UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA LM-18

UNIVERSITÀ DI PISA

Title: Image captioning with structure generation

Author : *Davide Italo Serramazza*

Supervisor : Prof. *Davide Bacciu*

Abstract

The aim of this thesis concern a comparison between image captioning task performed as image to sequence problem and performed with structure domain, more specifically using trees as targets or using trees in both input and output domains. To meet this goal we needed to represent both inputs and targets as trees, specifically image inputs as trees representing a hierarchical segmentation of the image, and sentence targets represented as parse trees. The model used to learn a mapping between inputs and outputs is *Conditional Variational Autoencoder for Tree-structured data* extended with the introduction of some components, such as an RNN and an embedding layer, needed by captions generation. This model was tested on two different datasets, Flickr8k and a new dataset, build ad hoc for the purpose of this thesis, starting from ms Coco. The results are encouraging indeed in our comparative experiments we found the best result with our proposed models.

Contents

Chapter 1

Introduction

Over the last few years, deep learning methods have gathering increasing attention thanks to their ability in providing accurate, effective and efficient data-driven solutions to a variety of problems. Many fields of research in Computer science were subjected to revolution due to the impact of deep learning methods: two of these fields are Machine Vision and Natural Language Processing (NLP). A popular task that involves both these two fields is image captioning, which is the topic of this thesis. Indeed it consists of generating a textual description for a given image: it requires Machine vision to process images, and NLP to generate the related description.

The most popular way in which a single image is represented, both when saved into a file and when used as input for Image caption task, is by a tensor of integer values of dimension $3 \times h \times w$, representing an image of dimension $h \times w$. The elements in position c, i, j encode the intensity values of the pixel at position i, j in the channel c. Indeed, the images are often encoded as RGB, an encoding in which each pixel in the image is encoded using 3 values that represent the intensity value, respectively, of red, green and blue color components: the combination of all the possible different intensities of these 3 basic colors can represent a large range of different colors.

Concerning targets for image captioning task, sentences, they are represented as a sequence of words: the i-th word follows the $(i-1)$-th until the end of the sequence is reached.

These representations are very natural, indeed when we write a sentence, we do it exactly like a sequence of words separated by a space, a comma, etc. However these two representations miss to describe the hierarchical aspects encoded in both data: for instance, we can most of the time split an image into different objects or areas and

among them, we could naturally represent a hierarchy going from the most relevant to the less relevant.

Also sentences have a structure, indeed in each natural language is not sufficient to build a sentence semantically well-formed but we need also to take into account the syntactic and grammar aspects of the sentence. Moreover, we could also identify a hierarchy of words into a sentence as done for instance by dependency trees.

A small fraction of works that use sentences as targets represents them as structured data: when this is done, it is often in the form of a parse tree or dependency tree. Moreover, in the image captioning task, to the extent of our knowledge, all works do not represent the image inputs as structured data, but using the image encoding as a tensor, as previously described.

Even if deep learning has shown superior results than other machine learning algorithms, most of the works in this field, do not consider the structure behind the data, using an unstructured representation for them. We think that the usage of structured data in deep learning algorithm deserves a deeper investigation because, potentially, the additional information that a structured representation of the data can bring in with respect to a flat one, can substantially improve the performance of adaptive models, helping them learning the true structure that is behind data distribution.

To make such step, two components are required: representing the data in a structured way, even if their "greedy" representations is not structured, and second to develop new deep learning algorithms that can process structured data.

For the first purpose, we use for both input and output data, a common data structure often used in computer science that is tree. Specifically, we develop a way to represent images as trees encoding their hierarchical segmentation, and we use parse trees to represent target sentences as trees.

For the second purpose, we use *Conditional Variational Autoencoders for Tree-Structured data* that is a Deep Generative Neural Network, specifically a Conditional Variational Autoencoder [53]. However even if it is able to process tree-structured data, there has been the necessity to make some modifications, specifically, the insertion of some new components is required, in order to generate captions more suitably.

Motivated from the reasons discussed before, in the following of this thesis we make a comparison between image captioning task performed as image to sequence, i.e. using the unstructured data representation described before (that are images as tensor and captions as sequence of words) and using structured data: we want to discover

how hierarchical information can improve the qualities of results.

Specifically, we use two different kinds of structured data representation: in both tested models we use as targets, sentences represented as parse trees while images are represented in one strategy as tensors (as described before) and processed by Inception v3 CNN [54] and in the other one as trees describing a hierarchical segmentation of the images.

The following of this thesis is structured in that way: in Section 2 we discuss the background of the different fields involved in this work, such as Image segmentation, Image processing by Neural Network and Tree-structured data processing by Neural Network.

In Section 3 we discuss the previous work used in this thesis, how thanks to that we reach our goal to represent both input and target as structured data and finally the modifications to *Conditional Variational Autoencoders for Tree-Structured data* done to adapt it to our goal.

In Section 4 we describe the principal experiments performed and the relative results achieved and finally in Section 5 we discuss the final conclusions of this work.

Chapter 2

Background

Image captioning is a popular task in machine vision. It aims to generate a caption for a given image. Ideally, we target at generating sequences that are well-formed from a syntactical point of view while describing in detail the contents of images: natural language processing is required for the former requisite, while image understanding is required for the latter.

Focusing on image processing, in order to ”understand” images, the algorithm which processes them, must extracts meaningful features from the images: the way in which this task is performed has changed over time.

In classic machine vision, features were extracted using handcrafted filters such as Scale-Invariant Feature Transform [38] or Histogram of Oriented Gradient [17]: after features have been extracted in this way, they were then fed to a classifier which was responsible for the recognition of objects in the images.

In the last years the tendency in machine vision and AI more in general, is towards deep neural networks which are automatically able to learn the suitable input re-representation i.e. features extraction. In particular *Convolutional Neural Network* (CNN) [33] is the standard way of processing images. In this latter approach, the process of features extraction is no more human-designed, but it is learnt from data, resulting in better flexibility in any possible scenario.

Focusing on the second requisite, that is natural language processing, with deep neural networks also the process of captions generation is learnt form data and no more human designed. In Figure 2.1 is represented a popular architecture of a neural network for image captioning: a deep convolutional part (one or more convolutional layers) which are needed for image features extraction followed by any kind of Recur-

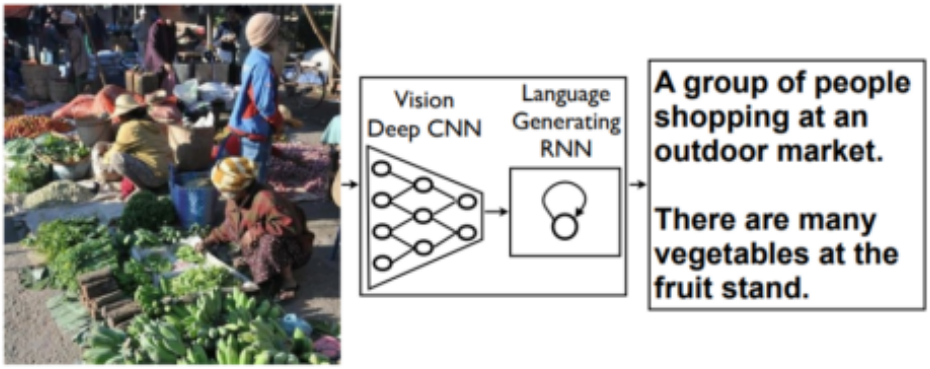


Figure 2.1: High level architecture of *Show and Tell: A Neural Image Caption Generator* [58] . This kind of architecture is very popular in image captioning: a deep CNN for image features extraction followed by a RNN for captions generation. Image taken from *Show and Tell: A Neural Image Caption Generator* [58] original publication

rent Neural Network (RNN), needed for caption generation.

Moreover another aspect that contributes to making image captioning a hard task is that a quantitative assessment of the results is not straightforward: some of the evaluation metrics commonly used are BLUE [47], ROUGE [34], METEOR [6] that are born for NLP tasks such as machine translation and language generation. Since these metrics are not designed for the specific task of image captioning, they all present the drawback of not directly considering how much the generated caption describes the given image. Recently, for this purpose new evaluation metrics are born such as CIDEr [57] and SPICE [1] which tries to take more into consideration also the relatedness of the generated sentence with the given image.

In the remaining of this chapter the principal tasks involved in this thesis will be introduced: in Section 2.1 will be introduced image segmentation task whose aims to identify segments in a given image, each of them has to be representative of a particular object or aspect of the image, in Section 2.2 will be discussed how the images are processed by Neural Networks and in Section 2.3 will be discussed general processing of trees using Neural Networks.

2.1 Image segmentation

Image segmentation is one of the most important and most difficult problems in machine vision. It is important because it is used both as stand-alone task in order to

segment an image in several areas and, at the same time, it is used as pre-processing for other computer vision tasks such as objects recognition or images classification.

Moreover, in the image segmentation field, there is no standardized approach but there are many algorithms which use very different approaches and among them there is no clear winner.

As a help to the reader, in Figure 2.2 a split among the different image segmentation algorithms is summarized.

One straightforward partition among the different algorithms could be made according to the purpose that they have: edge detection algorithms have as aim to assign to each pixel in the image a binary label that represents whether or not this pixel is a boundary pixel, i.e. a pixel that separates two different areas of the image.

The other class of algorithms is region segmentation that aims to divide the given image into several areas with empty intersections among each other. Even in this case, to each pixel in the image is assigned a label but this time it is not binary, because it represents which of the found regions, the current pixel belongs to. The more natural way to assign such labels is by giving an integer number: if in the image n different regions were found, labels $\in (0, n]$ is assigned to each pixel.

In edge detection, gPb [2] is one of the most relevant work. It becomes a standard in boundary probability map generation which is a greyscaled version of the given image, in which the color intensity of each pixel is proportional to the computed probability for that pixel to be a boundary.

More in detail from local cues, computed by oriented gradients in the original image, it defines a multiscale oriented signal, mPb, that aims of combine these local cues thanks to a weighted sum of them. Then it defines an affinity matrix from which an eigenproblem is defined and in order to be feasible computed, it will be solved just for a fixed number k of eigenvectors, which will be then combined in a weighted sum, sPb, using as coefficients $\frac{1}{\sqrt{\lambda_k}}$ which are the corresponding eigenvalues. Finally, since mPb and sPb contain different information, they are summarized, another time with a weighted sum, in order to have the final gPb. Coefficients for mPb and for gPb are learnt by gradient ascent on the F-measure.

Previous works in this field are Normalized cut [49] that also takes the affinity pixel matrix and solves a related eigenvalue problem or Mean shift that considers region segmentation as a density mode searching problem [14].

Exploring region segmentation methods, we could further split the algorithms

based on their approach. A class is the one that represents images using histograms. The idea behind this approach is to take some features of an image, represent them with histograms, and then find a threshold value that is able to separate two different areas in the image or an area from the background. Algorithms that use this approach are [48] in which the threshold is based on pixel value mean or pixel intensity values median.

In [18] a histogram technique is used along with Fuzzy C Means technique in which, thanks to its fuzziness, it can be detected the degree of participation of a pixel to each cluster. Moreover, it also improves robustness against noise thanks to the spatial probabilities of neighborhood pixels. Finally, fuzzy clustering is employed in the proposed approach to achieve proper segmentation.

Indeed, another different class of algorithms is the one based on clustering. Clustering is the task of assigning each of the given elements to a specific element set also called cluster. Ideally in a cluster, an element has to be more related to each of the other elements in the same cluster than to any other element in any other cluster. The most popular and one of the simplest cluster algorithms, is k-means which uses as metric the euclidean distance among data points in order to assign each of them to one of the k cluster (K is an hyper-parameter). When it is applied to image segmentation, the distance is computed according to color intensities, texture, locations or a combination of these and other factors.

In [19] it was proposed Subtractive clustering method which is an efficient technique to find cluster centroid, that is the point in which "the center of the cluster" is located, based on the density of neighborhood data points. It also estimates the number and initial locations of cluster centers.

In [8] it was proposed a system which creates the temporary and individual cluster which helps to find an optimal threshold value. The optimal cluster value is computed by applying K-Means clustering algorithm.

Also fuzzy clustering is used in this kind of algorithms. A fuzzy clustering is a cluster in which each data point can belong to one or more than one clusters. Membership degrees are assigned to each data point to express their degree of participation to each cluster.

One example of this class of algorithms is [11] in which they are used very simple features such as mean and standard deviation of pixels color to build a fuzzy cluster that is further processed in order to retrieve the required segmentation.

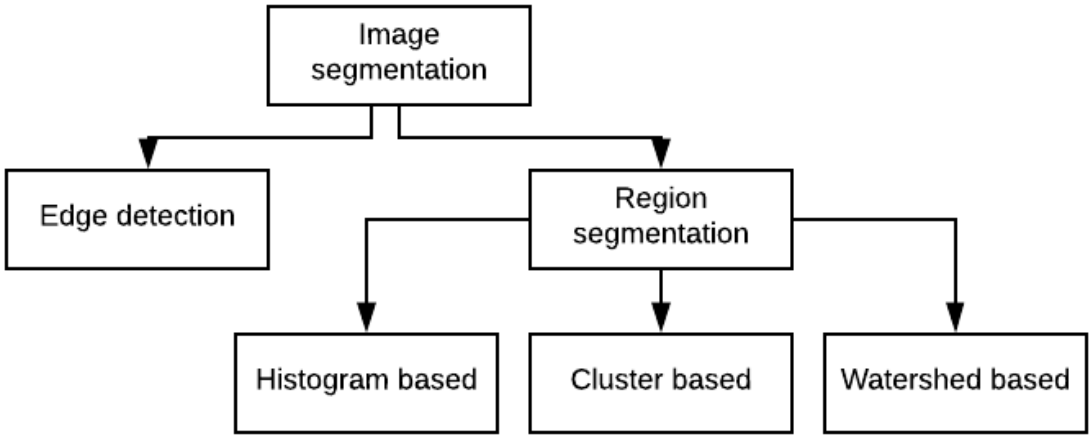


Figure 2.2: Conceptual division among the different image segmentation algorithms

Another very popular class of algorithms in image segmentation is Watershed algorithms, presented in [10]. In this algorithm, the image has to be in gray scale, so standard RGB images are first transformed. To each pixel of the image with location (x, y) it is also assigned a third dimension z that is its intensity value. A closed line built by pixels with the same intensity value is considered as a "threshold" from a segment to another one. This class of algorithms is called Watershed because it is possible to make an analogy with a catchment basin that has height x , width y and depth z in which someone pours the basin with water, starting from the lowest basin to the highest peak. The algorithm is called "Watershed" because what we previously called threshold with the analogy just made, is now more natural seen as a watershed.

From this simple idea, a huge variety of different implementations were born, each of them presenting some slight changes to the original algorithm, but nevertheless the key idea has always remained the same. Some variations concern the way in which the process of water pouring could be thought, such as for the watershed by flooding and its improved version in [7], or guided by the water droop principle as in [16]. Other changes were made, for instance in the topological watershed in which also the topology of the image is taken into consideration as in [15].

The work discussed in this Thesis will rely on gPb and Watershed to achieve image segmentation.

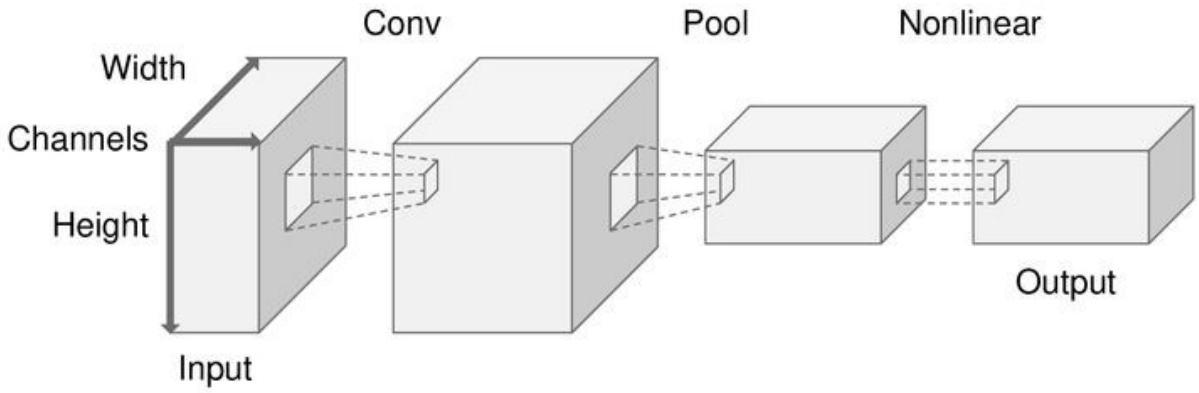


Figure 2.3: Common architecture of a convolutional layer. On the left is represented the original image characterized by width, height and number of channels (usually 3). Then going from left to right, the different transformations usually applied to images are shown: first several convolutional kernels are applied, usually augmenting the depth of the image, followed by a pooling operator first and finally a non-linear activation. All these transformations are discussed in the following.

2.2 Image processing by neural networks

2.2.1 CNN

In Section 2.1 we discovered how in image segmentation field there is no standardized approach. On the other hand, if we consider image processing in a more general way, Convolutional Neural Network (CNN) is a de facto standard even if CNNs are also very popular in other AI tasks such as Natural Language Processing (NLP) or time series processing.

CNN was first introduced by LeCuN [33] but nowadays the common meaning of convolutional neural network is slightly different and more complex: usually there is more than one convolutional layer in order to extract features from images, followed by any other kind of layer suitable for the specific task the network is designed. Convolutional layer term is usually referred to an architecture like the one shown in Figure 2.3: more than one convolutional kernel, followed by some kind of pooling and finally a non-linear activation. We briefly analyze these three components in the following sections.

Convolutional kernel

A convolutional kernel works by dividing the image into several smaller blocks. For a single block, the respective elements of the kernel are multiplied by the corresponding elements of the image block and then they are all summed together. More formally the convolution between a kernel K of dimension $m \times n$ and a usually bigger dimension image I is defined as

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{2.1}$$

From the formula above we can notice how this filter is applied only to a subregion of the whole image of dimension $m \times n$ and also the overall image dimension reduction: starting from a dimension of $m \times n$ every convolution ends up in a scalar.

To process the whole image, the kernel "slides" in it, thus kernel is convolved with different image blocks. As illustrated in figure 2.4 the kernel moves from one subsample of the image to another one with a sort of sliding window that at each step moves rightward or downward, according to its current position. The amount of movement, i.e. the number of pixels the kernel skips at each step is regulated by a hyper-parameter called stride.

Another key aspect to notice is that the same kernel is applied to the whole image or in other words the same $m \times n$ scalars are multiplied with the different pixel values in the image. This is a key aspect, when backpropagation is applied to the network, in order to learn suitable weights for the elements in the kernel and thus end up in a convolution which is able to extract meaningful features from the images.

The last aspect to stress is that more then one kernel are applied to each convolutional layer. All image channels are convolved together, so each of these kernels is applied to the full image. In this way, potentially, thanks to the use of many convolutional kernels, it is possible to extract from the same set of pixels, different information with different degrees of abstraction.

Pooling

Pooling has two different aims, namely: making the image representation smaller and in the same time more robust. There are different types of pooling such as max pooling, L2 pooling or average pooling. In Figure 2.5 is shown a max pooling example of size 2×2: the 4×4 image is divided into blocks of size 2×2 and for each of these

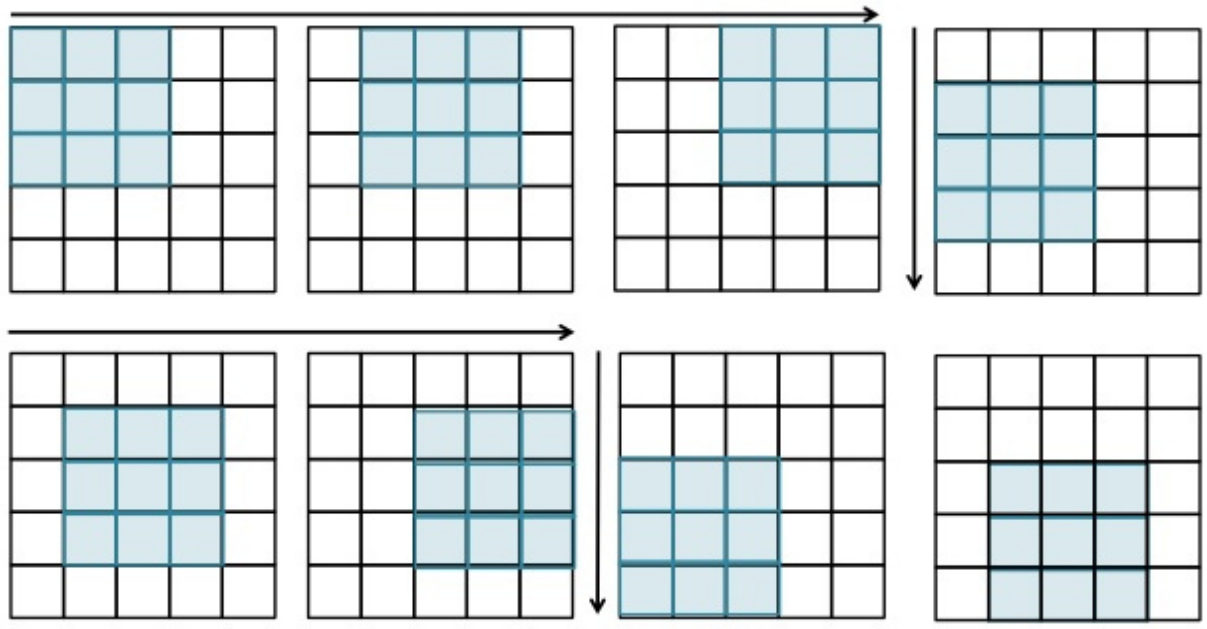


Figure 2.4: Visual representation of kernel stride with a stride value of 1. At each step the kernel skip 1 pixel, moving to its right. When there are no more elements to its right, the kernel move to the beginning of next row.

blocks only the maximum element is taken as result of the pooling operation. This is useful in order to obtain a more robust representation: once features are extracted the exact location in which they are, has not a huge relevance and moreover, pooling makes the features extracted invariant to scaling, rotation or other geometric transformations. Finally, the reduction in representation dimensions helps in regulating the complexity of the network.

Also in pooling there is the stride hyper-parameter that regulates the number of pixels skipped at each step.

Non linear activation

The last ingredient is non-linear activation. Recently, ReLU and its variants are preferred over other activation functions such as sigmoid or tanh, because it helps to prevent the gradient vanishing problem.

2.2.2 Autoencoders

The architecture of a simple autoencoder (AE) is shown in Figure 2.6. It is composed of an input and an output layer having the same shape. There is a hidden lower-

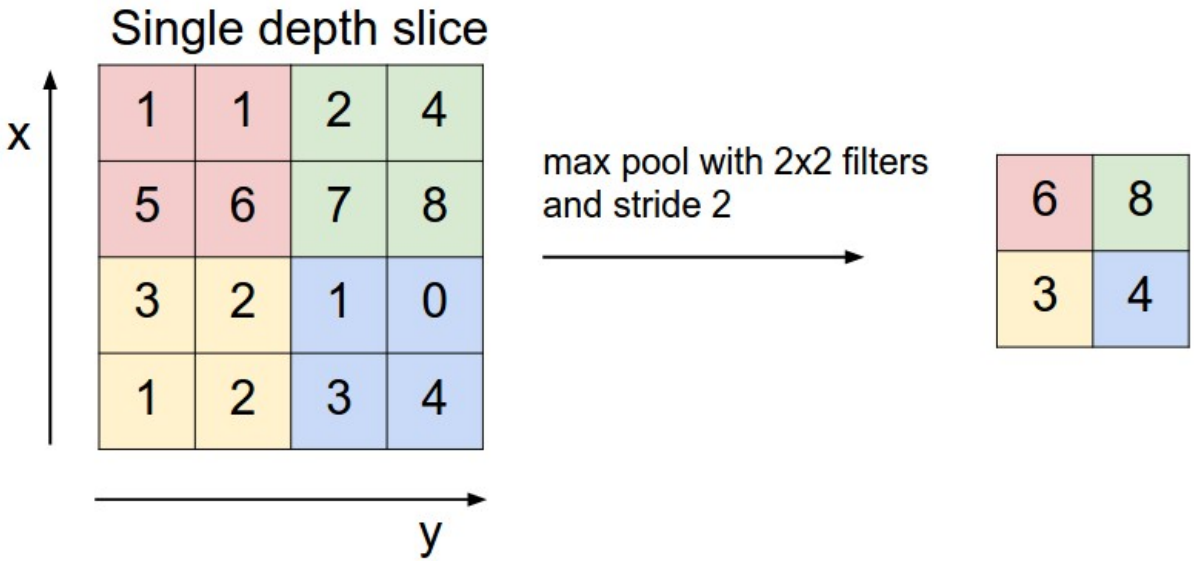


Figure 2.5: Max pooling of dimension 2×2 with stride of 2 applied to an image of dimension 4×4. For each of the 4 blocks of dimension 2×2 only the maximum elements are taken as results.

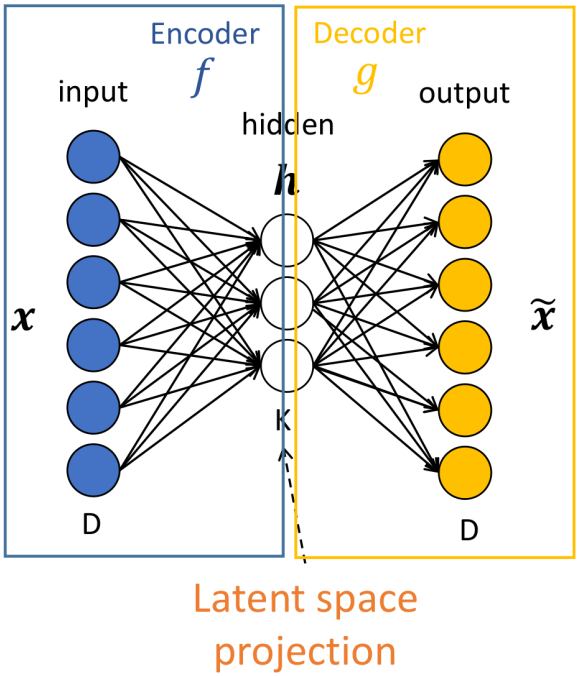


Figure 2.6: Simple autoencoder architecture. The dimension of input and output is D, while hidden dimension is K with $K < D$. The hidden representation h is a lower dimension latent space projection of the input.

dimensional layer between them that could be considered as a ”bottleneck”. Indeed an autoencoder works by encoding the input in a hidden representation and starting from that, it tries to reconstruct the original input in the decoder part.

More formally, given an input x, the reconstruction made by decoder will be referred as $\tilde{x}$. Given a loss L, the cost function the network is designed to minimize is $L(x, \tilde{x})$. Denoting encoder as f and decoder as g the loss is rewritable as

$$L(x, \tilde{x}) = L(x, g(f(x))) \tag{2.2}$$

From equation 2.2 it it possible to notice that the network aims to reconstruct the input, passing through the encoded representation.

This process also presents a probabilistic interpretation: denoting as in Figure 2.6 the hidden representation as h, we could interpret the encoder as modeling the conditional probability $P_e(h|x)$ of the hidden representation given the original input. Similarly we could interpret decoder as modeling the conditional probability $P_d(\tilde{x}|h)$ of the reconstructed input given the hidden representation.

Previously we talk about encoder/decoder part, instead of encoder/decoder layers, because what is shown in Figure 2.6 is the simplest possible autoencoder with only a single hidden layer, but it is also possible to use several of them to have a more powerful network. Even in that case there is a latent projection of the input that is what we get from the smaller layer of the network. Each layer ”before” the latent projection belongs to the encoder part, while each layer ”after” the latent projection belongs to the decoder.

The autoencoder final aim is to learn a latent space projection in which the original input could be represented using the encoder f and from which the original input could be reconstructed using the decoder g: one possible interpretation of this process, is input re-representation. When applied to images instead of using the simple autoencoder just described before with the unrolling of a 2D/3D image into a single dimension, a Convolutional autoencoder (CAE) is preferred.

An example of CAE is shown in Figure 2.7: starting from the original image, the encoder part aims to reduce the dimension of image representation until a latent space projection is reached. After that, the decoder part tries to reconstruct the original image.

The differences between AE and CAE are in the layers composing the autoen-

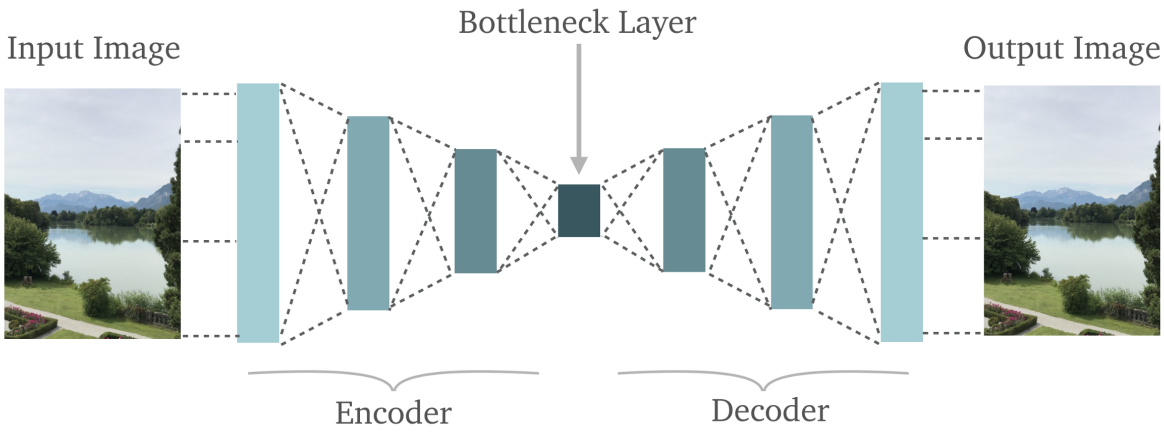


Figure 2.7: Popular convolutional autoencoder architecture. The input layer shapes and output layer shapes are symmetric, in the middle there are a different convolutional layers in order to represent the image in a lower-dimensional latent space, followed by de-convolutional layers to reconstruct the original image. The number of layers in the encoder part is usually the same as the ones in decoder parts.

coder: in the first one there are only dense layers, in CAE instead there are convolutional layers in the encoder and de-convolutional layers in the decoder.

As previously said in Section 2.2.1, a single convolutional kernel is not enough to extract a huge set of features, for this reason, each convolutional layer is composed of several convolutional kernels.

The key difference between a CNN and a CAE is that while the former is trained in a supervised way to perform a huge variety of tasks, such as image classification or image captioning, the latter is trained in an unsupervised way and has as only aim to extract features from the images. Layers trained in the last way could be further used in other networks where feature extraction from images is required.

2.2.3 Image captioning with Neural Network

Early attempts of image captioning were focused on retrieval based and template based models. In retrieval based, the generation of captions was done by retrieving one or more sentences from a predefined set. An example of this kind of work is [22] in which the image to describe is mapped in a meaningful space by solving a Markov Random Field. Next it uses a metric that determines the closest sentence in the pre-defined set: this sentence will become the generated caption for the given image. This approach obviously presents some drawbacks, for instance it retrieves a well-

formed sentence form a set of captions pre-built, but the quality of this sentence as a description for the queried image is very likely to be poor.

Another early approach was template based. In this approach, first of all we need to extract a set of visual features from the image. Associated with these features there are same sentences: starting from that, a caption is generated using template or specific grammar rules defined ad-hoc for this task. An example is [45]: it uses Conditional Random Field to determine image contents, in particular graph nodes correspond to objects or spatial relations. Once the contents of the image have been detected, in order to choose which ones to describe, Conditional Random Inference is used, in particular output of the inference is used to generate a template based description.

More recent works use deep learning. In the two fields needed for image captioning task which, as said previously, are NLP and machine vision, deep Neural Network outperform older approaches.

Some works which rely on Neural Network are [41] that use a RNN to predict the next word in the caption, given the previous word and the image representation, while for the features extraction it is used the pre-trained convolutional layers of AlexNet [31].

In [40] it uses two different embedding layers in order to capture both semantic and syntactic aspects of words. After that layers there is a Elman network [21] as RNN that receives as input the current word embedding and the recurrent activation. Finally, there is a multimodal layer that takes as input current word embedding, recurrent layer and image representation, that come from AlexNet [31] or in some other variants from VGGNet [50], which is aimed to produce word predictions.

Show and tell: A neural image caption generator [59] is close to [41], but it uses as RNN a more recent and powerful model that is a LSTM [26]. Moreover it feeds its image representation, which comes from pre-trained [28], just as first timestamp to LSTM.

Finally a more recent work is *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention* [61] in which CNN used in the encoder is VGGNet [50]. Image representation is fed at each timestamp to RNN, that is a LSTM, because it uses an attention mechanism, specifically Bahdanau Attention [5] originally developed for machine translation, in order to select the relevant information from the image representation, to generate a suitable word at the current timestamp.

In this thesis, two different "words modules" are used and they are inspired by

the last two works described in this Section, that are *Show and tell: A neural image caption generator* and *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*.

2.3 Tree structured data processing by neural networks

In order to perform tasks such as classification, regression, generation of new data, the extraction of meaningful representations of data is needed: this process is called *Representation Learning*. Many real-world data as for instance images and sentences have as usual representations that are not optimal for machine learning algorithms because they naturally encode hierarchy, that these representations are not able to fully express.

Considering for instance images, their natural representation are matrices in which each element encodes the pixel intensity value. An example of representation learning is convolution in CNN : as previously said in Section 2.2.1, in a typical CNN there are many convolutional layers used to extract features in a different level of abstraction from the images. In this new data representation, the data contains a form of hierarchical information going from more concrete ones contained in the first layers to more abstract information contained in the last layers. That kind of representation is more suitable to process images than the "classical" dense way, in which the whole 2D/3D image is reshaped as 1D vector and then processed.

Anyway, this kind of representation is not optimal: the data represented in this way, have poor structured representations, indeed it is just possible to interpret the result of convolutions, which are tensors, as hierarchical thanks to the presence of different layers in the network.

If we want to fully exploit the hierarchical structure of data, we need algorithms that can handle a more structured representation. A commonly used data structure in computer science that can express in a natural way a hierarchy is tree.

Motivated from the previous consideration, in the rest of the paragraph, we first introduce some notions and formalization and then we summarize some works in which tree structured data are processed with Neural Networks.

2.3.1 Tree structured data and transduction formalization

A tree is generically identified by a triplet ($\mathcal{U}_n$, $\mathcal{E}_n$, $\mathcal{X}_n$) composed by a set of nodes $\mathcal{U}_n = \{1, ..., U_n\}$, a set of edges $\mathcal{E}_n \subseteq (\mathcal{U}_n \times \mathcal{U}_n)$ and a set of labels $\mathcal{X}_n$=$\{X_1, ..., X_n\}$.

The term $u \in \mathcal{U}_n$ is used to denote a node in the tree: we denote its parent as $pa(u)$, its children as $ch_l(u)$ where the sub-script l denotes the $l-th$ child of u (we make the assumption that node children are ordered). A pair $(u, v) \in \mathcal{E}_n$ is used to denote an edge between the two nodes u and v. Finally to each node of the tree has been associated a label $x_n \in \mathcal{X}_n$.

A tree transduction defines a mapping from elements of input domain to elements of outputs domain, where both domains are composed by trees. A function $\mathcal{F} : \mathcal{I} \rightarrow \mathcal{O}$ where we denote as $\mathcal{I}$ the input domain and with $\mathcal{O}$, the output domain, both composed of trees, is a function defining a tree transduction.

2.3.2 Tree structured encoders

In this Section we report several models that encode a tree in a fixed-length representation, i.e. embedding. We include in this section also Recursive neural networks [23]. Tree as data representation for Neural Network is principally used in some NLP works because sentences intrinsically have a hierarchical representation that could be not captured by a sequence. For this reason most of the following works regard NLP tasks.

In [51] a recursive neural network is applied to parse trees. Denoting with the pairs (A, a) and (B, b) two matrix-vector pairs for two adjacent word x and y, the encoding of two adjacent words is done by getting another matrix-vector pair (p, P) as result of

$$p = f(Ba, Ab) = g(W \begin{bmatrix} Ba \\ Ab \end{bmatrix}) \tag{2.3}$$

$$P = f_M(A, B) = W_M \begin{bmatrix} A \\ B \end{bmatrix}) \tag{2.4}$$

Thanks to the dimensions of P and p, that are the same of the matrix-vector representing one word, this equation can be applied in a bottom-up fashion, recursively, on each word in the parse tree until the tree root is reached: this process has, as a result,

a matrix-vector pair representing the whole sentence. This model outperforms for the first time a Conditional random field, in the task of movie reviews classification.

In [52] the key idea behind [51] was extended. In it there is just a single, more powerful composition function, aggregating meaning from smaller constituents. This reflects in the equations of the model: instead of having two equations that was 2.3 and 2.4, in this work there is only one vector representing each word, coming form the result of equation:

$$p = f(\begin{bmatrix} a \\ b \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ b \end{bmatrix} + W \begin{bmatrix} a \\ b \end{bmatrix}) \tag{2.5}$$

in which along with the parameters matrix W, and adjacent word representations a and b the principal component is the tensor $V^{1:d]} \in \mathbf{R}^{2d \times 2d \times d}$, where d is the word vectors dimension, that defines multiple bi-linear forms. The main advantage of this equation is that tensors can directly relate input vectors: indeed, we can interpret each slice of tensor V as capturing a specific type of composition. This model sets the state the art at the time for sentence binary sentiment classification.

In [55] the encoding was no longer composed by a Recursive neural network but by a modified version of LSTM, ad hoc designed to encode trees rather than sequences. There were proposed two different models: *Child-Sum Tree-LSTM* and *N-ary Tree-LSTMs*. What is in common between these two, is the idea that the LSTM, in order to perform a prediction on the current node, exploits several inputs and hidden states that come from current node children. In this way the hierarchy encoded in the trees is taken into account also in the final fixed representation. This model outperforms other models in the tasks of semantic relatedness of two sentences and sentiment classification.

All the models seen in this section can just encode a tree in a fixed representation in order to perform classification/regression but they can not be used for a not fixed dimension output such as the ones in generative tasks.

2.3.3 Tree structured decoders

Tree decoders generate trees starting from fixed representations. With this network, along with a tree encoder, we could achieve tree transduction task as previously described in Section 2.3.1.

In [62] it was used 4 different classifiers in order to predict if a node has to be added to the current node right, left, next-right or next-left. If one of these 4 classifier returns true, the corresponding LSTM generates this predicted node. This model was used on dependency parsing re-ranking.

In [20] it was used the classical sequence encoder of a sentence, i.e. by recurrent neural network, but the decoder aims to generate a tree. In order to do so, some special tokens were used to denote non-terminal nodes such as the begin and the end of sequences. The decoder operates top-down starting from the root: it generates ”word nodes” using non-special nodes, and it uses special nodes to go into a lower level in the tree, i.e. to represent tree topology.

In [4] a structure-to-substructure transduction is realized to perform text summary: starting from a parse tree representing input sentence, it ends in another parse tree representing the output sentence which is a summary of the input. In this work, the decoder is designed as top-down because in this way, processing first POS tag and conditioning the generation of words node on them, it is more likely that the model can capture the differences among the possible usages of a specific word in different contexts, for instance when the role it plays in the sentence changes (e.g. subject, direct object).

Chapter 3

Tree to Tree transduction for image captioning

In this chapter, will be presented the model developed in this thesis. In Section 3.1 will be discussed previous works used in this thesis that are GLIA [37], AlexNet [31] and Inception v3 [54] CNN and finally *Conditional Variational Autoencoders for Tree-Structured data*. In Section 3.2 will be discussed the different models used in this thesis which are Conditional Variational Autoencoders for Tree-Structured data and its modified versions and in Section 3.3 will be discussed how what is just described are used in this work.

3.1 Architecture

3.1.1 GLIA: graph learning for image analysis

Graph Learning Library for Image Analysis (GLIA) [37] is an image segmentation algorithm. Using the terminology introduced in Section 2.1 it is a region segmentation algorithm: its final aim is to find a hierarchical subdivision of the whole image into several regions with empty intersection among each other. It relies on early works such as gPb [2] and Watershed [10].

First of all, to reduce the segmentation complexity and making this task feasible, GLIA does not perform the segmentation on the original image pixels but on the so-called image super-pixels. There are many algorithms that staring from a given image create super-pixels: they make up an over-segmentation of the image, with very small regions, so small that they could be considered as larger pixels or super-pixels.

Many region segmentation algorithms prefer to search optimal segmentation from super-pixels instead of pixels because in a super-pixel there are only connected pixels with similar properties among each other, thus they are very likely to end up in the same final segment and moreover starting from super-pixels rather than pixels, the complexity search is reduced.

GLIA uses as superpixels the regions found by Watershed over a blurred version of the gPb algorithm applied on the original image to segment.

To describe how starting from that superpixels, the algorithm ends up in an image segmentation, some concepts such as merge tree need to be introduced first.

Merge Tree

A merge tree is a binary tree in which the nodes are image segments and arcs from parents to children are inclusion relationships. Leafs of this tree are the initial superpixels while other nodes represent regions that are the union of the ones described by their children.

Formally let us denote the set of nodes as V, the set of edges as E, a single node in the tree as $v_i^d \in V$ where d denotes the depth at which this node is, two edges between a node v_i^d and its children v_j^{d+1}, v_k^{d+1} as $(e_{ij}, e_{ik}) \in E$.

At the beginning of the process, a merge tree is composed only by leafs, i.e. the super-pixels. Then at each step two nodes are merged until this process end up in a single node describing the whole image. In order to decide which nodes to merge at each step, a saliency function $f : V^2 \to \mathbb{R}$ is used. For this purpose $f(v_i, v_j) = -\infty$ if v_i and v_j are not connected in the image, while for connected nodes f is defined as

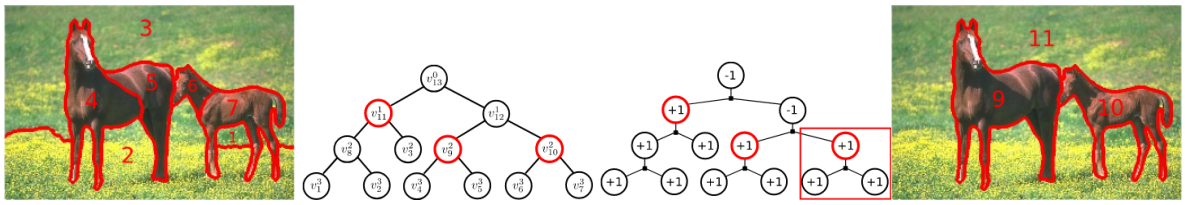
$$f(v_i, v_j) = 1 - median(Pb(k)|k \in B(v_i, v_j)) \tag{3.1}$$

where $B(v_i, v_j)$ denotes the set of boundary pixels between v_i and v_j and $PB(k)$ denotes intensity value of the k-th neighborhood in the boundary probability map.

Building of the tree is iteratively defined by merging the nodes v_i and v_j such that $arg\ max_{v_i^d, v_j^d \in S}\ f(v_i, v_j)$ until the last created node represents a region corresponding to the whole image. The result of this process is a binary tree that describes the image.

In Figure 3.1a is shown an image with the initially found superpixels, and in Figure 3.1b the related merge tree computed by the algorithm.

Once the merge tree is computed, the way in which GLIA achieves the optimal segmentation is by selecting from it the suitable sub-set of nodes.



(a) (b) (c) (d)

Figure 3.1: Image processing by GLIA example: in (a) are shown the initial found super-pixels for the given image, in (b) the merge tree obtained starting form that super-pixels, in (c) the related factor graph obtained from the merge tree (in the red box is shown a clique) and finally in (d) is shown the image segmentation computed by GLIA. Images taken from original GLIA original publication [37]

Constrained Conditional model

Once the merge tree is computed, the algorithm has to decide the nodes to select as final segments. In order to perform this task, a factor graph is built starting from the merge tree previously computed: the node set is the same of the merge tree and each merge with children v_j^{d+1}, v_k^{d+1} and parent v_i^d is considered as a clique p_i.

Each tree node in the factor graph is labeled with $y_i = \pm 1$ indicating whether its children merge or not, i.e. whether the regions described by these two nodes will end up in the same final segment.

To assign each of these labels y_i, a boundary classifier (described more in the detail in the following section) need to be trained to predict the probabilities of each merges $P(y_i)$.

Once these probabilities are computed each clique is scored with two kinds of energies that are merge energy Em and split energy Es, defined as.

$$E_m(y_i) = -log(P(y_i = 1)) \tag{3.2}$$

$$E_s(y_i) = -log(P(y_i = -1)) = -log(1 - P(y_i = 1)) \tag{3.3}$$

With these factor graph just described, the labeling problem is defined as

$$\begin{aligned} min_y \sum_{y_i \in Y} E(y_i) \ \text{ constrained to} \\ y_i = +1 \ \forall i \ \text{ s.t. } v_i^d \text{ is a leaf} \\ y_i \leq y_j, \forall i, j \ \text{ s.t. } v_i^d \text{ is parent of } v_j^{d+1} \end{aligned} \tag{3.4}$$

that is a min problem with constraints that each leaf must be label $+1$ to not select them as a final segment and that for each node v_i labeled as $+1$ all its descendants must also be labeled $+1$.

In Figure 3.1c is shown the factor graph obtained starting form the merge tree in Figure 3.1b.

Until now we have defined the merge tree, how the algorithm selects a subset of nodes that make up the desired image segmentation by solving equation 3.4. What remains to be described is how the boundary classifier, computing the merge probabilities $P(y_i)$, is designed and trained and finally the kind of inference used to solve equation 3.4.

Boundary classifier

To train the classifier to predict the probability of two segments to be merged, ground truth segmentation is needed: thanks to that it is possible to define a supervised task. In GLIA to extract labels, both split and merge energies, defined in the previous Section, were compared against ground truth segmentation using as metric Variation of Information [43] while as input to the classifier were extracted boundary and region features from the original images.

As just defined this is a generic supervised task that is not limited to any specific model. In GLIA and in the work of this thesis, we use as Boundary Classifier a Random Forest classifier [25].

Inference

The last aspect to discuss of GLIA is how inference is performed.

An exhaustive search to solve equation 3.4 is unfeasible. Nevertheless using the merge tree and the constraints to which the equation is subject, GLIA can perform the inference in linear time. It consists of a bottom-up/top-down visit of the tree: in the

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Figure 3.2: AlexNet architecture: the first 5 layers are convolutional ones of different sizes and stride some of them followed by max pooling. After the convolution layers there are 3 dense layers, and finally a softmax to classify the images among 1000 classes

bottom-up process for each node, merge and split energies are computed. For leaf nodes the energies are assigned by default as $E_i^m = 0$ and $E_i^s = \infty$, in order to force merging. For a generic other node v_i^d with children v_j^{d+1} and v_k^{d+1}, having both energies recursively computed, the computation is defined recursively as

$$E_i^m = E_j^m + E_k^m + E_i(y_i = 1) \tag{3.5}$$

$$E_i^s = min(E_j^m, E_j^s) + min(E_k^m, E_k^s) + E_i(y_i = -1) \tag{3.6}$$

In the top-down step, each node has its associated energies computed and labels are assigned. Starting from the root, a depth-first search is performed: each node that has merge energy lower then the split one is labeled $+1$. Moreover, all its descendants are also labeled $+1$ and the current sub-tree visit is left. Otherwise, if split energy is lower then merge one, the current node is labeled -1 and its children are put in tree visit frontier, thus they will later label accordingly to the rules just defined.

Eventually, all the nodes labeled $+1$ and whose parents are labeled -1 are selected as final segments.

In Figure 3.1d is shown the segmentation found accordingly to the factor graph in Figure 3.1c.

3.1.2 Alexnet

Alexnet [31] was designed to compete in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012) of 2012. It achieved a top-5 error of 15.3% more than 10 percent better than the current state of the art at that time. Today it is considered as having a historical impact on machine vision and deep Learning thanks to the usage in this work of some newel concept as ReLu activation function, training on GPU and Dropout. Up to now, AlexNet paper has been cited more than 44000 times.

It was trained on ImageNet, a dataset of over 15 million labeled images belonging to roughly 22,000 categories. To reduce overfitting, data augmentation was used: train set was augmented thanks to several transformations applied to each image. Transformations were in part geometric ones, such as rotation or horizontal reflections, and in part not geometric as variation in RGB color intensities, performed by adding a multiple of the principal component of the image, found by PCA analysis.

Moreover also Dropout technique [24] was used: it consists of setting the output of some neurons to 0, for each time that new inputs are fed to the network. Whether or not to drop each neuron is regulated by a given probability (an hyper-parameter). If the output of a neuron is set to 0 for a specific input, it does not contribute both to the whole network output and to backpropagation. With Dropout each neuron is forced to learn robust features because each of them can not rely on the other one outputs. Essentially this technique is prevention against the neurons co-adaptation: in this sense Dropout reduces over-fitting.

Concerning the architecture of the network, it is composed of 5 convolutional layers in order to extract meaningful features from the images, followed by 3 fully connected layers for classification (ILSVRC is a classification task). In Figure 3.2 is shown a table with details about the network architecture. First convolutional kernel is of size 11×11, the second one is 5×5, the other ones are 3×3. In the first and second kernels the feature map size grows, in third and fourth remains the same, while in the latter one it decreases. Finally following the first, second and last layer there are also 3 different max-pooling layers.

After these layers, the resulting tensor is reshaped into a vector and fed to the last 3 fully connected layers.

Finally, the output layer is a softmax that classify the images among 1000 class labels.

3.1.3 Inception v3

Inception v3 [54] is the 3-rd version of the inception CNN, originally designed by Google. All the versions were called Inception because they all rel on *Inception module*. Thanks to them they can achieve very good results with a small complexity in terms of network parameters. Inception v3 achieved less than 4% on the top-k error in 2015 ImageNet Large Scale Visual Recognition Competition, so more than 10% better then alexNet while it has less than a half of AlexNet parameters number.

In original Inception v1, the Inception modules are characterized by different convolutions, in terms of different kernel sizes, concatenated together as the output of the current module and thus as input to the following layer. With this strategy, the network can extract different features that have different levels of abstraction in each Inception module, while kernels composing these modules, have very little dimensions such as 3×3 or 5×5.

In inception v3 these kernel sizes are further reduced with different strategies such as replace 5×5 kernel with two 3×3 or replacing a $N \times N$ kernel with one $N \times 1$ and one $1 \times N$.

Also, the grid search reduction is done in a very original way: instead of using just max-pooling as conventional CNN, it concatenates a convolution and a pooling, with the same size.

There is also an auxiliary classifier before the real output of the network: it is used as a form of regularization.

After each convolution, there are Batch Normalization and ReLU activation functions. Also, the image input needs to be normalized in the range $[0, 1]$ before to be fed to the network.

The whole architecture of inception v3 is shown in Figure 3.3

3.1.4 Autoencoders for Tree-Structured data

Variational Autoencoders

In Section 2.2.2 we have introduced autoencoders, their probabilistic interpretations and the usual task they are designed to, that is extract features from input. Variational autoencoders (VAE) are different from classical autoencoders first of all in the purpose they have, that is Variational Inference. Instead of mapping input into a fixed-

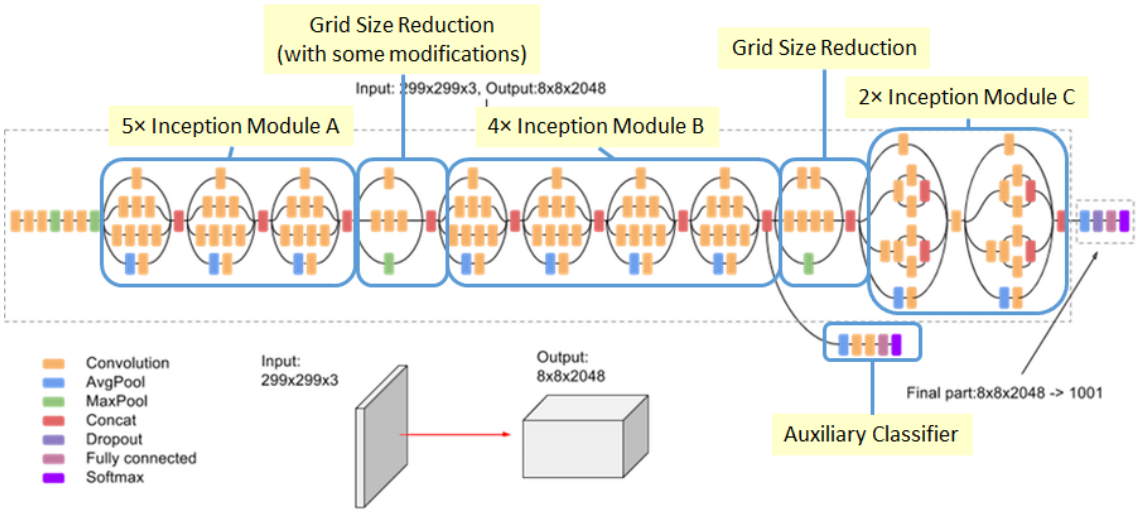


Figure 3.3: Inception v3 architecture: after some standard convolutional layers that aim to reduce the image dimension, there are some replicas of the tree inception modules. Between modules A and modules B, and between modules B and modules C there are two different grid size reductions. Between modules B and C there is also the auxiliary classifier. Image fed to network are reshape in 299×299, and pixel values are normalized in range $[0, 1]$. The last inception module, before an average pooling a dense and a softmax layers aimed to produce a probability distribution among the 1000 categories of images, produces a tensor of shape $8 \times 8 \times 2048$. Input dimension and finally representation of the image dimension are shown in the bottom of the Figure.

size vector, VAE maps the input into a distribution $P_\theta(Z)$ with θ denoting the parameters of that distribution, that will be the latent representation of the input. Moreover what we are interested in, is a way to sample from that distribution a sample z, fed it to the decoder that will compute $P_\theta(x|z)$ that should look like a real data.

In this setting, we can write $P(X)$, our optimization target as

$$P(X) = \int P(X|z)P(z)\ dz \tag{3.7}$$

However equation 3.7 is not yet useful because it has two problems: the integral computation is not feasible at least in high-dimensional space and the true posterior $P(Z)$ is unknown. The solution to the latter problem comes to the main idea behind Variational methods, which is to find a simpler distribution Q that is as close as possible to the true posterior. In our case Q is a simple Gaussian, in formulae $z \sim \mathcal{N}(\mu(x), \sigma(x))$. However, even in this setting, there is a problem: sample operation is not differentiable. Indeed in Neural Network, all the parameters that need to be learnt must have this propriety because, in the computation of their update, derivatives need to be computed to minimize the cost function which is the goal of the network. The easiest way to work around of it, is to sample from a simple Gaussian distribution with 0 mean ad deviation 1, i.e. $\epsilon \sim \mathcal{N}(0, 1)$ and then transform it in to the desired distributions as $z = \mu(x) + \sigma(x) \times \epsilon$. In this way sample operation not involve the variables that need to be differentiate that are $\mu(x)$ and $\sigma(x)$. This technique is called *re-parametrization trick*. In the following this new distribution will be denoted as $Q_\theta(Z|X)$.

For the former problem, the solution is more complex and it relies on Evidence Lower Bound (ELBO) [29]. ELBO is a way to measure how close two distributions are, in this case, the latent distribution and observed one.

Denoting with $log(P(X|\theta))$ the log-likelihood we need to maximize (maximizing log-likelihood is equivalent to maximize likelihood but thanks to logarithm proprieties it is easier to compute) and denoting as $L(x, \theta, \phi)$ the ELBO of a variable x with two different distributions P and Q that have as parameters respectively θ and ϕ, the following relation holds:

$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_Q[log(P(x|z))] + \mathbb{E}_Q[log(P(z))] - \mathbb{E}_Q[log(Q(z))] \leq log(P(X|\theta) \tag{3.8}$$

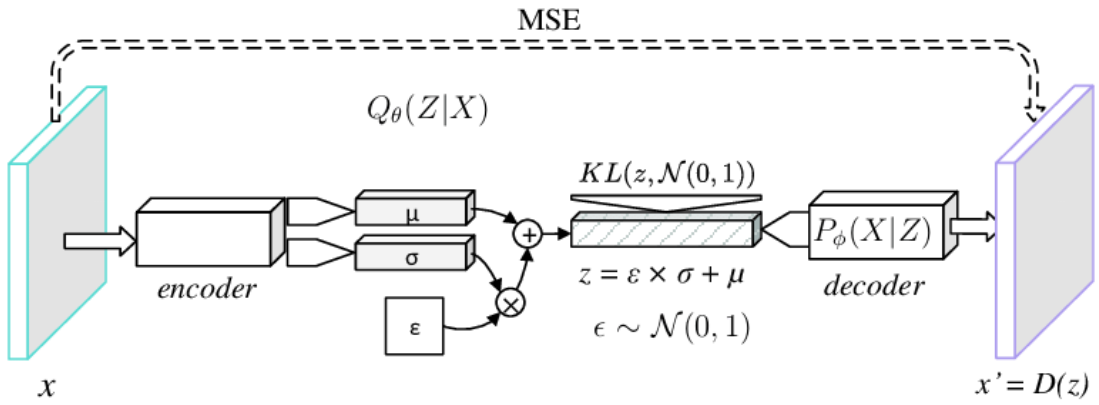


Figure 3.4: VAE architecture: data input x is encoding into a latent representation that is a distribution $z = \epsilon \times \sigma + \mu$. This z and the $KL(z, \mathcal{N}(0,1))$ that acts as regularization, is fed to decoder that tries to reconstruct the input.

From equation 3.8 appears how ELBO is a lower bound for what we need to maximize: thus, even if the distribution is un-tractable, we can maximize it, maximizing the tractable ELBO lower bound.

For our purpose in equation 3.8, term $\mathbb{E}_Q[log(P(x|z))]$ can be rewritten as $\mathbb{E}_Q[log(P(x|z = \mu(x) + \sqrt{\sigma(x) * \epsilon}, \theta))]$ and the rest of the equation that is $\mathbb{E}_Q[log(P(z))] - \mathbb{E}_Q[log(Q(z))]$ is the so called Kullback–Leibler divergence [32], denoted as $KL(Q(z|\phi), P(z|\theta))$ that aims to measure the difference between two generic distributions.

Thus the equation 3.8 can be rewritten as:

$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_Q[log(P(x|z = \mu(x) + \sqrt{\sigma(x) * \epsilon}, \theta))] - KL(Q(z|\phi), P(z|\theta)) \tag{3.9}$$

in which the first term measure decoder reconstruction and the second acts as regularization. Indeed VAE training is done by backpropagation on θ and ϕ optimizing equation 3.9.

In Figure 3.4 is summarized the whole VAE architecture.

Conditional Variational Autoencoders

A Conditional Variational Autoencoders simply adds conditioning to the whole process of VAE: rather then models $P(X)$, it models $P(Y|X)$. This also reflects to the encoder and decoder computation: encoder now computes $Q(Z|y, x)$, decoder

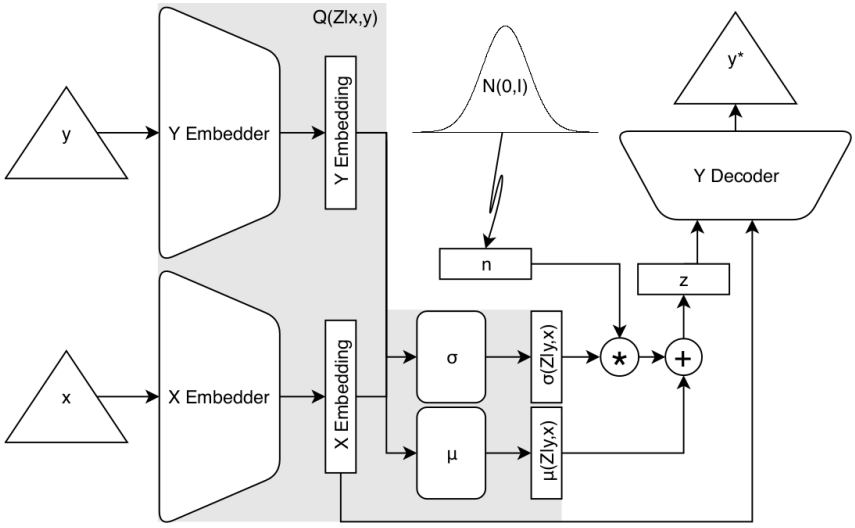


Figure 3.5: Conditional Variational autoencoder for tree-data architecture: x and y are fed in two different components that compute $\mu(Z|x,y)$ and $\sigma(Z|x,y)$. This two, thanks to the re-parametrization trick, are used to generate the latent posterior $P(Z|x,y)$. From that distribution several samples z are extracted and fed along with x embedding to decoder that tries to generate target y.

$P(Y|z,x)$ and the latent prior will be $P(Z|x,y)$. The advantage of Conditional Variational Autoencoders with respect to VAE is that they allow learning a probabilistic mapping from X to Y based on their joint distribution.

Conditional Variational Autoencoders for Tree-Structured data

We are now able to introduce Conditional Variational Autoencoders for Tree-Structured data. It is essentially a Conditional Variational Autoencoder that aims to work with trees rather than with non-structured data.

In Figure 3.5 is shown the full architecture of the model. Given as input a pair of trees $< x, y >$, we define appropriate encoding and decoding models that are respectively able to encode a tree into a flat representation and to handle decoding of a structure form the latent vector. As latent prior it was used $\mathcal{N}(Z; 0, I)$ that is an isotropic multivariate standard normal distribution and therefore the posterior $Q(Z|x,y)$ in this model is modelled as $\mathcal{N}(\mu(Z|x,y), \sigma(Z|x,y) \times I)$.

Both input x and target y are first fed in two different embedder modules to have

fixed size vectors as result. These two vectors are then fed in two different networks computing $\mu(Z|x,y)$ and $\sigma(Z|x,y)$ that are mean and co-variance vectors of the posterior. From this distribution, several samples z are sampled and fed along with x embedding to the decoder that tries to reconstruct target y

One key aspect to stress is that the given pair $< x, y >$ could be with $x \neq y$, differently to usual autoencoder, thus it is possible to perform a tree to tree transduction rather than the usual autoencoder task. Indeed transduction is the task for which the network was designed: obviously, the scenario in which make sense to use condition $P(Y|X)$ is only whenever $Y \neq X$.

Among the components just described tree encoder and decoder should deserve a deeper description while the other ones are the same as VAE. Nevertheless, before describing encoder and decoder, we need to briefly introduce the characterization of each tree that is required from the network.

Tree characterization

In order to explain how encoder and decoder works, we introduce only the minimal notions about the tree characterization needed by the network. Each tree is characterized by a set of node types, thus a tree can be defined as

$$TreeType\{NodeType_1, \ldots, NodeType_k\}$$

Each of this $NodeType_i$ is characterized by:

- Value Type: characterizing the value associated with a node. It is composed of a concrete value and an abstract one. The difference between these two is that concrete one is the representation of the value used during computation in the network, while the abstract is the human-friendly one. For instance, if we model words, we use as abstract value the word as it is used in natural language and as concrete value its embedding. Along with node definitions, network requires also a mapping from each possible concrete value to an abstract one for each possible node types.
- Arity: denotes what is the number of children allowed for a specific node type: it could either be defined as fixed (in case it is 0 this obviously model a leaf) or it could be variable in a specific range of allowed children number.

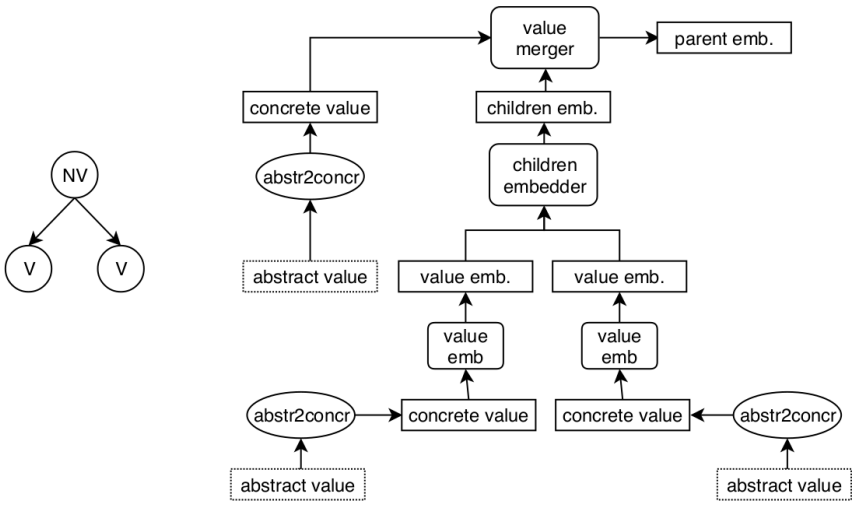


Figure 3.6: Encoder module task: on the left the tree to encode in a fix dimension representation, on the right the unfolded module. Processing is bottom-up, starting to the two leaves for which their abstract values are map in the corresponding concrete ones. Then value embedder transforms these two concrete values in two fixed dimension vectors which are embedding. They are then merged into a single vector by means of the child embedder. Eventually, value merger, merge the vector computed for the leaves with root concrete value to get the root embedding.

Encoder

The encoder, as often happens in neural network for trees, is essentially a Recursive Network that aims to map every possible subtree in a fixed flat dimension vector, that is more manageable. In order to obtain compositionality, the idea of a common embedding space $\mathcal{E}$ among all the possible trees and sub-trees was used. In Figure 3.6 is shown the unfolded encoder. The process is bottom-up, starting from the leaves and traverses the tree until the root is reached. In this figure, we can notice 3 kinds of sub-network that are value embedder, value merger and children embedder. Let analyze each of them denoting whit $\mathbb{V}_i$ the current node concrete value.

- Leaves embedder: network designed to project a single leaf in the embedding space $\mathcal{E}$. For every pair $< ValueType, NodeType >$ such that the current node has zero as arity, i.e. it is a leaf, the network build a sub-network projecting leaf concrete value into $\mathcal{E}$. It achieve a function $\mathbb{V}_i \to \mathcal{E}$

- Value merger: network designed to work whenever in the bottom-up traverse of the tree, an internal node with an associated value is met. First, the node

embedding is computed, then it is augmented by its associated value. It achieve a function $\mathbb{V}_i \times \mathcal{E} \to \mathcal{E}$

- Children embedder: network designed to compress all the children embedding into a single one representing each of them. Here some care is required in order to ensure every time compositionality. For any different node types, a specific subnetwork is built. If the current node type has a fixed arity, let say n, no particular care is required and a sub-network implementing a simple function $\mathcal{E}^n \to \mathcal{E}$ is sufficient. More care is required when network has to deal with variable arity node type: sub-network for this kind of node is shown in Figure 3.7. In order to ensure compositionality, we need to fix a hyper-parameter called cut-arity. If cut-arity is set to k, only k nodes are directly fed to the network. This means that if we encounter a node that has $n \leq k$ children, they are all directly fed to the network, otherwise, if the current node has n children with $n \geq k$, the first $k-1$ are fed directly to the network, while the following ones will be "summarized" into a single vector using an attention network. In the attention network for each extra child c_i with $i \in [k, n]$ a scalar $\mathcal{A}^{c_i}$ is computed , denoting the relevance of that child. After that these coefficients are normalized as

$$a_i = \frac{e^{\mathcal{A}c_i}}{\sum_{j=k}^{n} e^{\mathcal{A}^{c_j}}} \tag{3.10}$$

Computed a_i are then used to perform a weighted sum of these nodes as

$$c^{(a)} = \sum_{j=k}^{n} c_j \times a_j \tag{3.11}$$

$c^{(a)}$ is fed to the subnetwork as k-th input.

Decoder

The decoder is the symmetric module with respect to the encoder: while the latter one, as described, achieves a projection of concrete values in to fix dimension vectors, the decoder starting from a latent fix dimension vector can assign a specific value for a node. With respect to encoder here an additional task is required: besides mapping

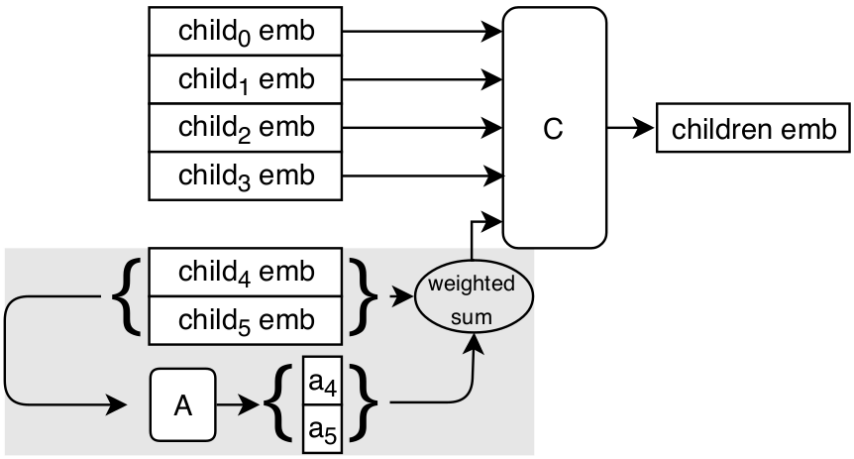


Figure 3.7: Children embedder architecture for variable arity node with a cut-arity of 5. The first 4 children is fed directly to children embedder, the reaming ones are fed to an attention module the summarized them in to a single vector.

any fix dimension vector into concrete value, also the tree topology, i.e. the number of children to generate for each node, has to be learnt.

The Decoder processes the tree top-down, while for encoder this process is bottom-up. What is in common between them is the idea of having a common space $\mathcal{E}$ where all trees, sub-trees,and leaves lie. In Figure 3.8 is shown an high-level perspective of the decoder. In that figure we can notice two new sub-network that is Value Network and Inflater. The extra information that is passed as input along with parent embedding, are for instance parent node type, tree root embedding or tree-structured attention among other nodes embedding.

A crucial point to stress is that here relies the conditioning: current node value is chosen both from Value Network and from Inflater based on the kind of information just enumerated, mainly on the node type.

Let us analyze more in detail the two new sub-networks:

- Value Network: it is a simple network that takes as input parent embedding and extra node information and computes a concrete value for that node. The concrete value is then mapped into the abstract one using the mapping provided with current node type specification.

- Inflater: sub-network that aims to expand tree frontier possibly generating a list of nodes (it can be decided to not generate child at all, i.e. current node will be a leaf). As Value network, it receives as input parent embedding along with extra information. As shown in Figure 3.9a, if n nodes have to be generated first of all

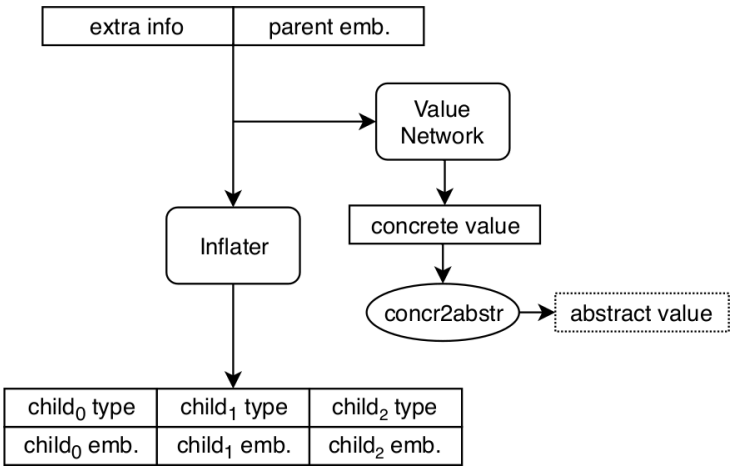


Figure 3.8: Decoder high-level architecture: current node embedding along with some extra information as node type or root embedding, are fed as input to the module. It is composed of a value network that aims to associate a value along with the current node and inflater that aims to decide how and what kind of node to generate as child.

for each of these n children, a node type is chosen, then thanks to inflater module an embedding is associated with them. As in the Embedder case, also here there is a substantial difference between the fixed arity case, that is shown in Figure 3.9a in which no particular care is required because the number of children to generate is known in advanced and the variable arity case in which this number is unknown. In Figure 3.9b is shown an example of the latter case. If cut-arity is set to k and number of nodes to be generated is $n \leq k$ they are processed in the same way described for the fixed arity. Otherwise, if $n > k$, the first k nodes are generated directly, while other ones are all computed by the same recurrent sub-network. It uses as internal state the child number to generate, encoded as 1-of-k, that encode also the type of node to generate and when to stop generation (a special kind, *nochild*, is used for that purpose).

3.2 Different versions of CVAE used

Conditional Variational Autoencoders for Tree-Structured data, as described in Section 3.1.4, was not designed for the specific task of Image Captioning, in particular, it lacks some classical components used in Natural Language Processing. For this reason, we also implement two different variations of this network in order to be more suitable to tasks involving NLP such as Image captioning.

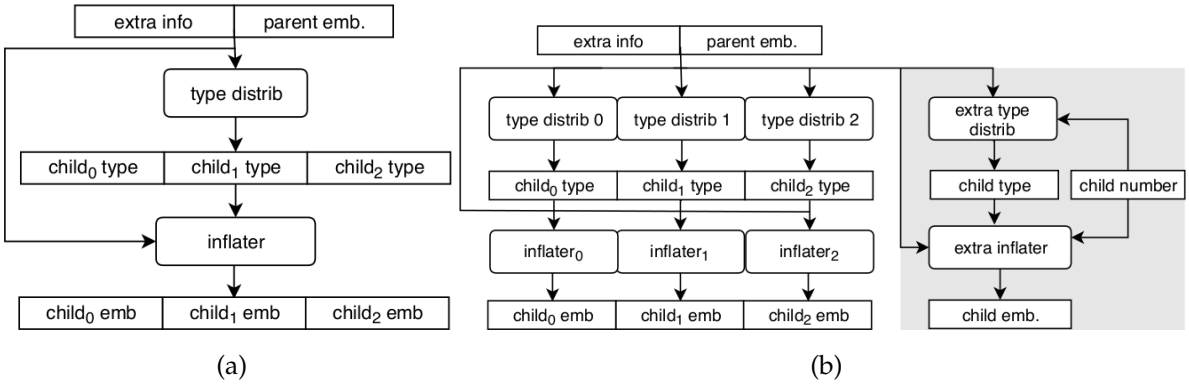


Figure 3.9: Inflater network: in (a) is shown the fixed arity case for which no particular care is required and in (b) variable arity case witch cut-arity set to 3. The fist 3 nodes are generated by dedicated sub network, while the following ones are all generated by the same recurrent sub-network (in the grey area). This recurrent network has as internal state child number encoded as one-of-k that aims to express the next node type to generate and when to stop generation with a special type *nochild*

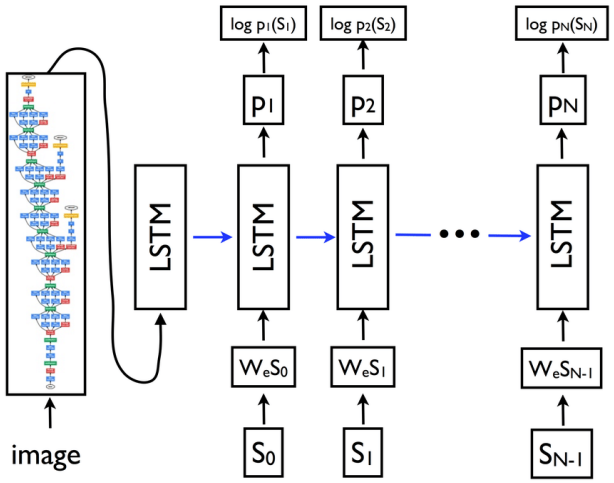


Figure 3.10: *Show and Tell: A Neural Image Caption Generator architecture*. The input of the network are images that are processed with a CNN: the hidden representations obtained are then fed as first timestamp to the LSTM. In the following time stamps, let say the $i-th$ the input to LSTM is the previous predicted word (it' categorical representation S_{i-1}) multiplied by W_e, the word embedding layer, in order to obtain its embedding. This just obtained embedding is then fed to the LSTM. The output vector of LSTM is fed to a softmax layer in order to have word prediction distribution p_i among all the possible words for the $i-th$ timestamp. Finally cross-entropy between predicted words and target words in computed. Image is taken from original publication [58].

As said the main reason of these modifications is words processing: in the standard version of this network, all the sub-networks are composed of Dense Layers that are not suitable to process words. Thus with the standard version, we represent target words as word embedding taken from a large pre-trained dictionary.

For the reasons just reported, all the modifications tested regard only the decoder part. The first modification was to introduce a word module inspired by *Show and Tell: A Neural Image Caption Generator* [58]: we represent the words as a category, i.e. with one-hot vector, and we process them using an embedding layer that transforms the categorical representation into a dense one. Finally, we introduce a RNN, in this case a LSTM to predict the i-th word based on predicted $(i-1)$-th. In Figure 3.10 it's shown the main architecture of Show and Tell: A Neural Image Caption Generator.

In our work, we have image latent representation replaced by the tree latent representation as first input to the LSTM and as input for a generic i-th timestamp we have along with the word embedding of the previous predicted word, also the embedding of the POS tag node that is the father of current word leaf in the final generated tree.

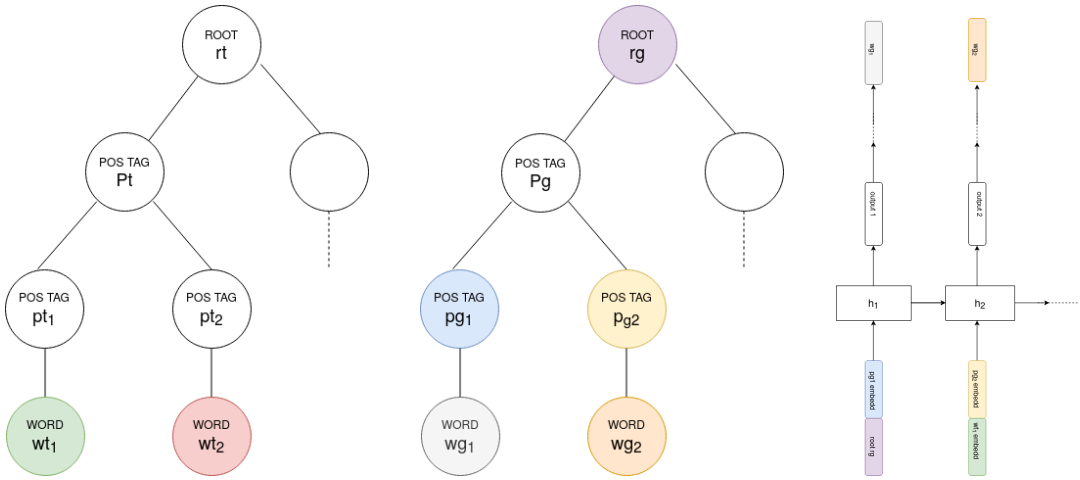
In Figure 3.11 is inserted ad an help for the reader and shown the task done by the RNN in our model.

In order to avoid confusion, we want to stress the different nature of the word embedding and tree node embedding: they are different and they are computed from different parts of the network: POS tag nodes embedding are computed by the Conditional Variational Autoencoders for Tree-Structured original sub-networks, while word embedding are computed by the embedding layer just introduced in the word module.

As said, in deciding the i-th word to generate, as input along with the word embedding of the $(i-1)$-th word generated we fed to the RNN also the POS tag embedding of the parent of this word leaf in the generated tree. This is possible because another modification we made to the network is to generate all POS tag nodes before the beginning of the words generation in order to have available them when we generate the words.

These choices were made mainly for two reasons: for homogeneity with the standard part of the network and because this additional input could give useful information in deciding the word to generate for a specific timestamp.

The last point concerns test time: in word processing there are two classical way to generate captions that are sampling and beam search. We choose to use sampling because the last one is more complex as inference procedure and how much the bream



(a) (b) (c)

Figure 3.11: RNN processing of words. In (a) is show part of the target tree, in which each node is named with a ending t, to underline that they are targets. In (b) the corresponding generated part of the tree in which each node is name with a ending g, to underline that they are generated ones. Finally in (c) is show how RNN processes nodes: in the first time stamp it receive as input the latent input representation, concatenated with the value of the parent node that RNN is currently generating (wg_1) that is pg_1. From the second timestamp, input latent representation is substitute by the previous word target embedding that in this case is wt_1. Even in this case it is concatenated with parent value of the node that is going to be generated, that is pg_2. Outputs of the RNN are further process to have word predictions, in this Figure wg_1 and wg_2.

search can affect the qualities of the produced captions is out of our interest. Sampling means to make, rather than in beam search, every time the best possible local choice. This means that at inference time, for the i-th timestamp is chosen every time the word predicted as more probable, which is the word with LSTM output higher.

Since also the second modification concerns the introduction of a word module, we later refer to this first modification as *NIC words module*, while the second modification will be referred as *Attention words module*.

Indeed in the second variant of the network, we introduce a word module with an attention mechanism: we want to test a module chosen to be as different as possible from the first one, to evaluate how the chosen word module can affects the qualities of the generated captions.

Some of the modifications introduced for the first word module are maintained such as representing words as one-hot vectors, the presence of an embedding layer, a

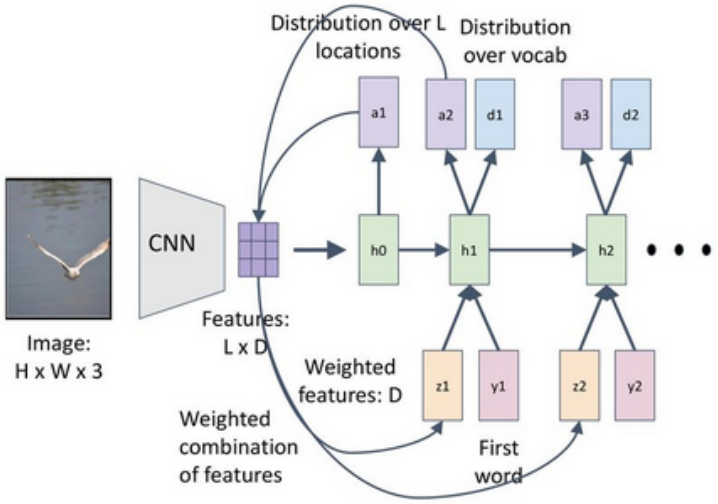


Figure 3.12: *Show, attend and tell: Neural Image caption generation with visual attention* architecture. The encoder result is a grid of features of dimension $L \times D$. The attention mechanism, at each timestamp, extracts from the grid a D dimensional vector that is fed at each timestamp to the RNN as image representation, along with the previous word predicted (in the first timestamp this input is zeroed). This D dimensional vector is called context vector in this work because it contains indications about the image parts in which the model has to focus its attention to generate the next word in the caption.

RNN which takes as input both previously predicted word and POS tag value of parent node and finally also the sampling operation for the testing time is the same. In this modification we follow *Show, attend and tell: Neural image caption generation with visual attention* [61] to implement the word module. The biggest difference with the previous module is the attention mechanism that is a Bahdanau Attention [5]: its contribute in the original work is summarized in Figure 3.12. Starting from a $L \times D$ grid of features, that we could imagine as L vectors $a_1, a_2, \ldots, a_L$ of dimension D, the attention mechanism reduce this grid into a vector of dimension D. In our work was used the "soft" attention variant of it in which to reach this goal, a weighted sum of the L dimensional vector is performed as $z_t = \sum_1^L \alpha_i a_i$ were t index the different timestamps (the context vector z is different for each timestamp) and α_i are the coefficients learnt by the attention mechanism.

With this mechanism, we fed the image representation at each time stamp, differently from the previous work. Moreover, in this word module, we use as RNN, a GRU [12]. After GRU output we insert two different dense layers, instead of just one as in

encoder	decoder
CVAE encoder	CVAE decoder
CVAE encoder	CVAE decoder + NIC words module
CVAE encoder	CVAE decoder + attention words module
CVAE encoder	flat NIC decoder
flat encoder	CVAE decoder + attention words module
flat encoder	CVAE decoder + NIC module
flat encoder	flat decoder

Table 3.1: Different architectures tested. Standard encoder/decoder stand for original Conditional Variational Autoencoders for Tree-Structured encoder/decoder, NIC module stands for *Show and Tell: A Neural Image Caption Generator* word module and attention module stands for *Show, attend and tell: Neural Image caption generation with visual attention* word module.

the previous word module, before the softmax that produces a probability distribution for the current word generation.

These two are the main modifications to the network we tested. However, in some experiments, we tested just the encoder part of the network for structured data and a sequential decoder or the other way round that is a flat encoder and structured decoder.

For instance, we tested this second modification, just described, with all the components included the attention module, when we use a flat encoder: indeed if we have as encoder a CNN we get as latent representation for each image a grid of features, let say $N \times emb\_dim$ where emb_dim is the embedding dimension, and the attention mechanism aims to shrink it into a single dimension vector, specifically a emb_dim dimensional one. On the other hand, when we have the structured encoder described in Section 3.1.4 we get as latent representation of a single image, a vector of size emb_dim, thus it makes no sense to use the attention mechanism.

In Figure 3.13 is underline the role played by the attention mechanism in this second word module.

A summary of the different architecture tested was reported in Table 3.1.

In the following, we refer, as done in Table 3.1, Conditional Variational Autoencoder for tree structure data as CVAE for brevity.

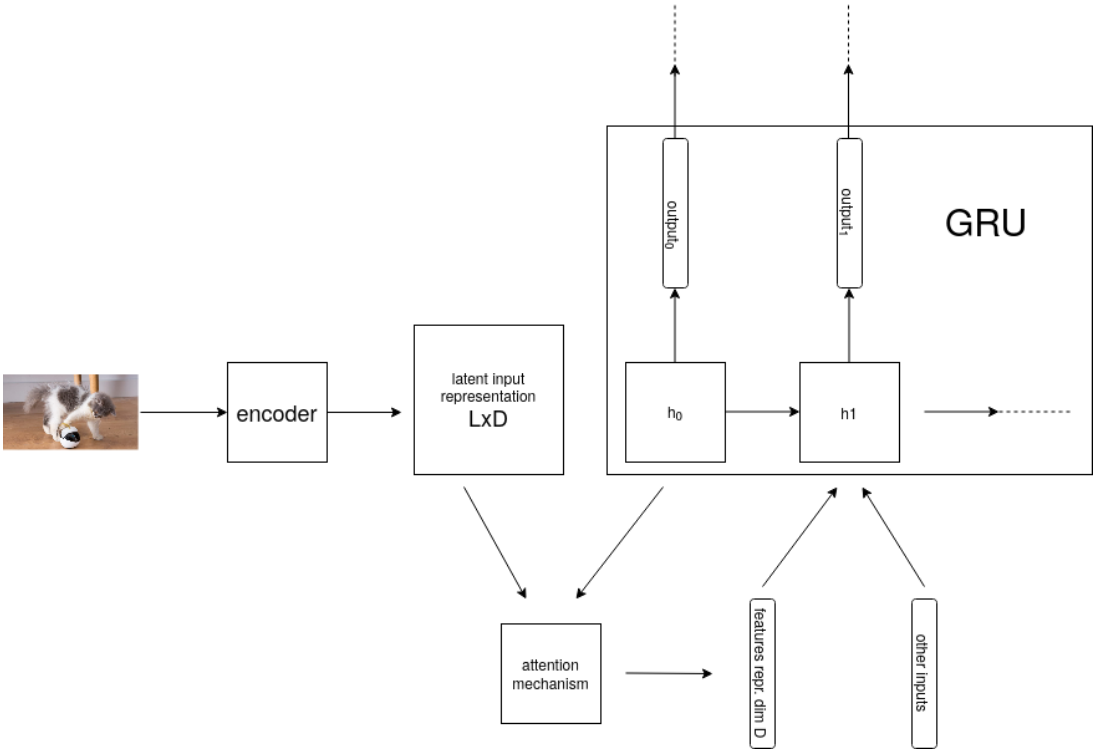


Figure 3.13: Attention contribute in the second word module in the second timestamp of caption generation. From left to right image input is transformed in a $L \times D$ grid of features by the encoder that is a CNN. The attention mechanism receives, at each timestamp, as input, h_{i-1} the hidden state of the RNN at the previous timestamp, in this case, h_0 and the whole image representation. It shrinks the $L \times D$ image representation into a D dimensional vector that contains the information on which the RNN must focus its attention to generate the current timestamp word. Along with this D dimensional vector, the RNN receive also the other inputs that are previously word embedding and embedding of current POS tag node parent.

3.3 Data representation

In this paragraph will be presented how the models described in Section 3.1 and 3.2 were used in this work. Remember that our task is image captioning thus inputs are images and targets are sentences. The aim of this work is to represent both as trees and to use CVAE for Tree-Structured data to learn a mapping between the two.

3.3.1 Input processing

Concerning image processing, the main idea is to obtain a tree representing a hierarchical segmentation of the given image thanks to GLIA and then label the nodes

of the obtained tree, using convolutional information coming to a CNN.

Generation of input trees

First of all to use GLIA we need gPb applied to images in order to retrieve some meaningful initial super-pixels. As sketched in Section 3.1.1, rather than feed boundary probability map to Watershed as it is, GLIA prefers to feed a blurred version of it. This choice is motivated since a blurred map produces less segments then a non blurred one. Indeed after watershed applications, there was also the necessity to merge the smaller super-pixels found, in order to reduce their number.

In Watershed the only relevant hyper-parameter is the level of water and we select it as 0.1.

We use as blurring process the Gaussian blur commonly used in computer vision pre-processing. It replaces pixel values with a weighted average of neighborhood pixel values following the formula

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.12}$$

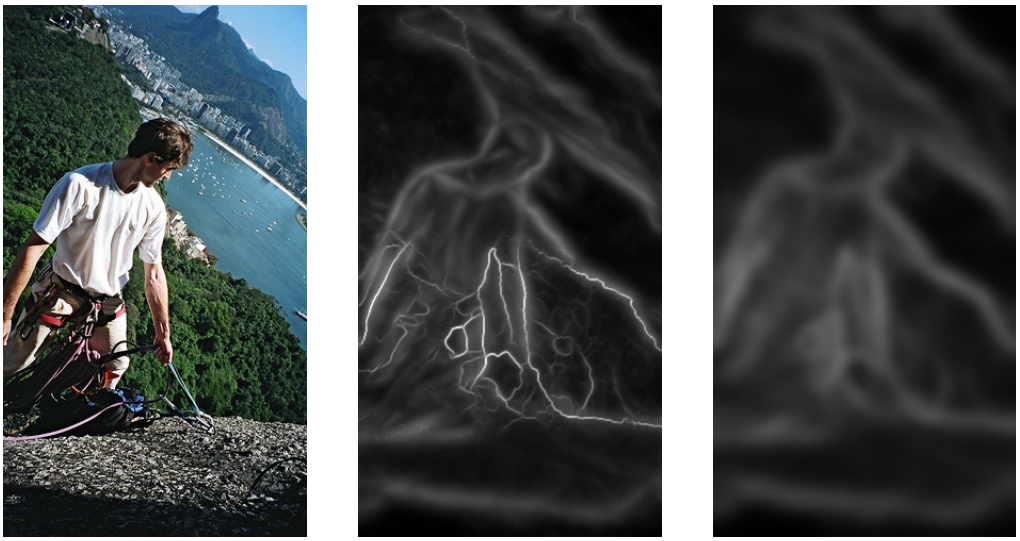
where x and y are the distance in the horizontal and vertical axis of neighborhood pixels to consider. The only parameter to chose is σ, Gaussian standard deviation, that we could imagine in this context as the amount of "blur" added to the picture: we select for it a value of 6.

In Figure 3.14 is shown an example of original image, the computed gPb and its blurred version used during the segmentation process.

Watershed level of water and σ of the Gaussian blur were chosen with the following criteria: a set of possible values for them were tested in order to produce initial superpixels for both sample images and for their horizontally reflected ones: after that the whole segmentation is performed with the just produced superpixels. Horizontally reflected images were used in order to select as σ the value for which the two segmentations are closer or in other words the one for which the resulting segmentation is more robust to that transformation.

In this process we notice that a major impact on the robustness of the results is played by σ, specifically, the higher it is and the less segments are found although they are more robust.

Concerning the watershed level instead, values close to 0.1, as 0.2 or 0.05, also produce a good segmentation, but slightly less robust.



(a) (b) (c)

Figure 3.14: Example of application of gPb in this work. In (a) an image taken from Flickr8k train set, in (b) the gPb computed on this image and in (c) the blurred version of (b) that are the image use to perform GLIA segmentation.

In Section 3.1.1 we also sketched that to use GLIA we need to train a boundary classifier: for that task, we use as input data *BSDS500* [3] and as model a random forest, as done in GLIA original work.

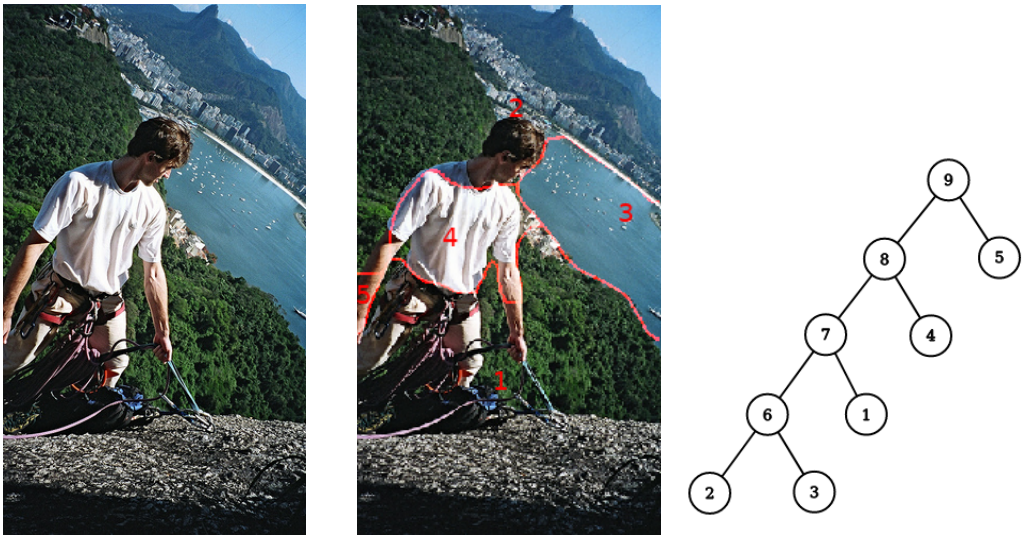
Concerning the rest of the segmentation process, there is a huge number of hyper-parameters to possibly test. However since our goal for this step was to find a robust representation and as said we achieved enough satisfying results only with the values chosen for σ and watershed water level, default GLIA parameters were used.

In Figure 3.15 is shown an example image and its segmentation.

Moreover, two changes were made to the algorithm: the first one regards what we get from this step. Specifically, along with the original output which is a map describing the segments found, we also return the merge tree describing only final segments.

More in details the first one is a matrix having same width and height of original image but with only one channel and in which for each element (x, y), a value s is associated if that pixel belongs to segment s: we later refer it as segments map. The second one is the merge tree in which all the descendants of a node that is selected as final segment, do not appear.

The second change we did to the algorithm, concerns a modification of the way it works: in the resulted tree, the common setting is that for each node v_i^d, the two



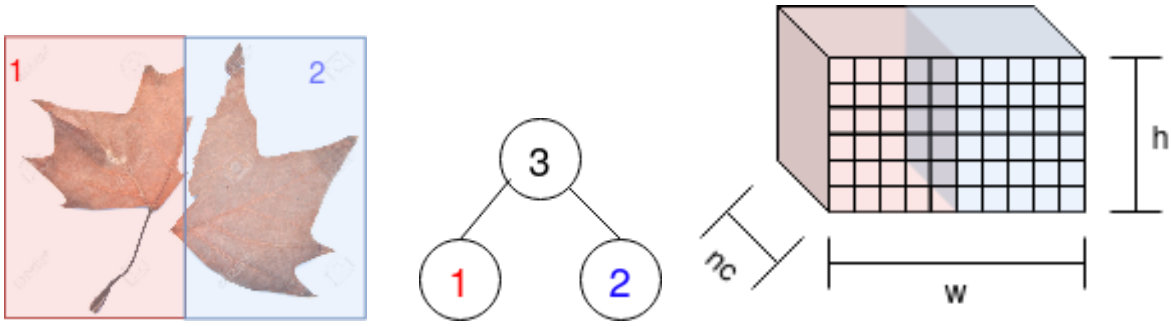
(a) (b) (c)

Figure 3.15: Segmentation example for the image also used in Figure 3.14 . In (a) the original image, in (b) the segments found by GLIA and in (c) the relative merge tree in which leaves node, i.e. node from 1 up to 5 are segments found in the original image, while the other ones describe regions that are union of the ones described by their children. The merge tree shown in (c) is unbalanced: usually each node in the merge tree has as children a leaf node and an internal node. Each node in this tree need, in a second time, to be labeled with convolutional information to represent the visual content

children v_j^{d+1} and v_k^{d+1} are one an internal node and the other one is a leaf, i.e. a final segment. Sometimes this doesn't happen and the two children v_j^{d+1} and v_k^{d+1} are both internal nodes. We suppose that in this scenario the constraint of the merge tree, to be a binary one, doesn't allow to put two different segments in the same level of the hierarchy. For this reason if in the tree we found a node v_i^d, with the two children v_j^{d+1} and v_k^{d+1} both internal node, we remove them and we put as v_i^d children the ones of v_j^{d+1} and v_k^{d+1}, that are v_i^d grandchild in the original tree. This modification has a consequence that the merge trees computed by the algorithm are no more binary ones but the ariety of its nodes could potentially be an higher value.

Label input trees with convolutional information

So far we have obtained a tree describing the hierarchical elements in the image. However, these segments are denoted just with integer labels assigned to each node



(a) (b) (c)

Figure 3.16: Labeling process with alexNet. In (a) is shown a sample image for which only two segments are found, as described by the computed merge tree shown in (b). (a) is just a sample image because as we said we decided to not use images in which we found just one or two segments but we used this image in this example for its simplicity. The highlighted red and blue sections are used to show the two found segments that correspond roughly to respective the left and red parts of the image. In (c) is show the resulting tensor of dimension $h \times w \times nc$ in which with red and blue colors are used to highlight the vectors that are influenced by the corresponding pixels belonging to the two found segments.

in these trees, representing the region that this node describes: as they are these trees do not give any information about the visual content of the image. For this purpose, the following step is to label each node in the obtained trees with information coming from a CNN. Two different labelings were performed using the two CNN described is Section 3.1.2 and in 3.1.3 that are AlexNet and Inception v3.

Let us first describe the way the labeling task is performed with AlexNet and later we will describe the minor change we made to the process when we use Inception v3.

First of all, we run a forward in order to retrieve the features extracted by AlexNet for a specific image (we use a modified version of the network in which the results are the extracted features, ignoring the Dense part used for classification). Then for each segment found in the image, we select from that features, which for a single image is a tensor of dimension $h \times w \times nc$, just the elements of the tensor influenced by the pixels contained in the region we are currently analyze: the best way to thinks about this elements in the tensor is as $h \times w$ vectors of dimension nc.

In Figure 3.16 it is shown an example of selection of the proper vectors among the ones in the final tensor, for each segment found in the image.

Once we have gathered all the "vectors" influenced by the pixels belonging to the current segment, we merge them into a single one computing the mean vector among them.

We left the third dimension of the resulting tensor nc, which are the number of channels in the image representation, as a variable, because its value depends from which layer we want to take as final output or in other words from which layer we took the information to label our trees. We try for that, in the following experiments, both the output of the 3rd and the 5th convolutional layer of AlexNet.

Up to now, we have labeled nodes describing final segments in the images that are leaves. The last step is to handle nodes which do not describe any final segments that are internal nodes. This is needed because each node in the tree must be processed by the CVAE, thus we need to label each node with some data. For this purpose we could try several strategies: we prefer to take max-pooling of each child vectors inspired by the max pool operator usually used in CNN. The result of max pooling applied on a set of vectors V in which each of them has the same dimension, is another vector r with the same dimension of the ones in V in which each component, is defined as $arg\ max_{v \in V} v_i$.

In figure 3.17 is shown an example with a tree and the associated values for each node.

The other CNN we use to label trees is Inception v3. With this CNN we can't apply directly the process used for AlexNet, because as described in Section 3.1.3 the Inception modules concatenate different convolutions with different shapes in the same layer. This makes each element of the result tensor, i.e. the $8 \times 8 \times 2048$ ones, depending on each pixel of the image. Obviously if for Inception v3 we use the exact same strategy we used for AlexNet we get as result a tree with nodes labeled every time with the same vector.

For this reason with Inception v3 we have proceeded in this way: for the leaves, we use the exact same strategy as before using inception v3 until the last convolutional layer before inception modules that produce vectors of dimension 192 and we also use max-pooling to label all non-leafs nodes, as done with AlexNet, except for the root.

The root is labeled with the original inception v3 CNN: this means to feed to the network the image pre-processed as described in Section 3.1.3. Root is labeled with the average pooling of the layer coming from the last inception module that reduces tensor of dimension $8 \times 8 \times 2048$ in just a vector of dimension 2048.

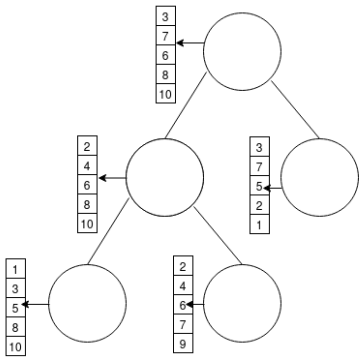


Figure 3.17: Example of max pooling operator applied on internal nodes of the merge tree. For each node in the tree we have a sample data associated, shown on its left, that is a vector of 5 dimensions. For leaf nodes, the value is the one computed by CNN used in the labeling process described. For each internal node, instead, the associated value is the max-pooling of its two children: for each of the five components of the vectors data, the component of the max-pooling result is the bigger among the ones that are children of this node. We report as data an integer vector just for readability: the true associated values to nodes in our trees are vectors of Floating point number.

In Figure 3.18 is show an example of tree labeling of the same image in Figure 3.16 with inception v3.

We chose to follow this strategy because the information coming from leaves are not enough abstract to have a meaningful representation of the image in the tree. Thus we need to include also global information as we do in the root. We choose to include also the average pooling because a $8 \times 8 \times 2048$ representation for a single node would have been too big.

At the end of this process, we have trees that describe images, in particular, each node in the tree describes a region and it is labeled with convolutional information that represents its visual content. Each dataset we used can be labeled either with AlexNet or Inception v3 with the corresponding strategy.

Moreover, this tree represents a hierarchy: upper nodes, and in particular upper leaves describe segments at top of the hierarchy, while the lower leaves describes segments at the bottom.

The result of this process is what we use as inputs for Conditional Variational Autoencoders for Tree-Structured data.

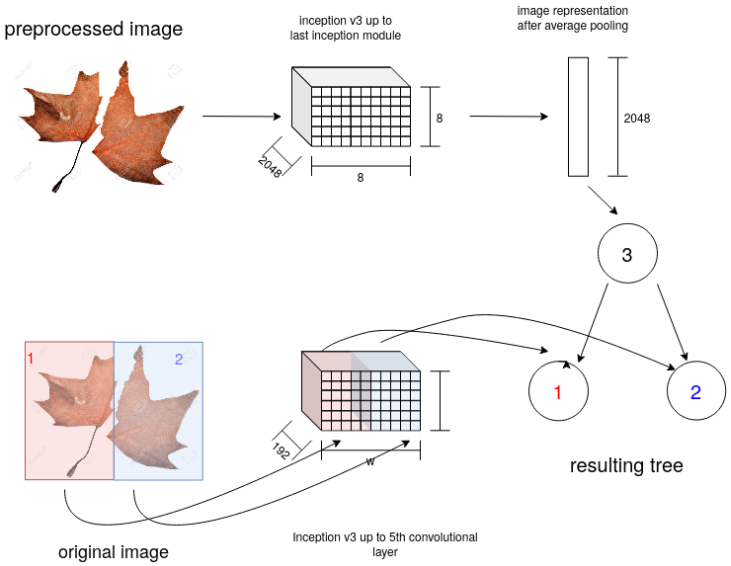


Figure 3.18: Example of labeling with Inception v3 CNN. In the low part of the image are shown the original segmented image with the two segments found, processed by Inception v3 "cut" to the 5th convolutional layer, that is the last one before inception modules. This information is used to label tree leaves. All internal nodes, except the root, are labeled as max-pooling of their children thus they are influenced by this first CNN. In the upper part of the image, there are the pre-processed image as required by Inception v3 thus having a shape of $299 \times 299 \times 3$, and Inception v3 "cut" to average pooling before the Dense part. The information extracted in that way are used to label the root of the tree.

3.3.2 Target processing

The target of our task are sentences, specifically sentences describing the images, which are captions. We need to represents also captions as trees: the more natural way to do so is using the so-called parse trees.

A parse tree is a tree in which the syntactic structure of the sentence is expressed: as illustrated in Figure 3.19 it is a tree in which the internal nodes are labeled by specific tags that aim to underline the syntactic role of a word/group of word in the sentence while leaves are the words belonging to the sentence described. Tags are called part of speech tag (shorten in POS tag): they are represented in the tree as contractions, for instance, all parser tree roots are labeled as S that stands for sentence.

In this tree if a node is labeled with a particular POS tag, let it be P, this means that every leaves, which represent words, descending from that node belong to the same part of speech, specifically P. For instance more meaningful tags then S, are in

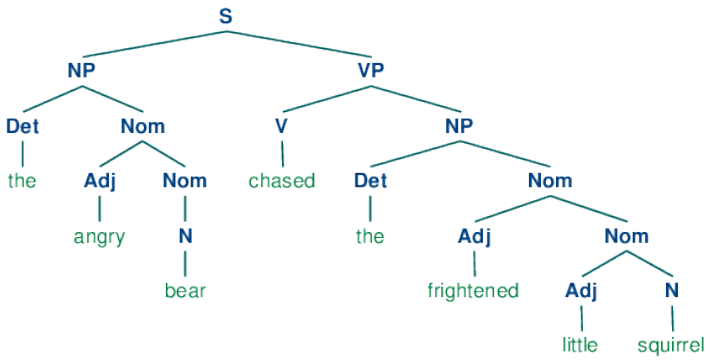


Figure 3.19: Parse tree: leafs nodes are words while internal nodes are POS tags. Last ones are indicated by contractions, specifically in that tree: *S* stands for sentence, *NP* stands for noun phrase, *VP* stands for verbal phrase, *Det* stands for determiner, *Nom* stands for noun, *V* stands for verb, *Adj* stands for adjective. For each POS tag all the words rooted by this sub-tree, belong to that same part of the sentence.

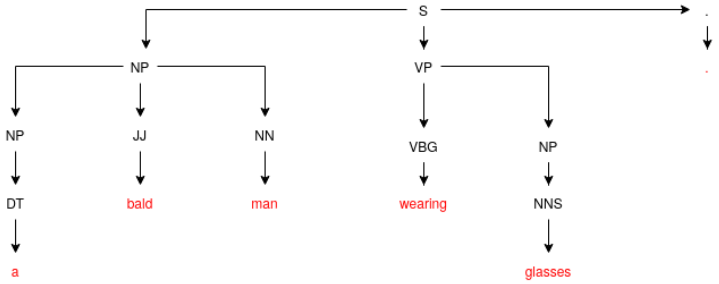


Figure 3.20: An example of parse tree obtained from Flickr8k dataset. In the upper part of tree, black nodes that denote POS tag, while in the low part of the tree there are red node that denote words.

the second level of the tree in Figure 3.19, that are *NP* that stands for noun phrase, i.e. a phrase that has as head a noun and *VP* that stands for verbal phrase, i.e. a phrase that is composed of at least one verb.

Clearly, some of this TAG can occur only in the top of the tree as for instance, *NP* or *VP* because they indicating a phrase, which is a group of words thus they need tho have some children in the tree. Other ones as *Adj* or *Det* indicate the role played by a specific word in the sentence and thus they appear only as leaf parent.

In order to obtain such trees, captions associated with images were fed to Stanford parsed [30], specifically using the standard Lexical Parser.

In Figure 3.20 is shown, with a different visualization then Figure 3.19, a parse tree obtained with Stanford parser and used as target of one input image in our experiment.

However even if it possible to represent such tree as a data structure, as it is just described, it is not a data structure that can be fed to a Neural Network.

In order to reach this goal, some representation changes, and some design choices have to be made: we chose to represent leaves, i.e. words in the sentence, using a pre-trained word embedding [44] when using the standard architecture of Conditional Variational autoencoder for Tree structured Data. It is a word embedding of 1 million words trained on a corpus composed of 16 billion tokens. Each word is represented as a 300 dimensional vector, a typical dimension for that task.

In all the other variants of the network, which are the modified versions with the addition of a word module, we represent words as *1-of-k* or alternatively *one-hot* vectors: if in the dataset are present n different categories, we represent each of them as a n dimensional vectors in which all the component are 0, except only one dimension that is set to 1. To each dimension is associated a particular category in order to express, using the position of the component set to 1, which of the possible n different categories is the current represented.

Internal nodes, i.e. POS tags, are represented in each scenario as 1-of-k vectors.

These trees just described were fed as targets to Conditional Variational Autoencoders for Tree-Structured data.

Chapter 4

Experiments

A set of experiments have been conducted to compare image captioning performed with standard "flat" representation of data and with tree-structured data. In these experiments, we used two different data set: Flickr8k [27] and msCoco [35].

The first one is composed by a set of 8000 images taken from Flickr, a popular image hosting site. These images were chosen from 6 different Flickr groups to have a huge variety of scenes and objects among the whole dataset. Each image is provided with 5 different captions that are used as reference targets to evaluate the generated captions qualities.

The second one also contains images of common objects in their usual context (Coco stand for common objects in context). As opposed to the Flickr8k, the 2014 release of coco (the one used in this work) contains 82.783 train, 40.504 validation and 40.775 test images that are approximately a division of $\frac{1}{2}$ of total data for train set, $\frac{1}{4}$ for validation set and another $\frac{1}{4}$ for test set. This dataset has an important feature that we used in some experiments namely each image is labeled with one or more labels out of the 91 available: these labels are all common object categories.

In the rest of the chapter first of all, in Section 4.1, we describe the metrics used to evaluate resulting tree qualities and in the following two sections we describe in more details the two datasets used while we report the best results found for each of them for each different model used.

All the code used for the experiments is hosted on git-hub [1] [2] .

In order to reduce search complexity, some hyper-parameters were fixed in all

[1] `https://github.com/dave94-42/image_captionig_tree2tree`

[2] `https://github.com/dave94-42/image_captionig_tree2tree_input-target_processing`

experiments and we do not test any other values, such as learning rate fixed to 0.001 as well as any other hyper-parameters peculiar of CVAE.

We use as regularization technique L2 regularization which consists in adding to the cost function, a term $\beta \sum_i w_i^2$ where i indexes all the parameters in the network and sparsification of the latent representation, which consist to add another time a term to the cost function namely $\lambda \sum_j |h_j(x)|$ that penalizes huge latent representation norms. Along with these, we use less frequently Dropout either along with the other regularization techniques just described or as the only regularization technique. For this reason, for each experiment, we also report the coefficients β and λ used respectively in L2 regularization and in sparsifcation, as well as the drop rate which is the probability to "disable" a neuron in Dropout technique.

We ran all experiments using hold out search as testing technique and Adam as optimizer.

4.1 Results evaluation

Using trees as targets, there is no straightforward ways to evaluate the quality of the match between a predictive and the ground truth. Fortunately, along with CVAE, some heuristic to score the quality of obtained results were provided: in some of the following experiments, we use these metrics to evaluate trees generated goodness along with more traditional metrics for image captioning as BLEU scores to evaluate just the captions goodness.

BLEU score is a metric originally defined to evaluate machine translation but it is also used in image captioning tasks. It compares machine output against a single sentence or a set of human reference targets. In particular, there are many different BLEU scores, each one indicated as BLEU-n where n is the length of the $n-$grams considered. For instance, BLEU-1 score is a fraction (so it's a number between 0 and 1) defined as $\frac{mw}{tw}$ where mw is the number of generated words that are also found in at least one of the references provided and tw is the total number of words generated. BLEU$-n$ with n greater then one is computed in the same way but this time we don't consider single words but n consecutive words. In our experiments, according to the other image captioning works, we compare the generated sentence against 5 reference captions that are provided along with each image in the used datasets.

Let us now introduce the heuristic metrics used to evaluate generated trees qual-

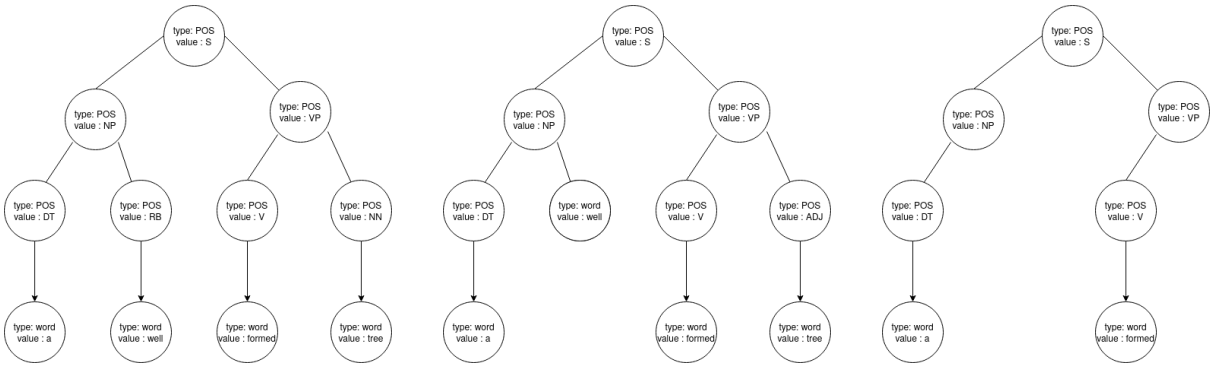


Figure 4.1: . BCPT computation example. In (a) and in (b) two sample parse trees and in (c) the relative computed BCPT tree. In the last one there are only nodes that have the same type (second nodes on 3rd level differ in type, namely for (a) is a POS node and in (b) is a word node) and same parent (even if 4th nodes in the 4th level in the sample trees are equal, they have different parents, namely both types is POS but in (a) value is NN and in (b) value is ADJ). In this latter case the the visit is left for the subtree rooted by this node).

ities.

The major difficulty in comparing two trees is that their topologies could be different and even if the number of children for a specific node is the same in both target and generated trees, these nodes could be of different types or have different values.

To overcome different topologies problem, we use *Biggest Common Prefix Tree (BCPT)* to compare how two trees with potentially different topologies are close to each other.

Let y be the target tree and $\bar{y}$ the generated one. We visit the tree top-down to compute BCPT. Starting from the two roots we try to match every node. Match in this context means that node types and parent nodes are the same for the two nodes. On two generic matching nodes, we compare the children starting from left to right and we recursively repeat this process for the matched ones, while the tree visit, for not matched ones is left. The BCPT tree is the tree composed just by the matched nodes.

An example of BCPT computation with two sample trees is shown in Figure 4.1

If we denote as $|\cdot|$ the operator counting number of nodes in a tree and as $BCPT(x, y)$ a function that given two trees returns the BCPT computed on them, our

structure accuracy score is

$$acc_s = \frac{2|BCPT(y, \bar{y})|}{|y| + |\bar{y}|} \tag{4.1}$$

Once we have a metric to measure the structural error, we can easily define a metric also for the value error accuracy. Denoting as $matched\_nodes$ the nodes in the BCPT for which the value is the same that in the target tree and as tot_nodes all the nodes in BCPT, we define as value accuracy:

$$acc_v = acc_s * \frac{|matched\_nodes|}{|tot\_nodes|} \tag{4.2}$$

We later refer to the two just metrics described as accuracy or unsupervised loss respectively for structure and values.

4.2 Flickr8k

Flickr8k dataset has a usual split of 6k images for train set, 1k for validation and 1k for test. We use this usual split for this dataset. We pre-process inputs and targets following the procedure described in Section 3.3.

An important caveat is that in the process of image segmentation, with the hyper-parameter we fixed for this stage, a subset of images end up in segmentations with just one or two segments found. As specified in Section 3.1.1, image segmentation was described by a tree: this results with only one or two segments is reflected in a generate tree with only root or root and two leafs children.

In the case described above, we prefer to drop these images from our transformed dataset, leaving training set with 5343 images, validation set with 909 images and test set with 897 images. Statistically in the process of transformation from "flat" images to trees describing the hierarchical segmentation of the images, we lose about $\simeq 10\%$ of the data.

These statistics are summarized in 4.1

With the dataset described above, we ran a set of experiments to evaluate the qualities of obtained trees and the qualities of captions. Finally, with this dataset, all the experiments were run using trees labeled with Alexnet.

Size	Training set	Validation set	Test set
Original size	6000	1000	1000
Transformed size	5342	909	897

Table 4.1: Dimensions of train, validation and test set of Flickr8k before and after transformation applied

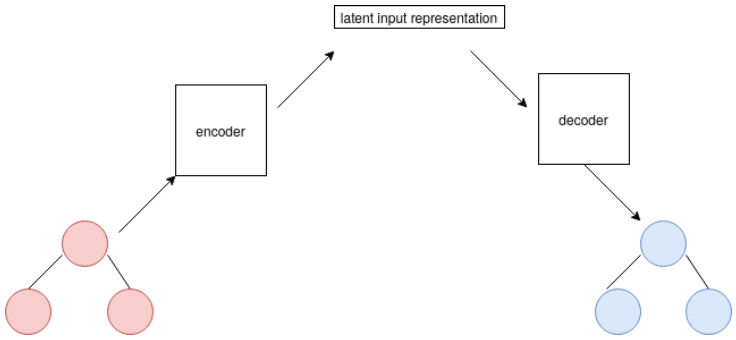


Figure 4.2: High level architecture of CVAE. Red tree is input tree that is transformed through the encoder in a fixed size vector that is the latent input representation. This last one through the decoder is then processed to generate a tree, shown with blue color.

4.2.1 Experiment Results

We first ran a set of experiments with the CVAE described in Section 3.1.4 for which a schematic summary of its architecture is reported in Figure 4.2.

Each tested model was able to reach a good structure reconstruction accuracy: using the structure accuracy described in 4.1 we were able to reach a score of at least 0.8 for each trained model.

This is a very good result that exceeded our expectations: be able to reconstruct the tree topology is considered a very difficult task, indeed most of the works cited in Section 2.3 use some tricks to have a fixed tree structure and thus avoiding tree structure reconstruction problems.

We analyzed value accuracies using metrics described in Section 4.1 but we have a drawback with them: the number of total values to analyze is not stable because this value relies on the nodes in the BCPT. These trees, as described in Section 4.1, are the ones in which we have only generated nodes that match type and parent of the target nodes. This number can largely vary during training, making hard to have a real accuracy assessment.

For this reason, we report in the following these metrics for POS tag nodes, but for word nodes, we report also the BLEU score that is a more detailed indication of the captions goodness. For the last consideration, all the experiments were made using early stopping on BLEU-1 score.

We report in Figure 4.3a , 4.3b and 4.3c respectively, the number of matched POS tag nodes, the number of total POS tag nose and the percentage of matched POS tag nodes out of total. From the last plot appear that without taking into consideration the number of matched and total nodes analyzed, the ratio between these two is more or less stable around 0.6, with a little growth during training.

As described in Section 3.3.2 with the base CVAE model, all words were represented as 300-dimensional embedding vectors. Reconstruct exactly the embedding of the target word is a difficult task for a Neural Network: for this reason, we assigned as predicted word, the one that has embedding vector closer to the predicted one. Closer in this context means the word for which, the norm of the difference between its embedding and generated value is the smaller among all possible word embedding contained in the used embedding dictionary. In formula it is:

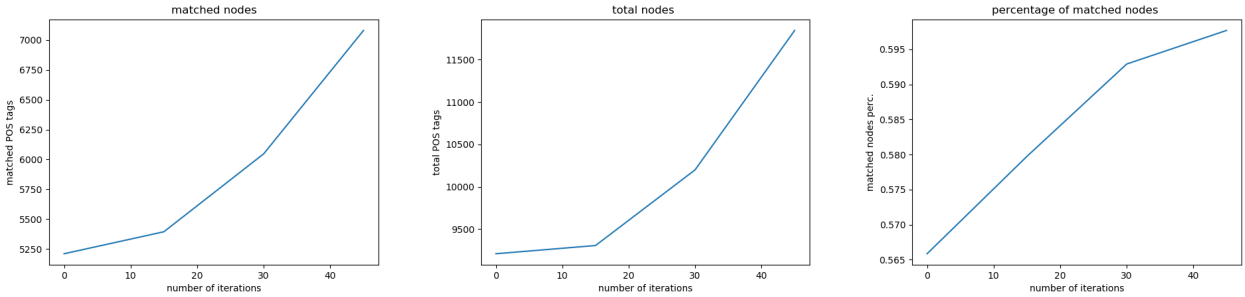
$$arg\ min_{v \in V} |v - g| \tag{4.3}$$

where V is the set of word embedding vectors in the used dictionary and g is the generated one.

With this setting we report the results, again using the metrics defined in Section 4.1, computed just on word nodes in Figure 4.4a, 4.4b, 4.4c that are respectively the number of matched word nodes, the number of total word nodes and the percentage of matched nodes out of total.

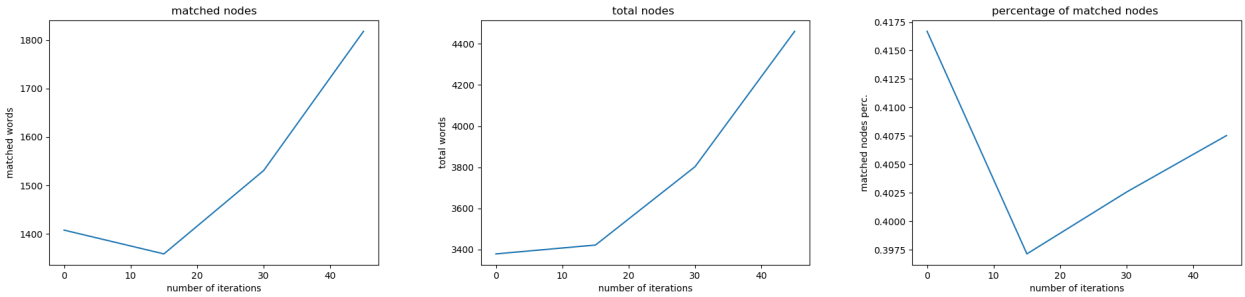
We also took into consideration the BLEU score with the sentence obtained: sentences are made up of single word nodes predicted as leaves of the generated trees. Results for BLEU-1 up to BLEU-4 are reported in table 4.2 as *CVAE*, in which we report the BLEU scores for all the best models obtained using this dataset. These scores clearly show up that the captions qualities is very low: indeed as we grow n, where n is the number of $n-$grams took into consideration in the BLEU score computation, this score dramatically decreases. For instance, the number of $4-$grams matched is 0.8% of the ones in the reference targets.

Finally, we report in figure 4.5 the structure accuracy for this base model.



(a) (b) (c)

Figure 4.3: Accuracy results of POS tag nodes with base model: (a) number of matched nodes , (b) number of total nodes, (c) percentage of matched nodes out of total



(a) (b) (c)

Figure 4.4: Accuracy results of word nodes with CVAE base model: (a) number of matched nodes , (b) number of total nodes, (c) percentage of matched nodes out of total

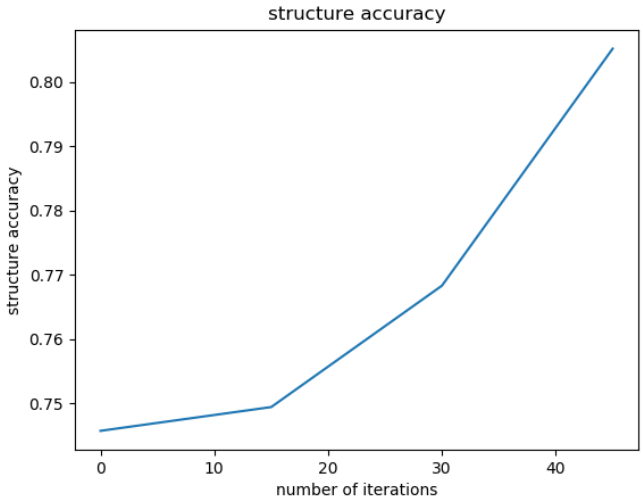


Figure 4.5: Structure accuracy for CVAE

model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	emb	λ	β	Drop
CVAE	0.3705	0.1142	0.0386	0.0087	200	0.05	0.005	0.0
CVAE with NIC	0.3805	0.1579	0.0674	0.0318	225	0.05	0.005	0.0

Table 4.2: Blue scores form Blue-1 up to Blue-4 for different models trained on Flickr8k.

As explained in Section 3.2 we extended the basic CVAE with two different word modules aimed to predict word nodes value in a more standardized way. Another set of experiments we ran is by using the modified version of CVAE using the first word module, which is NIC words module. In particular the encoded representation of the tree was fed to the decoder as before, but now we take care that all the POS tag values are generated before any leaf values computation. This modification was made because, as discussed in Section 3.2 we use POS tags values of parents of words node as inputs for a just introduced RNN, aimed to predict values of these nodes: this means that we need to have all POS node values available when generating words.

The way in which this modified version of CVAE generate results is schematically explained in Figure 4.6

Along with this change in the order in which the different nodes are generated, the new components added to the network are:

- an embedding layer responsible for transform words from a categorical representation, i.e. a one-hot vector, in a dense one.
- a RNN in particular LSTM, that takes as inputs POS tag values of the parent of the word nodes in the generated tree, concatenated with the previous timestamp target words. Feeding previous time stamp targets values to a RNN, at least for the first training iterations, is the common way to train them: this technique is called *Teacher forcing* [60].
- a Dense layer: transforming LSTM output in word prediction scores, its output dimension is equal to vocabulary size in order to associate to each possible word a score.
- a softmax layer: normalizing output of the previous layer into probability distribution among the possible words to generate, i.e. it scales its input vectors into a normalized namely ones having a sum of the composing elements of 1.0.

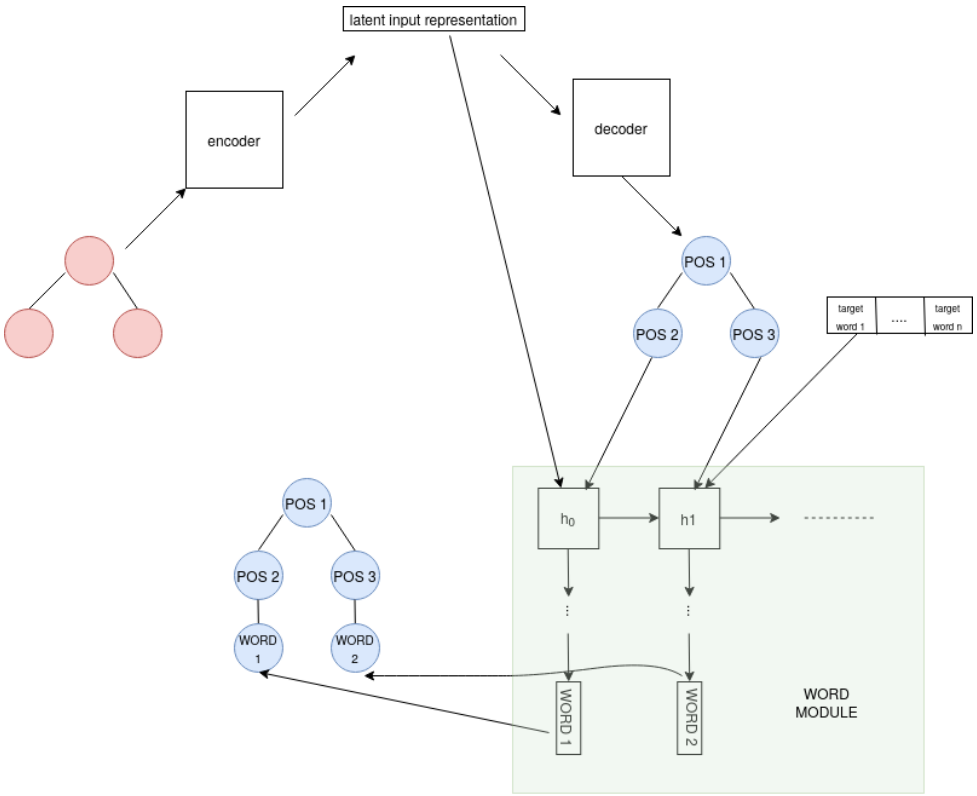


Figure 4.6: Schematic representation of CVAE modified with the addition of a word module. The input tree is transformed through the encoder in a fixed size vector that is the latent input representation as in the standard version of CVAE. From this latent input representation, the standard CVAE decoder part aims at generating POS tags nodes that make up the upper part of the generated tree. In the grey area is shown the word module. A RNN takes as input POS tag embedding of the father of node to generate in the final tree, along with previous word target (apart for the first timestamp in which the latent input representation of the image tree is fed to the RNN). In this way word nodes, that complete the generated parse tree, are computed.

We also follow the classical way to pre-process words value mainly tokenization of words and including only words that have more than 5 occurrences in training set. All the words with less than 5 occurrences are replaced with a special token $< unk >$. This words, together with the $< unk >$ token, make up the dictionary of possible generated words.

We do not use an end of sequence token, *EOS token*, because since we have a tree, the number of words to generate is known a priori (it is equal to the number of leaves in the tree) thus we do not introduce it.

With this setting, we run other experiments evaluating just captions qualities using the blue score. Performance for the best model are summarized in Table 4.2 as

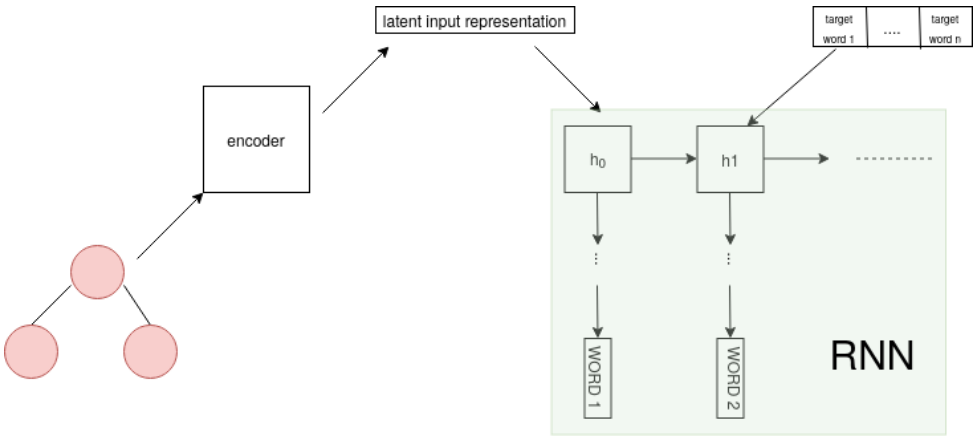


Figure 4.7: Schematic representation of CVAE encoder and sequential decoder. Input tree are mapped in a latent representation vector thanks to encoder part. Next this latent representation vector is fed, along with the word targets to the RNN that produce a sequence of words.

CVAE with NIC.

The last model we test for this dataset was one in which we have CVAE encoder and a sequential decoder, in particular, NIC module as proposed in [58] and summarized in Section 3.2. A schematic illustration of the process carry out by this model, is shown in Figure 4.7.

With this model, we observe no gain with respect to the standard CVAE with NIC module so we do not report results for this model in Table 4.2.

Among the results exposed in this section, tree reconstruction accuracy and POS tag value accuracy were promising but from word nodes, we got very poor results even with the modified version of CVAE, in which the introduction of a word module was made to improve word nodes generation.

With the introduction of a word module, we were able to get slightly better results for BLEU-2, BLEU-3, BLEU-4, probably due to the introduction of a RNN, that generate single words based on the previous ones, but these results are still unsatisfactory.

Moreover, for the last two architectures tested, in which there is a RNN involved in word nodes generation, we also tried to introduce *Curriculum learning* [9] in the process of training: this technique consists in a gradually scheduled shift between teacher forcing in which, as said before, we fed to RNN previously timestamp target values and sampling in which to RNN is fed the $(i-1)$-th word predict at i-th timestamp input.

category	number of total data	train set data	validation set data	test set data
airplane	1055	845	105	105
broccoli	978	784	97	97
cake	950	760	95	95
cat	1024	820	102	102
cow	1030	824	103	103
frisbee	1032	826	103	103
kite	1038	832	103	103
sandwich	956	766	95	95
snowboard	962	770	96	96
teddy bear	954	764	95	95

Table 4.3: Images categories selected and number of example for each one. All the images were took from msCoco dataset. For each of this categories 80% of the images was assigned to train set, 10% to validation set and the remaining 10% on test set. Final results are a train set with roughly 8000 images, validation set with roughly 1000 and test set with roughly other 1000 images for each image category.

This technique is well known as having a good impact on the RNN results: however, in our experiments, we observed no gain with it.

For that reason with the following dataset, we mainly focus on caption qualities ignoring other aspects such as POS tag values and tree topologies assessment because we have already reached enough satisfactory results for them.

4.3 MsCoco

MsCoco is a large dataset containing more than 100000 images in the 2014 release that is the one used in this work. Using such a large dataset would be unfeasible for the purpose of this thesis because training a model with a so large dataset requires too much time. Anyway an important feature of this dataset was used: each image is given along with labels defining the object categories contained in each image. To run some preliminary experiments with this dataset we selected 10 categories among the 91 available. In alphabetical order, the selected ones were airplane, broccoli, cake, cat, cow, frisbee, kite, sandwich, snowboard and teddy bear.

For each of these 10 popular categories, we removed the images in which two or more of these categories were contained to have each image labeled with just one of these 10 categories.

We pre-process inputs and targets as defined in Section 3.3, excluding the images for which we get just one or two segments, as we did for Flickr8k dataset, and we select among the raiment images roughly 1000 images for each category. In table 4.3 were detailed reported how this second dataset is composed.

With this dataset we ran experiments using either AlexNet or Inception v3 to labeled input trees.

4.3.1 Preliminary experiments

With the just described dataset we run some preliminary experiments in order to test the encoder quality. The dataset summarized in table 4.3 were labeled using AlexNet in the experiments described in this section, were the experiments use a tree representation of the input.

CVAE encoder with object detection decoder based on image categories

The architecture we use in this preliminary experiments is still an encoder-decoder architecture with the same encoder described in Section 3.2 but this time decoder is composed of just a layer with an output shape of 10 followed by a softmax in order to predict the object category to which a particular image belongs.

The decoder architecture of this model is shown in Figure 4.8

These experiments were motivated by the necessity to assess the quality of the encoding obtained using this dataset and this encoder: since our final purpose is to achieve good captions, a good visual recognition is a required ability.

We evaluate two different ways of labeling this tree using alexNet: among the layer described in figure 3.2 we have to decide which output of convolutional layer among the 5 present in alexNet is the best one to label trees composing our dataset.

For our choice, we restrict between the 3rd and the 5th convolutional layer, because a previous one is likely to encode features that are at a very low level.

In the experiment just described there is a straightforward way to evaluate performance that is simple accuracy: our best result with dataset labeled with the 3rd convolutional layer achieved 0.62 as accuracy, while the best one labeled with the 5th

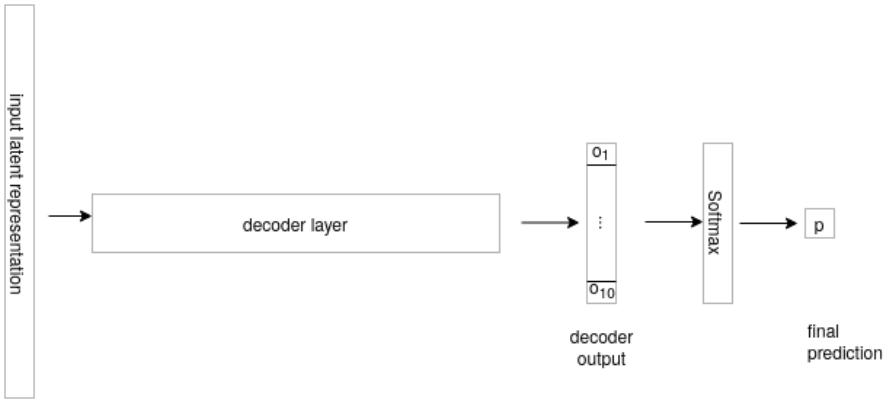


Figure 4.8: Schematic representation of decoder. Input latent representation is fed to decoder that produce a vectors of 10 components. This is then fed to a softmax from which the final prediction is extracted.

one achieved 0.72, which is considerably higher. For this reason, in the following experiments, we use alexNet to label the trees, levering the activation of the 5th layer.

Moreover, these results lead us to think that using the more abstract possible information, which is the one coming from the last layer, is the best way to label the leaves of our trees. This consideration was also used with the other method of tree labeling process, described in Section 3.3: when this process is performed using Inception v3, for which as said in Section 3.3, we select the last layer before the Dense part (aimed to classify each image in one of the 1000 possible categories for ILSVRC competition) to label our tree roots.

The result achieved in this experiment, namely 0.72 of accuracy is much more than a random classifier that among 10 categories would have achieved a ~ 0.1 accuracy: thus we can be reasonably sure that our model input, built with the procedure described in section 3.3 is informative about the image contents.

Moreover we use data visualization tool as *T-sne* [39] to project how data distribute among the embedding space in a 2D dimension. In Figure 4.9a there is a plot of the training set data point and in 4.9b a plot with the validation set.

CVAE encoder with object detection decoder based on image captions

Another preliminary experiment was run using trees labeled with the output of 5th convolutional layer of alexNet: we use the words contained in the related captions of each image to select other categories.

(a) (b)

Figure 4.9: T-sne [39] application on latent representations of tree images, in the 10 categories task. In (a) train set points and in (b) validation set points. Each image colour correspond to one of the categories listed in table 4.3

We could consider this experiment as a sort of linking task between the one previously described and Image captioning task. Indeed its difficulties is in between this two problems: it is still a classification task but it requires recognition of all relevant objects in the image, differently form the previous described one, but the generation of a well formed sentence is not required.

More in detail from the more frequent words, we select only the ones that are noun, excluding the object categories the image belongs or some of its synonymous. With this setting we were able to select other 17 categories (we prefer to drop the nouns that match all the requirements previously listed but that are very infrequent) to use as labels for our images, reaching a total of 27 labels.

The architecture of this decoder is shown in Figure 4.10

Two considerations on this experiment needs to be stress: probably these labels contain noises because the presence of one of this words in the captions, does not every time mean that in the corresponding image is present an object of that category.

The second consideration is that in the previous experiment reported about object recognition, the labels assigned are mutually exclusive meaning that one and only one of the labels can be active for each image. Here instead we have a multi-label problem because even if the originally ten categories, i.e. the one listed in table 4.3 are mutually

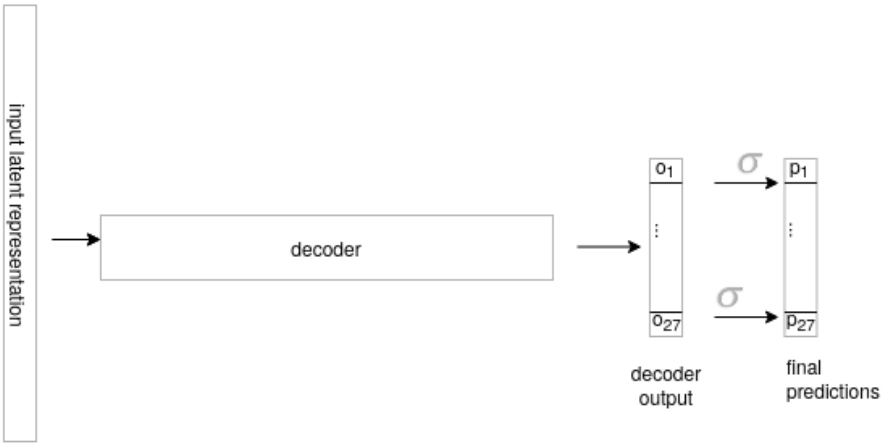


Figure 4.10: Schematic representation of decoder. Input latent representation is fed to decoder that produces a vector of 27 components. Each of these components is then fed to a sigmoid function that aims to shrink that value in range the $\in [0, 1]$. With this architecture, the prediction is no more just a scalar value but it is a vector composed of 27 independent value: the higher it is and the more probable that this feature is active.

exclusive among each other, this is not true for the remaining 17: possible one image can have all these 17 categories active. This is reflected in the decoder architecture: following a Dense layer with an output shape of 27 there are *sigmoid* functions instead of a softmax, to squeeze each of this 27 output in the range $[0, 1]$ that could be easily interpreted as probabilities for each category to be active.

Finally from this output, we select the top-k with a greater output. k is a variable value for each image, defined as $k = min(5, n\_el\_tr)$ where with n_el_tr we denoting the number of labels that are active for a particular image. This choice is due to the fact that we do not simply measure the accuracy in this task but also *precision*, *recall* and *f1-measure*.

Denoting with TP the true positive predicted, with FP false positive predicted and with FN the false negative, precision is defined as $\frac{TP}{TP+FP}$, recall as $\frac{TP}{TP+FN}$, and f1-score, aiming of summarizing the last two metrics is defined as $2 * \frac{recall*precision}{recall+precision}$.

The reason why we need to limit K is because of in precision we need to count the false positives: having a k value greater then the number of true labels would have been unfair.

In Table 4.4 we report the results for our best model: precision is still more or less the same as before with a vale of 0.73, recall is 0.46. Finally, f1-score has a value of 0.56.

accuracy	precision	recall	f1-score
0.73	0.72	0.46	0.56

Table 4.4: Accuracy, precision, recall and f1-score for object recognition task.

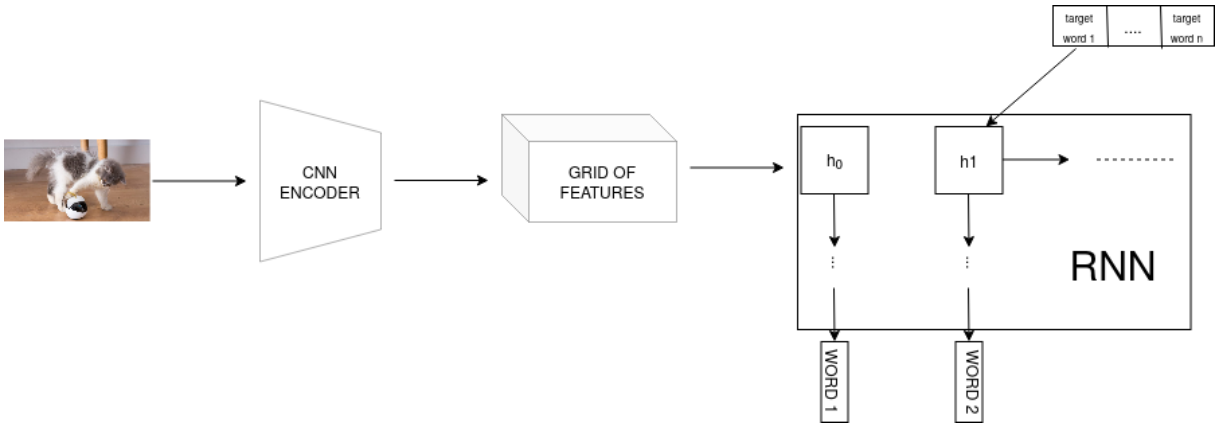


Figure 4.11: Schematic representation of flat to sequence model. Images are fed to CNN that produce a grid of features. Starting from that representation RNN produce a sequence which is the generated sentence.

Flat encoder and sequential decoder

Another quick preliminary experiment was to run the image captioning task, with a typical neural network architecture used for image captioning task, composed by a flat encoder and a sequential decoder: we schematic report this architecture in Figure 4.11. Data used is our newly built dataset starting from msCoco. For this model, we use as encoder Inception v3 CNN and as decoder the second word module described in 3.2, the ones we called Attention module. An important caveat is that because of the sequential decoder and the way in which the image representation is fed to the RNN, namely in each timestamp, for this set of experiments we need to introduce a *end of sequence token* and a *start of sequence* tokens.

These experiments were necessary due to, differently from Flickr8k, we don't have in scientific literature any baseline or result on this dataset, since it is built from scratch for the purpose of this work. A baseline is needed to compare the results we get from experiments, concerning tree representations, in order to access their qualities.

The results achieved from the best model were reported in table 4.6 as *imq2seq*

Differently from the experiments reported before in section 4.2.1 this experiment was run using just teacher forcing thus without using curriculum learning, moreover we tested just a few hyper-parameters. Even with this simple setting, we were able to

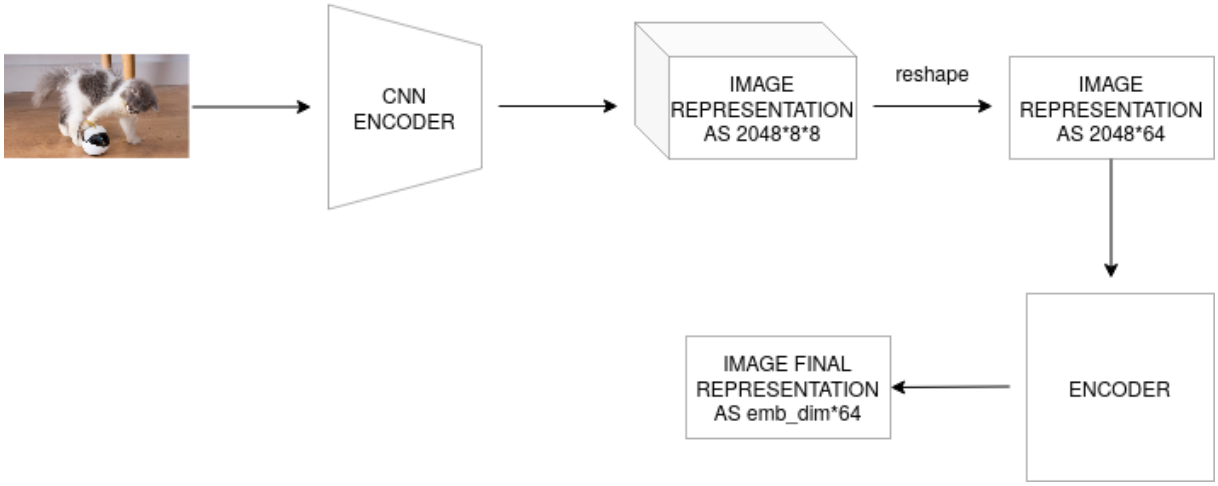


Figure 4.12: Schematic representation of CNN encoder. Image, preprocessed as described by Section 3.1.3 are fed to Inception v3 CNN that give as result a tensor of shape $2048 \times 8 \times 8$. After a reshape aiming of merging the last two dimensions, the encoder shrink its input that is a 2048×64 matrix in to a $emb\_dim \times 64$ matrix, with emb_dim dimension of embedding chosen.

overcome the results achieved with our previous experiments.

4.3.2 Experiment results

In these experiments we tested: a flat encoder and the CVAE decoder. Specifically, we use Inception v3 CNN to process images, in particular, form the layer shown in figure 3.3 we take as image representation the output of the $2048 \times 8 \times 8$ layer. Images fed as input are pre-processed according to what described in section 3.1.3 that is what expected by this CNN.

These $2048 \times 8 \times 8$ tensors are the input for that model. The encoder is a layer reducing the first dimension i.e. 2048, in the embedding dimension chosen. After that there is a reshape in order to merge the last two dimensions, resulting in a $emb\_dim \times 64$ tensor. The decoder used is the same as described in section 3.2 using both word modules that are NIC and attention word modules.

In Figure 4.12 is shown a detailed architecture of the encoder.

The results for this models were reported in table 4.6 as *img2tree NIC* and as *img2tree attention*

We consider these results, obtained without curriculum learning, as satisfactory because we were able to overcome the image to sequence model results found in the

model type	matched leaf in BCPT	total leaf BCPT	ratio
Img2Tree with NIC words module	696	2436	0.2857
Img2Tree with Attention words module	738	2398	0.3077
Tree2Tree with words module	380	1735	0.2190
Tree2Tree with Attention words module	337	1885	0.1787

Table 4.5: Different BCPT scores for tree decoder models. As encoder we have both Inception v3 CNN, reported as img2tree and CVAE reported as Tree2tree

preliminary experiment. For this reason, we also prefer to report in table 4.5 the BCPT scores described in section 4.1 for these best models. This Table show how even if the BLEU score is good, using the metrics defined on BCPT, at least for the word nodes, they give poorer results.

This is probably due to the fact, we are computing BLEU score against 5 reference captions, as done by any other image captioning works, while in BCPT score we are comparing the generated tree just with a single reference tree. We need to take in mind that these metrics was designed for classification tasks in which what is needed is just to the number of correctly generated items, but in our case, we are in a generative setting. For this reason, in all this chapter we haven't focused too much attention on these metrics, because they are not descriptive of the qualities of generated captions.

We also run a second experiments in which we have tree transduction with this dataset, using for the first time Inception v3 CNN to label our tree dataset. This experiment was done due to discovering how the used CNN to label trees and the strategy used in this process can affect the qualities of the result: indeed we have just tested a tree to tree transduction with Fickr8k dataset, but using input trees labeled with AlexNet.

We report the results obtained about blue score in table 4.6 as *tree2tree NIC* and as *tree2tree attention* and about BCPT metrics we report the results in 4.5 with the same names.

These results compared with the *img2seq* are worse especially for the BLEU-2, but for BLEU-1 the model using NIC word module achieves the best result found so far. Differently, the results obtained with just tree decoder and flat encoder, were able to overcome the flat model in all BLEU scores.

For the result concerning the BCPT module, the previously expressed considera-

model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	emb	λ	β	Drop
img2seq	0.4391	0.2436	0.1207	0.0592	300	0.0	0.0	0.5
img2tree NIC	0.5044	0.3064	0.166	0.0903	300	0.0	0.005	0.0
img2tree attention	0.4854	0.2667	0.1336	0.0686	300	0.001	0.007	0.0
CVAE NIC	0.5332	0.187	0.1041	0.068	300	0.0005	0.01	0.0
CVAE attention	0.456	0.1878	0.0784	0.0396	300	0.002	0.01	0.0

Table 4.6: Different BLEU scores for different model trained on our dataset built form msCoco. img2seq stands for flat encoder and flat decoder while img2tree stand for flat encoder and CVAE decoder

tion, namely they are not suitable for our task, is confirmed also from the result of this task because even in this case the results are lower than the BLUE ones.

Before jump to the conclusion we inform the reader of the presence of examples taken by our best models, namely the ones reported in this section. We report, in the Appendices, a good and a bad caption example for both the models described in this section: we first report examples for the CVAE model, thus an example of tree transduction learnt, then we report examples for the model with a Flat encoder and tree decoder.

Chapter 5

Conclusions and further works

In this thesis, we dealt with image captioning tasks performed with a structured approach. As explained in Section 4 we tested two different settings, one in which only targets sentences are represented as trees, specifically as parse trees, and one in which both image inputs and target sentences are represented as trees.

We also implemented and empirically validated different neural architectures built on the top of a *Conditional Variational Autoencoders for Tree-Structured data* with the addition of different components to generate sentences in a more suitable way, different ways to label our input tree with visual information coming from two different CNNs. Finally, we tested the models in two different datasets that are Flickr8k and another one ad hoc built for the purpose of this thesis starting from msCoco.

In the spirit of contributing to the research community, all the code is publicly available on git hub.

Concerning results with the dataset built from scratch, we run some preliminary experiments to find a baseline result, with a model using Flat Encoder and Sequential decoder, thus representing both inputs and targets as unstructured data. This result was necessary so that we could used it as a comparison baseline for other experiments.

Comparing to that baseline, the experiments in which we use a flat image representation and trees as targets have reached better results in all the BLEU scores. This set of experiments is a strong indication that structured data can help the network in learning how to generate a suitable target. Even if further experiments need to be run, probably the key factors in these improvements are POS Tag node values as additional inputs to RNN and the good results achieved in the tree topology accuracy. Indeed is well known that generated sentences are usually longer than reference ones, but in

our settings, the tree topology implicitly defines the number of words in the generated caption.

The experiments in which we have both input and target represented as trees were able to overcome the baseline results for BLEU-1, achieving the best result we found in our experiments, but they are worse especially for BLUE-2 score. However, even in this case, the obtained results are not so far from that baseline: a reasonable mapping between input and targets, that in this case is a tree to tree transduction, was learnt.

In this last setting we have also to consider the segmentation process: as said in Section 3.3 it was performed using as hyper-parameters the standard one used in GLIA. The only hyper-parameters we choose for this step was Watershed level of water and σ in the Gaussian blurring of the boundary probability map. For these two hyper-parameters, the selected values were the two that make them more robust to horizontal reflection transformation of just tens of sample images. It is very likely that for this step we have not selected the best possible hyper-parameters resulting in segmentation that are not perfect: with a deeper search for optimal hyper-parameters even for this step, it is very likely that the results can be improved.

Moreover, with these results, we are able to demonstrate that it is possible to build trees that represent inputs and targets. While for targets, there are well-known data structures as parse trees and dependency trees (indeed for this step we use the *Stanford parser* software), for input images we had to build from scratch our strategy. A relevant percentage of total work was spent in designing these input trees in all their aspects, for instance how to represent hierarchical segmentation and how to label them.

Even if all these experiments were run without the usage of curriculum learning for time necessity, we think the caparison between the baseline with the flat encoder/sequential decoder and our proposed data representation is a reasonable indication of quantitative comparison among structured versus unstructured data. Indeed all these experiments were run with just Teacher Forcing and there is no reason to think that our proposed models can not improve using curriculum learning.

The results just commented were the ones we have judged as positive, which are the ones in which we use Inception v3 CNN to label trees coming from our ad hoc built dataset. The other ones, that we have considered as unsatisfactory at least for the qualities of captions generated, were conducted using Flickr8k and AlexNet to label

trees. We can not be sure about which changes played a role in the results qualities. Indeed, in addition to the different dataset and the different CNN used to label input trees, there is also a difference in the way this process is performed: with Inception v3 CNN we need to include global information in the root of trees, while for AlexNet each internal node, included the root, is labeled with the max-pooling of its children, thus no global information are inserted. With the experiments we have conducted, we can not tell which of these modifications play a significant role: for this step, further work is required.

Another important conclusion concerns the metric used to evaluate the qualities of generated trees in the original *Conditional Variational Autoencoder for tree-structured data work*. As said in Section 4 they seem to be unsuitable to evaluate our results. This is probably due to the different nature of the work: in the original *Conditional Variational Autoencoder for tree-structured data* work there were only experiments with classification tasks, while image captioning is a generative task. For this reason, also a good evaluation of the POS tag is required further work. A greedy way in which this analysis could be conducted is by comparing the average tree depth at which a specific tag occurs both in the target trees and in the generated ones.

There is a last consideration to make regarding the word module used which seems not to play a big role, indeed our best results were found using the NIC module which is less sophisticated than the competitor.

Appendices

In this Appendices we like to report visual examples of good and bad results for both the model described in Section 4.3.2 which are the best models we get from this thesis. We do this to complete the quantitative description of the results summarized in Table 4.6 with also qualitative results. In the following we do not report the input tree because of the difficulties in its interpretation (we remind to the reader that input tree nodes are labeled with vectors of hundreds of components, each of this is a Floating point value) : we replace it with the original image and the segmentation computed by GLIA in the pre-processing step. On the other way round, for the target we choose to report the generated tree using capital letters for the POS tag and lower case letter for the words.

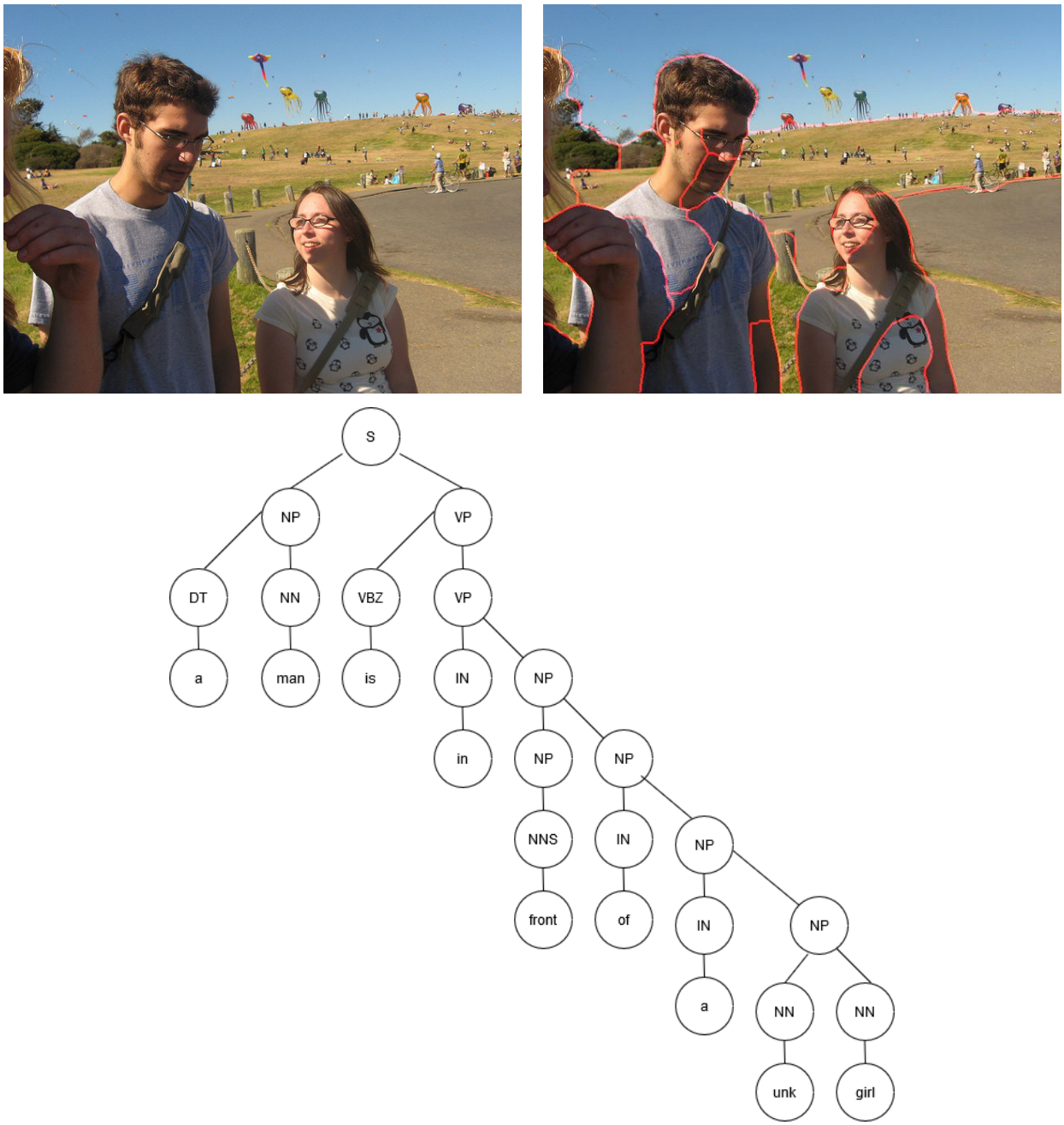


Figure 1: Example of good captioning result for CVAE model. In the first row (a) the original image and (b) the segmentation computed in the pre-processing step. We remind that the input for the CVAE is neither (a) nor (b) but the tree obtained as described in Section 3.3. We choose to replace it with (a) and (b). For the target, in the second row, we report the full parse tree generated for which uppercase nodes are the POS tags and lowercase nodes are the words nodes. unk is a special token used when the network chose to generates a word, but among the words in the used dictionary the network consider none of this as appropriated.

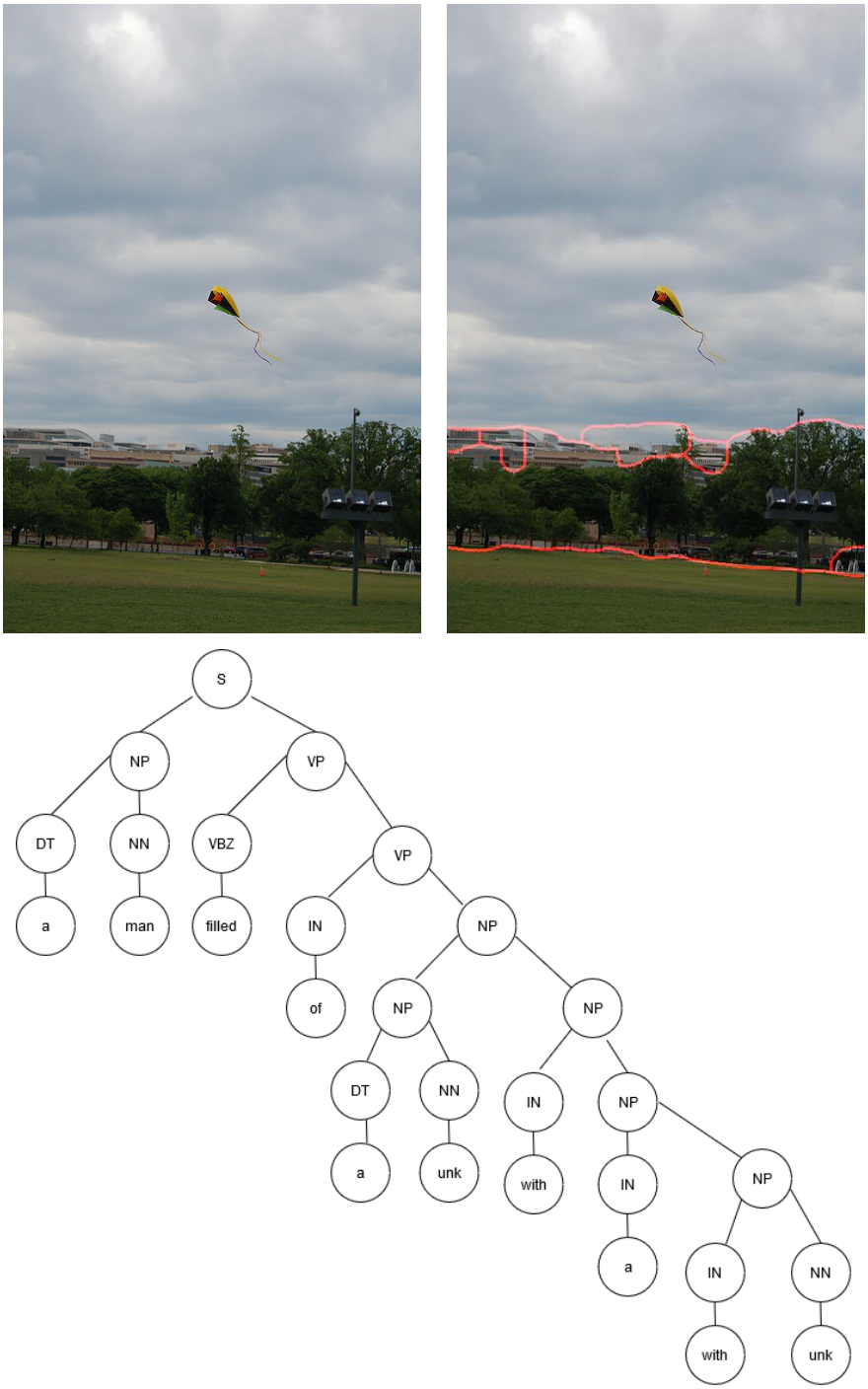


Figure 2: Example of bad captioning result for CVAE model. In the first row (a) the original image and (b) the segmentation computed in the pre-processing step. We remind that the input for the CVAE is neither (a) nor (b) but the tree obtained as described in Section 3.3. We choose to replace it with (a) and (b). For target, in the second row, we report the full parse tree generated for which uppercase nodes are the POS tags and lowercase nodes are the words nodes. unk is a special token used when the network chose to generates a word, but among the words in the used dictionary the network consider none of this as appropriated.

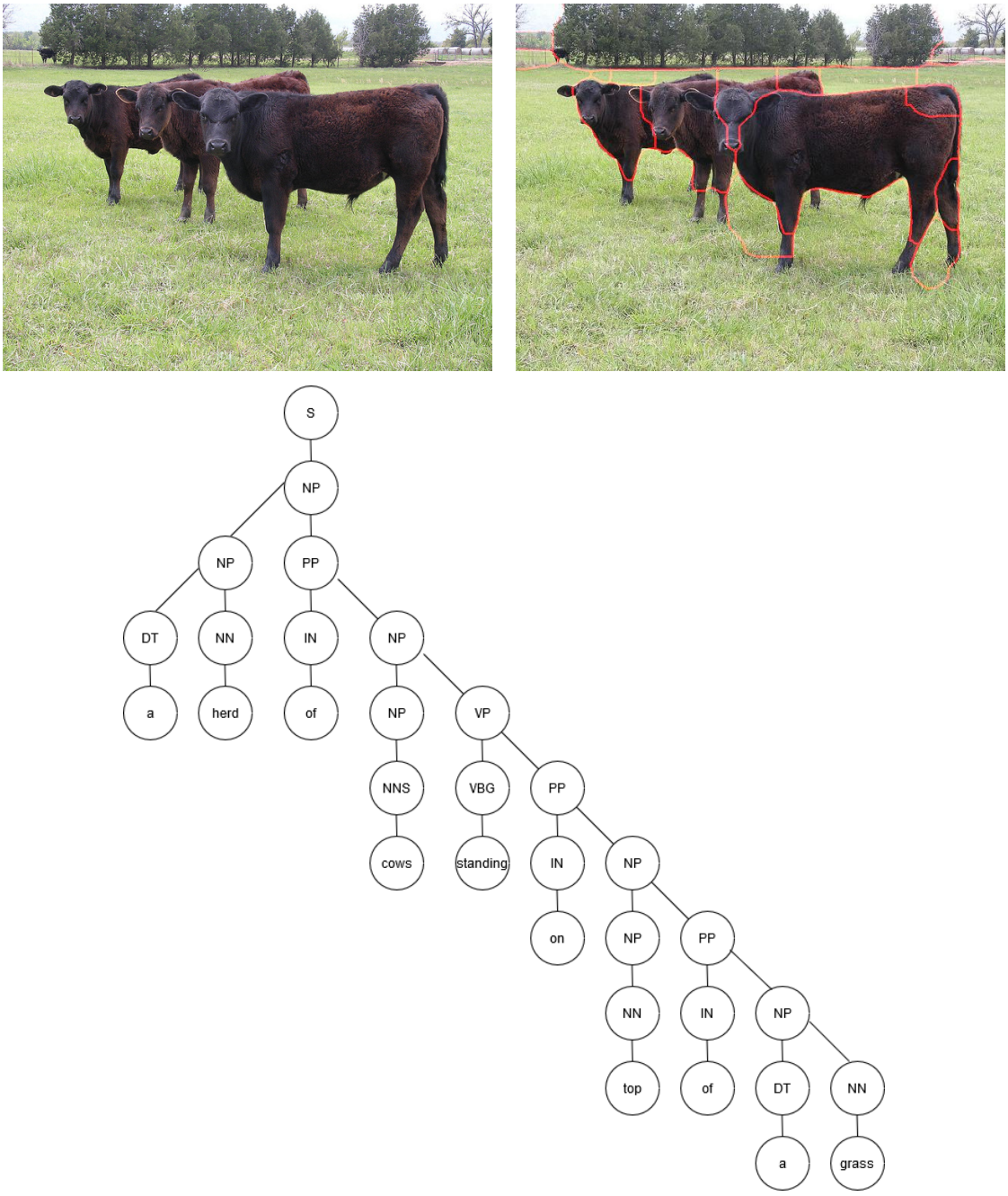


Figure 3: Example of good captioning result for flat encoder/tree decoder model. In the first row (a) the original image and (b) the segmentation computed in the pre-processing step. For target, in the second row, we report the full parse tree generated for which uppercase nodes are the POS tags and lowercase nodes are the words nodes. unk is a special token used when the network chose to generates a word, but among the words in the used dictionary the network consider none of this as appropriated.

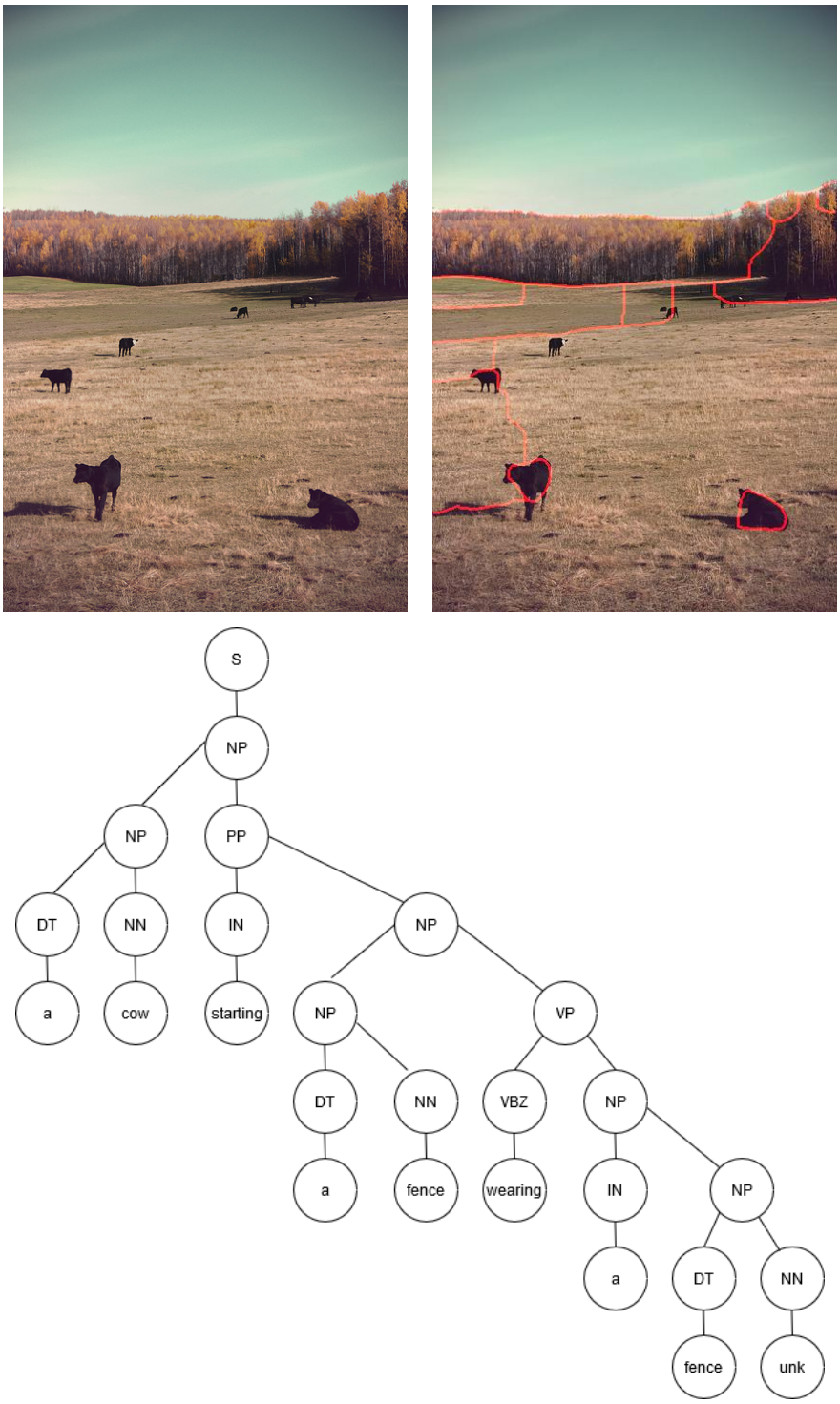


Figure 4: Example of bad captioning result for flat encoder/tree decoder model. In the first row (a) the original image and (b) the segmentation computed in the pre-processing step. For target, in the second row, we report the full parse tree generated for which uppercase nodes are the POS tags and lowercase nodes are the words nodes. unk is a special token used when the network chose to generates a word, but among the words in the used dictionary the network consider none of this as appropriated.

Bibliography

[1] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European Conference on Computer Vision*, pages 382–398. Springer, 2016.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2010.

[3] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[4] Davide Bacciu and Antonio Bruno. Text summarization as tree transduction by top-down treelstm. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1411–1418. IEEE, 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[7] Richard Barnes, Clarence Lehman, and David Mulla. Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models. *Computers & Geosciences*, 62:117–127, 2014.

[8] Sadia Basar, Awais Adnan, Naila Habib Khan, and Shahab Haider. Color image segmentation using k-means classification on rgb histogram. *Recent Advances In Telecommunications, Informatics And Educational Technologies*, pages 257–262, 2014.

[9] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[10] Serge Beucher. Use of watersheds in contour detection. In *Proceedings of the International Workshop on Image Processing*. CCETT, 1979.

[11] Somporn Chuai-Aree, Chidchanok Lursinsap, P Sophasathit, and Suchada Siripant. Fuzzy c-mean: A statistical feature classification of text and image segmentation method. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(06):661–671, 2001.

[12] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] Trevor Anthony Cohn and Mirella Lapata. Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, 34:637–674, 2009.

[14] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):603–619, 2002.

[15] Michel Couprie, Laurent Najman, and Gilles Bertrand. Quasi-linear algorithms for the topological watershed. *Journal of Mathematical Imaging and Vision*, 22(2-3):231–249, 2005.

[16] Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. Watershed cuts: Minimum spanning forests and the drop of water principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1362–1374, 2008.

[17] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005.

[18] Meenakshi M Devikar and Mahesh Kumar Jha. Segmentation of images using histogram based fcm clustering algorithm and spatial probability. *International Journal of Advances in Engineering & Technology*, 6(1):225–231, 2013.

[19] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.

[20] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[21] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[22] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *European conference on computer vision*, pages 15–29. Springer, 2010.

[23] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996.

[24] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[25] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, 2014.

[30] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[35] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[36] Ting Liu. Glia: Graph learning library for image analysis. `https://github.com/tingliu/glia`, 2016.

[37] Ting Liu, Mojtaba Seyedhosseini, and Tolga Tasdizen. Image segmentation using hierarchical merge tree. *IEEE transactions on image processing*, 25(10):4596–4607, 2016.

[38] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[39] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[40] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *arXiv preprint arXiv:1412.6632*, 2014.

[41] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. Explain images with multimodal recurrent neural networks. *arXiv preprint arXiv:1410.1090*, 2014.

[42] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 1–8, 2014.

[43] Marina Meilă. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, pages 577–584. ACM, 2005.

[44] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[45] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[46] Gustavo H Paetzold and Lucia Specia. Text simplification as tree transduction. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, 2013.

[47] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[48] P Daniel Ratna Raju and G Neelima. Image segmentation by using histogram thresholding. *International Journal of Computer Science Engineering and Technology*, 2(1):776–779, 2012.

[49] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211. Association for Computational Linguistics, 2012.

[52] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[53] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc., 2015.

[54] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[55] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[56] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[57] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.

[58] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. corr abs/1411.4555 (2014). *arXiv preprint arXiv:1411.4555*, 2014.

[59] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[60] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[61] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[62] Xingxing Zhang, Liang Lu, and Mirella Lapata. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060*, 2015.