor and Plunkett [14], the SOMs are trained separately in an unsupervised way in order to have two models able to correctly categorize the two types of inputs presented.

In this first phase the model simulates the process of creation of categories that takes place in the infants' brains: by the time that they are able to engage in joint attentional activities, their perceptual systems are already well-organized.

The joint attention activities are simulated presenting simultaneously an object (the visual stimuli) and its label (the auditory stimuli): when the two representations are presented, each SOM is activated consequently and the synapses that connect the two maps are modulated, in order to correctly associate the representations presented as inputs.

The updating of the synapses follows the Hebbian rule[1], with which neurons that are frequently strongly activated together are connected with strong weights whereas neurons that are weakly activated together are connected with weak weights.

The association between the object and its correspondent sound pattern will be generalised, automatically building associations between all the objects in a specific category to all the relative sound patterns thanks to the properties of the SOMs. Indeed, the SOMs activate, for each representation presented as input, not only a single neuron (the BMU) but also an area of neighbours corresponding to the category, introducing a generalization in the association built. The category is then correctly associated with its name, through a learning process that uses only few examples of that category.

In order to test the correctness of the associations built between the two SOMs, we can evaluate the propagation of the activations when a single input (visual or auditory) is presented to the model. If, for instance, we provide a visual input, the activation is propagated, through the Hebbian connections, towards the auditory SOM, which is activated consequently. If the Hebbian connections have been correctly modulated, the activation of the auditory SOM is similar to the activation produced in the same SOM by an auditory input that belongs to the same category of the visual input presented.

[1]The Hebbian rule will be presented in detail in the next section. Here we just want to give an idea on how it works.

9.2 Overview of the model proposed

The last part of the model built for this thesis follows the idea presented in the previous section: as in the model of Mayor and Plunkett [14], we use two SOMs in order to categorize the objects and the words presented as inputs and the Hebbian connections in order to create an adequate bond between the two SOMs. Despite this, the inputs used are substantially different and also the Hebbian connections are created in a different way.

We have already seen in the previous chapters what types of inputs have been used and how they have been categorized in the SOMs. Once the SOMs have correctly learnt to categorize the representations presented, we start to create associations between the two SOMs.

As done in the reference model, couples of inputs that belong to the same category are presented simultaneously to the two SOMs, and their activations are observed. At this point in each SOM we can see some areas with strong activation, some other areas with medium activation and the remaining parts of the SOM not activated at all. Each neuron of the first SOM is connected with each neuron of the second SOM and the value of those connections are modulated in order to tie with strong connections the areas that are simultaneously strongly activated; indeed, each connection is updated proportionally to the level of activation of the neurons that it ties.

As in the reference model, the connections created presenting couples of representations are generalized to the correspondent categories on the basis of the properties of activation of the SOMs.

At this point we understand how much is important that the representations (in particular those visual) are well clusterized into the SOMs: in order to create an association between a category and its name we present to the two SOMs two representations, respectively visual and auditory; when the visual SOM is activated, the area with the stronger activation identifies the category to which the input is associated, but, if the BMU of the visual representation is wrongly located in the SOM, the association consequently created is wrong.

For this reason we need that all the representations (or the most part of them) are well clustered into the SOM. This problem do not exist for auditory representations in our settings, given the simplified auditory model we adopted, while it is very significant for the visual representations.

In particular, the capacity of the visual SOM to correctly clusterize the representations presented as inputs, influences strongly the capacity of the system to simulate the word learning process. In order to reduce this problem we will propose different solutions in the next sections.

From now on we will describe in detail the algorithm used to update the synapses when a couple of representations is presented.

9.3 Updating of the connections

As said previously, during the training of the connections between the two SOMs, the weight assigned to each connection that ties two neurons is proportional to the level of activation of the neurons involved.

The Hebbian rule, in its theoretical description, concerns exactly this principle of proportionality: given two neurons i and j (with their levels of activation a), the weight w_{ij} that connects them can be described as follow:

$$w_{ij} = a_i a_j \tag{9.3.1}$$

Indeed, the value assigned to w_{ij} is higher, the higher are the levels of activations of the neurons involved. The rule used in our thesis follow this idea, but it realizes it in a different formula.

The weights associated to the synapses S are first randomly initialised with a normal distribution centred on $\frac{1}{\sqrt{S}}$ and with a standard deviation of $\frac{1}{\sqrt{1000*S}}$.

Then, we present to the SOMs multiple couples of representations associated to the different categories and, at each presentation (identified by n), we update the synapses as follow:

$$w_{ij}(n+1) = w_{ij}(n) + 1 - e^{-\lambda a_i a_j} \tag{9.3.2}$$

λ is the learning rate (fixed to 10 in our experiments), whereas a_i and a_j identify the levels of activation of the neurons involved.

When both the neurons are strongly activated the value of the exponent is a high negative number that produces a value near to 0. Therefore the previous weight is updated, increasing it of a value near to 1. Vice versa, if the activations of both the neurons are small, the exponent is a negative number near to 0 and the exponentiation produces a value near to 1. Therefore the previous weight is updated increasing it of a value near to 0.

For each neuron, its activation level is inversely proportional to the error between the weights that connect the input to that neuron and the input presented. In particular, the level of activation is calculated as follow:

$$a_i = e^{-\frac{q_{ik}}{\tau}} \tag{9.3.3}$$

where τ is a normalization constant (fixed to 0.5) and q_{ik} is the quantization error between the weights of the neuron i and the input presented x_k, defined as it follows:

$$q_{ik} = \frac{\sum_{j=1..N} |x_{kj} - w_{ij}|}{N} \tag{9.3.4}$$

where N is the length of vectors x_k and w_i.

The quantization error calculated is small if the representation in input x_k is similar to the weights of the neuron i, it is big otherwise.

If the error is small, the activation of the neuron i is a value near to 1, otherwise, if it is big, the activation has a value that approach zero exponentially fast with the magnitude of the error.

Each time that we present a couple of inputs to the SOMs, the activations of all the neurons are calculated as described in equation 9.3.3 and each weights between two neurons is updated consequently as described in equation 9.3.2.

At the end of the training, all the weights are normalized so that the following equation is respected:

$$\sum_{i,j} w_{ij}^2 = 1 \tag{9.3.5}$$

The algorithm for updating of the synapses presented so far is the one proposed also in Mayor and Plunkett [14].

9.4 Optimizations of the updating algorithm

In this thesis, the complexity of the representations used as input in the visual SOMs, and consequently the limits to the capacity of the SOMs to correctly clusterize the inputs, makes the algorithm described not sufficient.

Indeed, if we try to create connections between the SOMs relying only on the algorithm proposed by Mayor and Plunkett [14], the created bond areas are little significant and not always correct.

These problems have origins in some limitations imposed by Mayor and Plunkett [14] in their model.

9.4.1 Updating all the synapses

The first limitation is that they update, at each iteration, only the synapses related to the BMUs, indeed when they present the pair of representations to the relative SOMs, they proceed in a two steps process: first focus on the visual representation and update the weights between the visual BMU and all the neurons in the auditory SOM; then they focus on the auditory representation and update the weights between the auditory BMU and all the visual neurons.

The problem is that with this process we limit the amount of useful informations exploited during the updating.

Indeed, if we, during the updating of the synapses, consider the whole map and its activations, we use more time, but we will consider all the areas of strong activations in the two maps. As the evaluation of the SOMs done in the previous chapter shows, the same category can be splitted in more little subgroups, and this may limit the learning if only the BMUs are taken into consideration. When we present a representation, we can find more than a single area of strong activation, therefore, if we want to catch all these informations during the process of updating the synapses, we need to consider all the activations in the map and not only the BMUs.

These information, given the complexity of the representations and the way in which they are arranged on the SOMs, is useful in order to create more significant connections between the visual and the auditory maps. Furthermore, they are useful in order to extend the bond between large areas and not only between single neurons.

9.4.2 Presentation of more pairs of representations during the training

The second limitation, that was an advantage in the reference model, is that in that model the authors present for each category a single pair of inputs, showing that the association between the category and its name is correctly created after a single presentation.

Considering the visual representations that we use, to reach the same result is impossible in our model basically because of the anomalous inputs. Indeed it is not certain that the updating of the synapses, done using a single representation per class, connects strongly the correct areas.

If we consider for the training of the synapses a representation that is an exception, the synapses built connect strongly areas that do not belong to the same class; otherwise, if we consider a representation that belongs to a subgroup of the correct class, the synapses built are strong only for that subgroup and they do not reflect the totality of the examples in that class.

The solution is to use several presentations for each class. In particular we conduct different experiments increasing the number of presentations, evaluating the connections consequently created. As we will see in the next section, the presentation of more representations for each class helps solve the previous problems.

Indeed, if in the set of representations of a specific category used for the training there is an exception, the error introduced by the exception is mitigated by the other representations. Furthermore, if the set used for the training is chosen randomly, the types of representations in the training set follow the distribution in subgroups of the representations of that category, including some example for each subgroup existent in the SOM. In this way, the association created between the SOMs for a specific category can be considered as an association that reflects the differences present in the category.

9.4.3 Suppression of the low activations

A third problem is not directly connected to the limits imposed by Mayor and Plunkett [14], but emerges from the complexity of the representations used.

In order to understand this problem we start evaluating the activations that are produced in the auditory SOM.

As we can see in figure 9.4.1, when we present an auditory representation to the auditory SOM, the net is mainly not active, but it is strongly activated in the area associated to the category of the representation provided. In particular, since all the representations that belong to that class are identical, during the training the SOM associates the same area to all the representations, therefore there is not a slow decrease in the activations, but there is an area with strong activation and the remain part of the net is not activated.

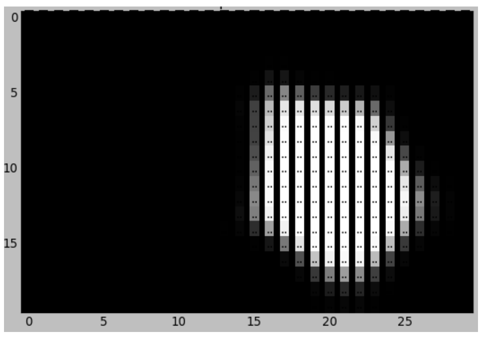
This means that when we create connections between the two SOMs, all the connections that involve the inactive neurons are not updated, because the product between the activation level of any neurons in the visual SOM and the activation level of these neurons is 0.

Instead, when we present a visual representation to the relative SOM, the activation produced is different. In particular, since the training set of this SOM is made by examples that have enough differences, even when the representations belong to the same class, it is difficult to find areas of the SOM that are completely not active. Indeed, all the map is activated with different levels of activations. This happens because of the exceptions that are present in each class and because of the variety of types of representations that compose each class and that form subgroups of activations in the SOM.

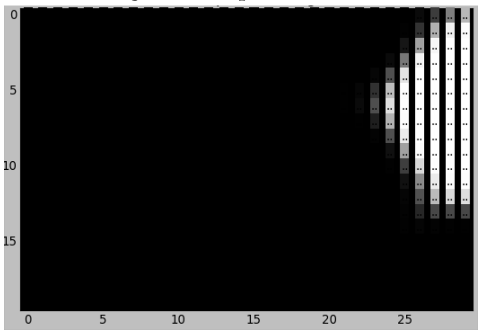
This did not happen in the model proposed by Mayor and Plunkett [14], because the stimuli used to build the SOMs and to update the synapses were more orthogonal.

Furthermore, when we use classes that are topologically similar (for example the classes *dogs* and *cats*, the neurons that are strongly associated with one class (for example the class *dogs*) are mediumly activated when the other class is presented (the class *cats*). We can see in figure 9.4.2 that the neurons that are strongly activated in the second map, are also mediumly activated in the first map.

For this reason when we present a visual stimulus and we use the consequent activation in order to create connections between the two SOMs, all the neurons influence the up-



(a) Using an example of the class *cats*



(b) Using an example of the class *dogs*

Figure 9.4.1: Activations of the auditory map, using examples that belong to the classes *cats* and *dogs*

dating of the weights, if only for a small value. Furthermore, if we use classes that are topologically related (like *dogs* and *cats*), the influence of the correspondent neurons is strong and can lead to significant errors.

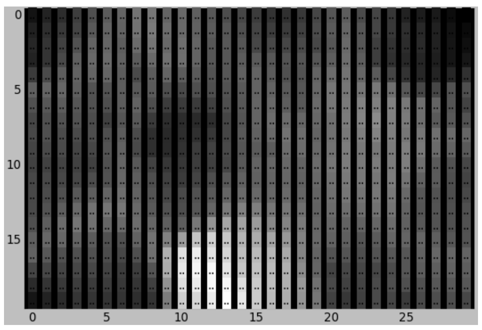
Therefore, if we present a couple of stimuli (visual and acoustic) that belong to a class in order to update the connections between the areas associated to that class, we do not update only those connections, but all the connections that are bound to the restricted activated area of the auditory SOM with all the neurons in the visual SOM.

This is an unexpected phenomenon, that creates unwanted effects during the training of the connections inter-SOMs.

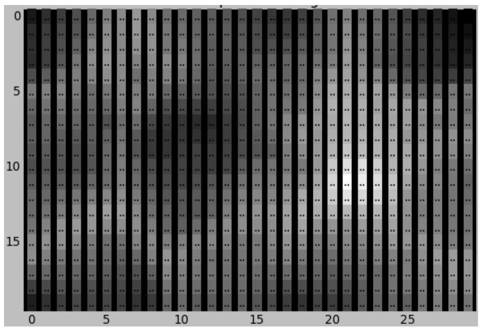
This is a consequences of the solution found for the first problem: if we update the weights only for the BMU, it is not relevant the distribution of the activations in the rest of the map; indeed, it is irrelevant if the other neurons are all off or if they have all a strong activation (a bit less strong than the BMU), their weights are not updated.

Instead, if we update the weights of all the neurons of the SOM, their activations are important and influence the resulting connections.

Despite this, it is too important to consider more than a single neuron (the BMU) when



(a) Using an example of the class *cats*



(b) Using an example of the class *dogs*

Figure 9.4.2: Activations of the visual map, using examples that belong to the classes *cats* and *dogs*

we update the weights, therefore we accept the presence of this third problem, and search for another solution.

As said before, this phenomenon creates unwanted effects during the training of the connections inter-SOMs: since that we present multiple couples of representations for each class, we can use a large number of representations during the training phase. If each of these representations activates only a limited area of the SOM and, consequently, influences only the weights relative to that area, the connections between the SOMs are well built; instead, if each of these representations activates the whole map, even with small activations, the consequence is that at each iteration the connections are updated (increasing their values) between all couples of neurons. At the end of the training the sum of this little increases make easier the propagation of the activations between the two maps, even between areas that are not correlated. Therefore if we present an input of any class to the visual SOM we do not know beforehand what is the auditory area associated by the system.

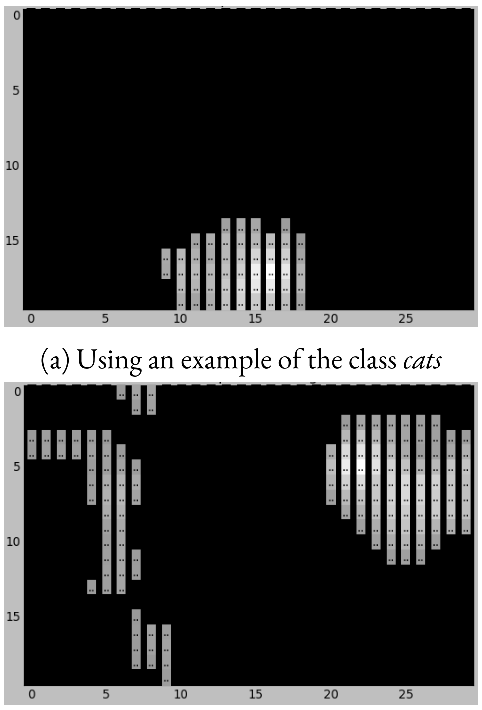
In order to solve this problem we conduct experiments suppressing activations below a certain threshold. The idea is that when we present a visual stimuli, the activations are more strong in some areas and weak in others: the areas with strong activation are those associated to the class to which the input belongs (if the input is not an exception), whereas the areas with weaker activation are most likely related to a different class. In our solution, we maintain only the activations that are significant for that input and, consequently, for that category.

At the end of this process we can update the synapses between all the neurons of the two maps, knowing that only the significant neurons (those that survived to the thresholding) influence the update of the synapses.

In order to achieve this result, each time that an input is presented to the visual SOM, the activations are calculated, then they are normalized between 0 and 1 (in order to have a better control on their values) and all the values under 0,6 are set to 0.

We tested different values for the threshold and 0.6 resulted in a good compromise between the amount of useful informations that we maintain and the amount of useless informations that we discard.

We can see in figure 9.4.3 that at the end of the process of suppression the most part of the map is not active, but the active area is the one mostly associated to the representation presented as input.



(a) Using an example of the class *cats*

(b) Using an example of the class *dogs*

Figure 9.4.3: Activations of the visual map, using two different examples that belong to the classes *cats* and *dogs* and the suppressing all the values under 0,6.

Furthermore, the suppression brings out all the significant activations in the SOM: as we have seen in the previous chapters, some classes are characterized by a significant uniformity in their representations (for example the class *cats*), therefore the suppression brings out a single continuous block of neurons. Instead, in other situations, in particular when the class is characterized by more disjoint clusters and when the representation presented belongs to one of these clusters, the suppression maintains activated more areas of the maps, trained to identify that specific class.

This is an important property of the suppression, because, differently from the algorithm proposed by Mayor and Plunkett [14], it allows us to consider all the areas that are significant for a specific class and not only the BMU of the input presented, but, at the same time, it allows us to discard the areas that are not significant and that can lead to errors.

The optimizations presented here help us to achieve better performances during the phase of association inter-SOMs, using all the significant informations acquired so far and discarding all the misleading pieces of information that could distract the system from the correct solution.

9.5 Evaluation of the associations

The tests have been conducted in order to understand the capacity of the system to create correct associations between the two maps. In order to understand the knowledge learnt by the system, we evaluate the propagation of the activations following these steps:

1. we present a representation that belongs to a specific class to the one of the SOM (for example the visual SOM);
2. we calculate the activation of the SOM;
3. we propagate the activation through the synapses towards the other SOM (in this example the auditory SOM);
4. we calculate the activation of the second SOM at the end of the propagation;
5. we evaluate the activation of the second SOM in order to understand its correctness.

So far we have seen how to solve the point 1 and 2, and we present now the algorithm used to propagate the activation from a SOM to another during the test and, consequently, how to calculate the level of activation of the final SOM.

9.5.1 Propagation of the activation

The propagation can be done in two different ways: in the first one we consider only the activation of the BMU in the first SOM and we calculate the activation of all the neurons in the second SOM as the product between the value of the BMU and the values of the weights (the synapses) that connect the BMU to the neurons of the second map. Let $a^{(1)}$ be the activation vector for the first SOM (i.e., $a_i^{(1)}$ is the activation value of the i-th neuron in the first SOM) and let $a^{(2)}$ be the activation vector for the second SOM. Therefore the level of activation of the final SOM is calculated as:

$$a^{(2)} = a_{BMU}^{(1)} w_{BMU} \tag{9.5.1}$$

This method is fast and catches the activation of the BMU, therefore it considers the most important information. At the same time it does not use all the available informations, that is all the activated area, therefore it does not take advantage of all the information present in the first SOM. This method is useful, in particular, when the activation of the original SOM is not suppressed, because it discard all the unexpected activations.

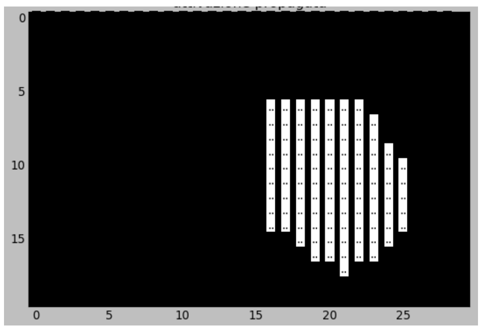
In the second method we consider the activation of the entire first map: for each neuron in the first map we consider the contribution brought to the activations of the neurons in the second map as the product between the value of the neuron considered and the weights that connect that neuron to all the neurons of the second map. The final value of activation of each neuron in the second map is calculated as the sum of all the contributions brought to that neuron by all the neurons of the first map. If we let W denote the matrix of the weights, i.e., W_{ij} is the weight between neuron i of the first SOM and neuron j in the second one, then the final activation is calculated as:

$$a^{(2)} = W \times a^{(1)} \tag{9.5.2}$$

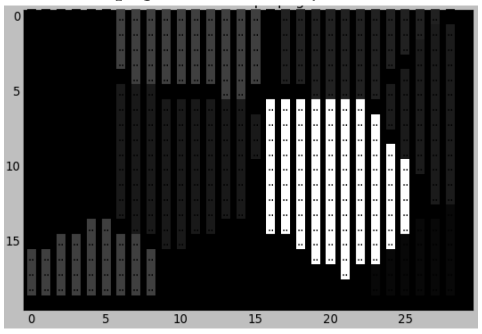
As a difference from the first method, using this second method the final activations must be normalized if we want to maintain values between 0 and 1.

This second method is usable only when the SOM is suppressed, otherwise the activation propagated includes a lot of informations that are not significant for the class considered.

In figure 9.5.1 we can see the activation of the auditory SOM, after the propagation from the visual SOM, obtained using the two methods with an example of the class *cats*. In order to generate these images the training of the synapses has been done with a single couple of examples for each class, and the learning has been tested presenting a representation of the class *cats* to the visual SOM. The images give an idea of the results obtained, but can not be considered as reference images because the learning depends from a lot of factors, as the number of pairs used during the training, the characteristics of the representations used during the training and the example used during the test.



(a) Propagation with only the BMU.



(b) Propagation considering the whole visual SOM.

Figure 9.5.1: Propagated activations in the auditory map, using two different methods of propagation.

As we can see the activations are similar, but if we consider the entire visual SOM, some other areas are activated on the basis of the activation of the visual SOM and of the learned synapses.

9.5.2 Methods for the evaluation

The evaluation of the activation is the final step to understand the capacity of the system to create correct associations between the objects and their names.

Actually, the evaluation of the learning capacity of the model involves multiple steps and this complicates very much evaluating it: if we have an image and we want to rebuild the name associated with it, we build a representation of the image, then we present this representation to the visual SOM, we calculate and propagate the activation towards the auditory map, we rebuild a representation starting from the activation of the auditory SOM and, at the end, we rebuild the auditory input from its representation. If the auditory output is correct the system has correctly learned the association between the object and the name. The system can be used also in the other direction, from the auditory input to the visual output.

Therefore, the goodness of the propagated activation should be evaluated by rebuilding the representation associated with that activation and, from that, rebuilding the original input, following the inverse process used during the generation of the activation. At the end, the evaluation should be done on the reconstructed input.

The problem is that to rebuild an input starting from the activation of a SOM is a complex process: first we have to reconstruct the representation that origins the activation, and then we have to reconstruct the input that generates that representation. In particular for what concerns the visual part, in which we have to go through the Deep Convolutional Network from the top to the bottom, the last part of the process presents a lot of difficulties, and it is not even usually possible since the representation of each input is the result of a series of abstraction that can not be inverted.

Therefore, we chose to simulate this last step of evaluation not by evaluating the correctness of the reconstructed input (or of its representation) but evaluating directly the activation of the SOM.

Even if we choose to evaluate only the activations of the SOMs we have to understand how we can evaluate their correctness.

The idea is that the more the activation propagated into a SOM starting from an example of a certain class is similar to the activation generated by an example of the same class in that SOM, the more the activation propagated is considered as correct.

For example, if we present an input of the class *dogs* to the visual SOM and we propagate the activation into the auditory SOM, we can consider the activation propagated as correct if it is enough similar to the activation generated in the auditory SOM by an example of the class *dogs*.

The computation of the similarity needs two parameters: the reference activation from which we calculate the distance of the other activations and the measure of distance.

For what concern the auditory input, the reference activation is easy to find, because all the used inputs that belong to a class are equal and produce the same activation. For what concern the visual input, the problem is harder since a single input will activate many units in different ways.

The measure of distance could be the average difference between the activations of the maps, or the distance between the BMU of the activation propagated and a reference BMU for each class. Again, these measures are easy to apply to the auditory SOM but more difficult for the visual SOM.

Actually, the way in which the synapses are built during the training helps to solve this problem. Indeed, the synapses are bidirectional, therefore the synapse that connects a visual neuron with an auditory neuron has the same weight if we propagate the activation from the visual to the auditory SOM as if we propagate the value in the other direction.

For this reason, in order to evaluate the correctness of the connections created we can evaluate them only in one direction, the easier: from the visual SOM to that auditory.

For each class we build the reference auditory activation as the activation produced by any auditory representation of that class.

The similarity between the propagated activation and the reference activation is calculated with two different methods:

Distance from the BMU: we find the more active neuron (the BMU) and we calculate the distance between this neuron and each reference BMU. The reference BMUs are the more active neurons in the reference activations of the available classes. Given that the neurons are placed into a bi-dimensional space, the distance is calculated as the Euclidean distance. The reference BMU nearer to the considered BMU identifies the class associated to the propagated activation.

More active class: we consider the fact that at the end of the training of the auditory SOM, when we present an auditory input, the activated area does not form a circle centered in the BMU, but can cover areas of different form. In particular, each BMU activated by a certain input is not necessarily nearer to the BMU of its class than to the others. For this reason to associate the BMU of a propagated activation with the nearer reference BMU is not always correct.

The idea followed with this second method is that considering the BMU of a propagated activation, the associated class is the one that produces the maximum level of activation in that neuron; therefore, we consider the reference auditory activations for each class and we find the class that produce the maximum value of activation in that neuron.

Seen that the areas of activation for each class in the auditory map surround their BMU the results achieved with the two methods are similar, even if the second gives us more precision.

9.5.3 Evaluation on the training set

In order to evaluate the correctness of the associations created we conduct some experiments following the considerations made so far.

In particular, training the connections starts using the two SOMs completely and independently trained as described in the previous chapter.

The we present an increasing number of pairs of representations (auditory and visual) for each class, and we measure the resulting performances.

We built two training sets: one containing the visual representations, and the other one containing the auditory representations.

For building the first training set, we start from a set of visual representations that includes, for each class, 100 representations that have been correctly classified by the Deep Convolutional Neural Network. From this set we built the training set extracting at random NC representations for each class (NC being the number of pairs we want to use).

Analogously, the auditory training set is composed of NC auditory representations for each class.

The training is done presenting together each pair of representations that belong to the same class to the SOMs, calculating the relative activations and updating consequently the synapses. This process is done for all the NC available couples of representations for class.

In order to achieve a result not dependent from the set of visual representations used for the training, we repeat the training and the consequent evaluation on 1.000 different training set: in each iteration we built a new random visual training set made of NC representations per class, and we use it for the training. The auditory training set is always the same because all the representations that belong to a certain class are equal.

At the end of each iteration we conduct the evaluation using both methods presented in the previous section (distance from the BMU and more active class). As said previously, the evaluation is done only by propagating the informations from the visual to the auditory map; therefore the visual representations are presented to the relative SOM, the activations are calculated and propagated to the auditory SOM and here we find the BMU. In particular, the test is done using all the available representations: the original set is composed of 100 representations for each class and we present all these representations finding and evaluating the correspondent propagated BMU in the auditory SOM.

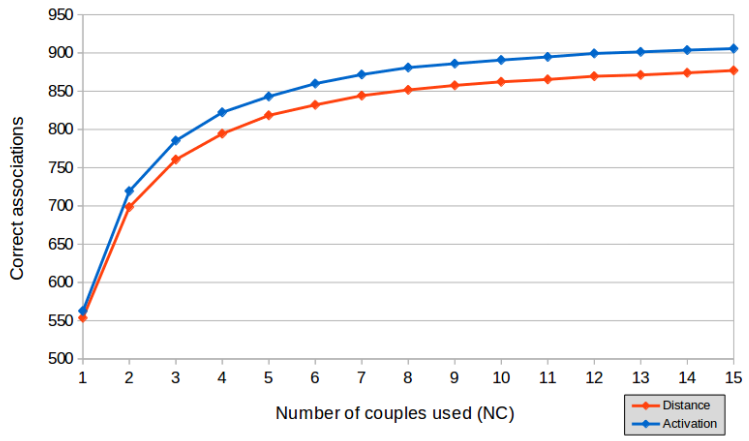
Each time that the class of the propagated BMU correspond to the class of the visual representation used as input, the score of that iteration of training is increased of 1. At the end of the training, a score included between 0 and (100*number of classes) is associated to each iteration. 0 means that with the connections built in that iterations we are able to correctly associate 0 visual representations, that is that the associations are completely wrong; (100*number of classes) means that with the current connections we are able to associate each visual representation to the correct auditory class.

The results achieved in each iteration are averaged in order to produce an information independent from the training set used.

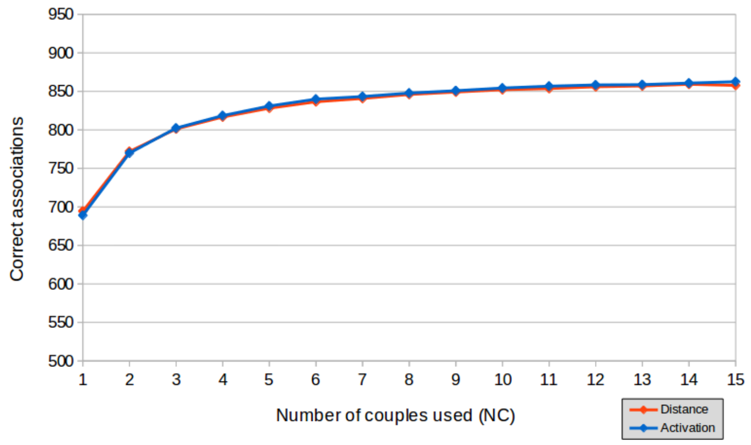
The charts in figures 9.5.2a and 9.5.2b show the increase in the performances of the system, as a function of the number of pairs that we use for the training. In Figure 9.5.2a we propagate the activations from the visual to the auditory SOM using only the activation of the BMU; in Figure 9.5.2b the propagation consider the activation of the entire map, after the phase of suppression.

The test is done on 10 classes, therefore, in each training iteration, we present 1000 visual representations as test cases and we evaluate the propagation of the activation in the auditory SOM. We can see that with 5 pairs the accuracy is 85%. After that value, the function stabilizes and the learning gets slower, reaching the value of 90% only after 10 additional in the training phase.

By comparing the charts, we can see that using only the BMUs to propagate the activations we reach better performances than using the entire map, even if it has been previously suppressed; this indicates that the more significant informations are contained in the BMUs and that all the other activations do not provide useful information. We notice that 100% accuracy is difficult to reach because of the anomalous visual representations and the subgroups present in the visual SOM.



(a) The propagation has been done considering only the activation of the BMU of the input presented to visual SOM.



(b) The propagation has been done considering all the activations of the visual SOM, at the end of the suppression phase.

Figure 9.5.2: Number of correct object-name associations, increasing the number of pairs used for the training.

Chapter 10

Conclusions

In this thesis we have studied the construction of a complex model that simulates the word learning process; as we have shown, the model is composed of three principal parts: the Deep Convolutional Neural Network (we chose the IncepionNet architecture), the visual, and the auditory Self Organizing Maps.

The advantages and the capacities of each of these neural networks originate a model able to face the complexity of high-resolution real images and to reach good performances in the creation of correct associations between the objects and their names.

In general when one simulates a real cognitive process, given the complexity that characterizes it, it is customary to introduce simplifications so that a limited model is able to successfully face the challenge.

In this thesis, we limit the amount of simplifications done and we simulate the word learning process in a way that is as much realistic as possible. For this reason on the one side we looked for a tool able to extract good representations from real images, and, on the other side, we use neural networks which are considered cognitively plausible.

The Deep Convolutional Neural Networks are an efficient tool for the extraction of features and also, as we proved with the experiments done in this thesis, to the extraction of good object representations. Indeed we proved that these representations exist and can be found in the last levels of the Deep Convolutional Networks, which is able to build them despite the complexity and the variety of the images presented as input.

The studies conducted on the representations extracted from the Deep Networks are a step towards a more complete comprehension of this type of networks and of the informations that they are able to build.

Despite what we have said, the Convolutional Networks can not be considered as cognitively plausible, even if they follow some general principles present in the human brain.

On the other side, the Self Organizing Maps alone can not face the complexity of real images, but, when they are used together with the Deep Convolutional Networks and they benefit of the capacities of this type of networks, the SOMs are able to reach significant results.

The Self Organizing Maps are useful during the simulation of the cognitive processes, because they are, architecturally and for the method with which they elaborate the inputs, similar to neural networks present in the human brain; indeed, the organization of the inputs in different areas on the basis of their similarities characterized some levels of the visual cortex.

In addition to its importance for the cognitive plausibility, the used Self Organizing Maps achieve good results in the categorization task, allowing us to distinguish the representations on the basis of their similarities and therefore bringing out the categories from the inputs. Furthermore, they allow us to build in an efficient way the connections that tie each category of objects to its name.

The optimizations done in the training algorithms both for the entire SOMs and for the synapses that connect the two SOMs, allow us to improve significantly the capacities of the SOMs and of the model.

The model, in its entirety, benefits of the capacities of its parts, reaching, as seen in the last chapter, good performances in the simulation of the word learning process.

In particular, when its parts are correctly trained, the model is able to understand which object is contained in the real image presented as input, to categorize the object and to associate its category to the correct name, in order to retrieve this association any time that an image that represents an object of that category or its name are presented to the model.

Furthermore the model presented is characterized by a strong modularity: the SOMs are the only networks in the model that are directly connected with synapses; the other networks (in our model only the Deep Convolutional Network) are connected with joints that are the visual and auditory representations.

The two SOMs ignore how their inputs are produced: this is well visible in our model where the visual and auditory representations are handled with the same algorithm, even if the visual representations are extracted from a deep neural network whereas the auditory inputs are produced artificially.

For this reason we may substitute the Deep Neural Network with another type of network, or we may insert a deep neural model in order to produce the auditory inputs from real auditory stream, without any change to the rest of the model.

Future developments may try to complete the construction of a realistic model by substituting the module that build the auditory representations with a recurrent neural network able to extract word representation from the auditory stream presented as input.

Future developments could be focused also on the production of the correct images or sounds starting from the activation of the corresponding Self Organizing Map, process that is not handled in this thesis, but that has to be considered if we want to simulate the entire process of word learning.

Appendix A

Experiments with mNeuron

In this appendix we present the experiments conducted using the tool mNeuron, in order to solve two different problems: the reconstruction of an input image from its representation and the comprehension of the features of an input image that allows InceptionNet to correctly classify the image. This two apparently different problems have a common solution well developed in the mNeuron tool.

First of all, as we have mentioned in chapter 9 rebuild an input, in particular a visual input, starting from the level of activation of a SOM is a difficult process.

In particular, we have to rebuild the representation associated to a specific activation of the SOM and then to rebuild from that the real input. The second part is the most difficult, especially when we deal with a network complex like a Deep Convolutional Neural Network. In this case we need to go through the entire network, from the top to the bottom, in order to reconstruct the input.

The alternative is to produce all the possible input images, calculate their representations and find that more similar to the produced representation. This last solution has a excessive temporal complexity, seen the dimensions of the input images, the fact that they are built on three level of color and the amount of values that each pixel can assume, therefore is discarded.

The second problem is related to the comprehension of the work done by InceptionNet. Networks with that level of complexity are hard to understand and, in particular, it is hard to understand what are the features that are captured in each level.

We find in mNeuron a partial solution for both the previous problems.

Between the functionalities offered by mNeuron, the tool presented by Wei et al. [27] and accessible at [2], there is that of find an image that optimizes the activation of a single neuron.

The tool starts from for an architecture (it is compatible with different deep networks) and it uses the level L and the neuron N in L whose activation we want control with the reconstructed image.

The method used is the following:

- we provide to the tool an initial image (even one randomly created);
- we provide a reference image;
- the tool calculates the activation of the level L starting from the reference image;
- the tool calculates the difference between the activation in the level L produced by the initial image and the reference activation;
- with the gradient descend method, the tool aims to reduce the difference between the two activations, in particular for the neuron N, changing the initial image.

At the end the image rebuilt by mNeuron contains all the characteristics that allow to the network to activate the level L, and in particular its neuron N, with a specific activation.

The tool is therefore useful to rebuild an image starting from its representation: if we set the level of the representation as L, with any neuron N specified, the resulting image contains the distribution of pixel that causes the build of that representation.

At the same time, if we use as L the classifier of the Deep Convolutional Neural Network, we can find the features of the images that lead to a specific classification: for example, if we provide a reference image that is correctly classified, the activations in the classifier are all near to zero, except for the neuron that identify the correct class. The algorithm, trying to build an image that produces that activation, builds an image that contains all the features useful to correctly classify the image.

Trying to understand the work done by InceptionNet, we can follow this last road in order to see what features help the net to correctly classify an input image.

In particular, we use the tool with the AlexNet architecture, which achieve good result in the classification of real images.

Figure A.1 shows the initial image that is corrected during the execution of the algorithm.

Figures A.2 shows the rebuilt images for three different classes of examples: *cats*, *dogs* and *tables*. The rebuilt images depends on the reference images provided, therefore they can not be considered as exemplar images for those categories.

We can see that the rebuilt images are difficult to understand: in image A.2b, we can identify three cats (in center and in the top corner) in which we can identify the ears; in image A.2d the head is identifiable in the top-left corner; in image A.2f a table is identifiable in the bottom-right corner.

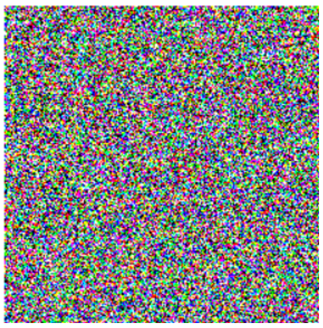


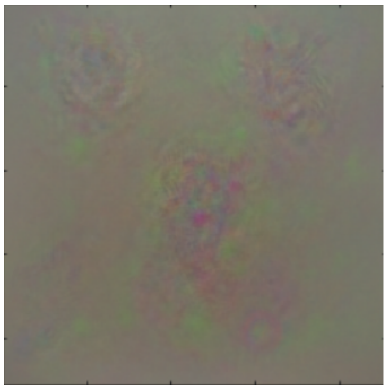
Figure A.1: Initial image randomly created.

This means that the features that are identified by the net in order to correctly classify the images may be located not in the center of the image, but the net recognize the presence of an object even if it is present multiple times of it is located at the corner.

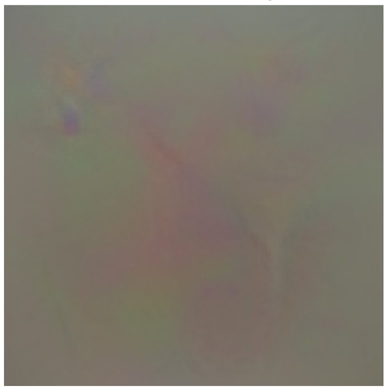
At the same time, the images are not so clear and do not give us other useful informations.

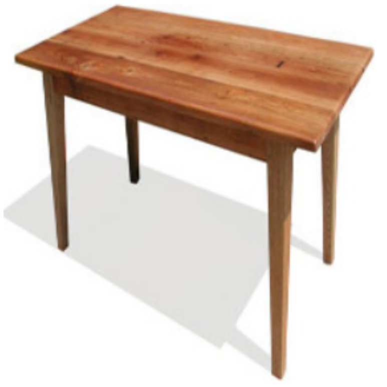
For this reason mNeuron can be considered as a useful tool in order to understand better what features are important for the network and its classifier, but is insufficient both for give complete informations and for rebuild a significant image starting from a representation.

(a) Reference image.


(b) Rebuilt image.

(c) Reference image.


(d) Rebuilt image.


(e) Reference image.

(f) Rebuilt image.

Figure A.2: Examples of the rebuilt images of the classes *cats*, *dogs* and *tables* with the relative references images.

Bibliography

[1] Tensorflow inception model. `https://github.com/tensorflow/models/tree/master/inception`.

[2] mneuron tool. `http://vision03.csail.mit.edu/cnn_art/`.

[3] E. Dromi. Early lexical development. *Cambridge University Press*, 1987.

[4] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. *CoRR*, abs/1312.2249, 2014.

[5] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2014.

[6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. 2012.

[8] Q. V. Le, M. Ranzato, R. Monga, M. Devin, G. S. Corrado, K. Chen, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2012.

[9] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area v2. 2007.

[10] H. Lee, R. B. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. 2009.

[11] D. Mareschal, K. Plunkett, and P. Harris. A computational and neuropsychological account of object-oriented behaviours in infancy. *Developmental Science*, 2: 306–317, 1999.

[12] E. Markman. Advances in developmental psychology. 1981.

[13] E. Markman. Constraints children place on word meanings. 1990.

[14] J. Mayor and K. Plunkett. A neurocomputational account of taxonomic responding and fast mapping in early word learning. *Psychological review*, 117 1:1–31, 2010.

[15] Y. Munakata, J. L. McClelland, M. H. Johnson, and R. S. Siegler. Rethinking infant knowledge: toward an adaptive process account of successes and failures in object permanence tasks. *Psychological review*, 104 4:686–713, 1997.

[16] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. 2010.

[17] K. Plunkett. Associative approches to lexical development.

[18] W. Quine. Word and object. *MIT Press*, 1960.

[19] E. T. Rolls. Computational and psychophysical mechanisms of visual coding. pages 184–220, 1997.

[20] E. T. Rolls and M. J. Tovee. Sparseness of the neuronal representation of stimuli in the primate temporal visual cortex. *Journal of Neurophysiology*, 73(2):713–726, 1995.

[21] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[22] E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10 1:89–96, 2000.

[23] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. 2013.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2015.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[26] G. Wallis and E. T. Rolls. Invariant face and object recognition in the visual system. *Progress in neurobiology*, 51 2:167–94, 1997.

[27] D. Wei, B. Zhou, A. Torralba, and W. T. Freeman. Understanding intra-class knowledge inside cnn. *CoRR*, abs/1507.02379, 2015.

[28] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. *CoRR*, abs/1412.6856, 2014.