

TECHNICAL UNIVERSITY OF CRETE

School of Electrical and Computer Engineering



Diploma Thesis

EXPLORING EFFICIENCY AND PERFORMANCE OF IMAGE CAPTIONING

Nadia - Eythymia Frechat

Thesis Committee:

Associate Professor Michail G. Lagoudakis, Thesis Supervisor

Professor Michael Zervakis

Professor Dionisios Pnevmatikatos (National Technical University of Athens)

A thesis submitted in partial fulfillment of the requirements for the degree of
Diploma in Electrical and Computer Engineering

Chania October 2019

Abstract

Image captioning is a complex problem that combines the fields of computer vision and natural language processing. It generates natural language sentences that describe the content of an image. Image captioning has several applications in the real world with significant practical impact, from assisting users with visual impairments to personal assistants through human-robot interaction.

The progress in image captioning is a major success of Artificial Intelligence. It has been reported that under some metrics, such as BLUE or CIDEr, the most up-to-date techniques even outperform human performance.

In this thesis, we implement and present a model based on machine learning techniques that combines the latest developments in computer vision and machine translation that can be used to create natural sentences that describe an image. Specifically, a combination of Convolutional Neural Networks together with Recurrent Neural Networks was used to obtain the desired results. The models were trained to maximize the likelihood of a target description given the training image.

Experiments on a huge set of training data, such as the MSCOCO 2015 used in this thesis, demonstrate the accuracy of the model and the fluency of the language that is acquired through the image descriptions alone. It has been tested qualitatively and quantitatively that the model is often quite accurate.

Περίληψη

Ο υποτιτλισμός εικόνας είναι ένα πολύπλοκο πρόβλημα που συνδυάζει τον τομέα της μηχανικής όρασης και της επεξεργασίας φυσικής γλώσσας. Στοχεύει στην παραγωγή προτάσεων σε φυσική γλώσσα που περιγράφουν το περιεχόμενο κάποιας εικόνας. Ο υποτιτλισμός εικόνας έχει αρκετές εφαρμογές στον πραγματικό κόσμο με σημαντικό πρακτικό αντίκτυπο, από την παροχή βοήθειας σε χρήστες με προβλήματα όρασης έως προσωπικούς βοηθούς μέσω της αλληλεπίδρασης ανθρώπου-ρομπότ.

Η πρόοδος στον υποτιτλισμό εικόνας είναι μια σημαντική επιτυχία της Τεχνητής Νοημοσύνης. Έχει αναφερθεί ότι υπο ορισμένες μετρικές, όπως το BLUE ή το CIDEr, οι πιο σύγχρονες τεχνικές ξεπερνούν ακόμα και τις ανθρώπινες επιδόσεις.

Σε αυτή τη διπλωματική εργασία, υλοποιούμε και παρουσιάζουμε ένα μοντέλο βασισμένο σε τεχνικές μηχανικής μάθησης που συνδυάζει τις πιο σύγχρονες εξελίξεις στην μηχανική όραση και τη μηχανική μετάφραση και που μπορεί να χρησιμοποιηθεί για τη δημιουργία φυσικών προτάσεων που περιγράφουν μια εικόνα. Συγκεκριμένα, χρησιμοποιήθηκε ένας συνδυασμός συνελκτικών νευρωνικών δικτύων μαζί με ανατροφοδούμενα νευρωνικά δίκτυα για την απόκτηση των επιθυμητών αποτελεσμάτων. Τα μοντέλα εκπαιδεύτηκαν έτσι ώστε να μεγιστοποιούν την πιθανότητα περιγραφής στόχου δεδομένης της εικόνας εκπαίδευσης.

Πειράματα σε ένα πολύ μεγάλο σύνολο δεδομένων εκπαίδευσης, όπως το MSCOCO που χρησιμοποιήθηκε σε αυτή τη διπλωματική, δείχνουν την ακρίβεια του μοντέλου και την ευχέρεια που αποκτά η γλώσσα αποκλειστικά μέσα από τις περιγραφές των εικόνων. Το μοντέλο, το οποίο ελέγχθηκε ποιοτικά και ποσοτικά, είναι συχνά αρκετά ακριβές.

Acknowledgements

At this point, I would like to thank my supervisor, Mr. Michail Lagoudakis and the rest of my committee Mr. Dionisios Pnevmatikatos and Mr. Michael Zervakis for the opportunity they gave me to prepare my diploma thesis, and the confidence they showed from the beginning at my work and myself. Then, I have to thank Mr. Antonis Nikitakis in particular for his constant guidance and for his immediate response for everything needed while writing this thesis.

Table of contents

Abstract	ii
Περίληψη.....	iii
Acknowledgements	iv
Table of contents	v
Table of Figures	vii
Chapter 1: Introduction.....	9
1.1 Introduction	9
1.2 Thesis Contribution	9
1.3 Introduction to Neural Networks	9
1.4 Main Characteristics of Neural Networks	11
1.4.1 Advantages of Neural Networks	11
1.4.2 Disadvantages of Neural Networks	11
1.5 Learning Process	12
1.5.1 Supervised Learning.....	12
1.5.2 Unsupervised Learning.....	12
1.5.3 Reinforcement Learning	12
Chapter 2: Neural Networks	14
2.1 Neuron.....	14
2.2 Activation Functions	15
2.2.1 Sigmoid or Logistic	15
2.2.2 Tanh(Hyperbolic tangent)	16
2.2.3 ReLu (Rectified Linear Units)	16
2.2.4 Softmax.....	17
2.3 Networks architecture.....	17
2.3.1 Single layer Feed-Forward Networks	17
2.3.2 Multi layer Feed-Forward Networks	18
2.3.3 Recurrent Networks.....	19
2.4 Network Training.....	20
2.4.1 Cost Function	20
2.4.2 Gradient Descent	20
2.4.2 Error BackPropagation	21
Chapter 3: Other Types of Neural Networks.....	25
3.1 Recurrent neural networks (RNN)	25
3.1.1 Architecture	25
3.1.2 Backpropagation Through Time	26
3.1.3 Vanishing/Exploding Gradients	27
3.1.4 Long Short-Term Memory (LSTM).....	27
3.2 Convolutional Neural Networks (CNN).....	29
3.2.1 Architecture	29
3.2.2 Convolutional Layer	30
3.2.3 Training	34

Chapter 4: Related Work	35
4.1 Object Detection	35
4.1.1 Mask RCNN	35
4.1.2 Image Classification	36
4.1.3 Inception_V3.....	37
4.2 Image Captioning.....	40
Chapter 5: Our Image Captioning Model.....	41
5.1 Overview of Model Architecture	41
5.2 Encoder.....	42
5.2.1 Mask R-CNN and Inception_V3 in our System.....	42
5.2.2 Image Embedding	44
5.3 Decoder	44
5.3.1 Word Embedding	44
5.3.2 Attention	46
5.3.3 Caption Generator Based on LSTM	49
Chapter 6: Model Training.....	51
6.1 Database MSCOCO 2015.....	51
6.2 Model Training	53
6.2.1 Training Variables.....	53
6.2.2 Training Algorithm.....	54
6.2.3 Hyperparameter Selection.....	55
6.3 Metrics.....	56
6.3.1 BLEU.....	56
6.3.2 ROUGE	57
6.3.3 METEOR.....	57
6.3.4 CIDEr	57
Chapter 7: Results.....	59
7.1.1 Qualitative Examples and Discussion	61
Chapter 8: Epilogue	74
8.1 Summary and Conclusions	74
8.2 Future Work.....	74
References	76

Table of Figures

FIGURE 1-2 : STRUCTURE OF A REAL NEURAL NETWORK IN COMPARISON WITH AN ARTIFICIAL NEURAL NETWORK.[51]..	10
FIGURE 2-1 : ARTIFICIAL NEURON MODEL [52]	14
FIGURE 2-2 : SIGMOID FUNCTION AND ITS DERIVATIVE.[30]	15
FIGURE 2-3 : HYPERBOLIC TANGENT FUNCTION. [31]	16
FIGURE 2-4 : RECTIFIED LINEAR UNITS FUNCTION.[32].....	16
FIGURE 2-5 : STRUCTURE OF SINGLE LAYER NEURAL NETWORK. [33].....	17
FIGURE 2-6 : STRUCTURE OF A MULTILAYERED NEURAL NETWORK. [34].....	18
FIGURE 2-7 : DIFFERENCE BETWEEN AN RNN (LEFT) AND A FEED FORWARD NETWORK (RIGHT). [35].....	19
FIGURE 2-8 : VISUALIZATION OF GRADIENT DESCENT. [36]	21
FIGURE 3-1 : SCHEMATIC REPRESENTATION OF AN RNN. [38]	25
FIGURE 3-2 : VARIOUS FORMS THAT AN RNN NETWORK MAY HAVE. [39].....	26
FIGURE 3-3 : RESULTS OF APPLYING THE SIGMOID FUNCTION MULTIPLE TIMES.[40].....	27
FIGURE 3-4 : ARCHITECTURE OF AN LSTM. [41]	28
FIGURE 3-5 : CELL STRUCTURE OF AN LSTM CELL. [41].....	29
FIGURE 3-6 : FORM OF A COLOR IMAGE. [42]	30
FIGURE 3-7 : A REPRESENTATION OF CONVOLUTIONAL LAYER. [1]	30
FIGURE 3-8 : PROCESSING AN IMAGE USING DIFFERENT FILTERS. [1]	31
FIGURE 3-9 : EXAMPLE OF A FILTER WITH STRIDE = 1. [1]	31
FIGURE 3-10 : APPLICATION OF ZERO PADDING AT THE BORDERS OF AN IMAGE. [1].....	32
FIGURE 3-11 : APPLICATION OF A 2x2 MAX-POOLING. [1]	33
FIGURE 3-12 : EXAMPLE OF A FC LAYER AFTER A CONVOLUTIONAL LAYER. [1].....	33
FIGURE 4-1 : MASK R-CNN STRUCTURE. [43]	35
FIGURE 4-2 : FEATURE PYRAMID NETWORKS. [44]	36
FIGURE 4-4 : (A) ORIGINAL INCEPTION MODULE (B) 5×5 CONVOLUTION IS REPLACED BY TWO 3×3 CONVOLUTION. [2]	38
FIGURE 4-5 : (A) FACTORIZATION OF THE $N \times N$ CONVOLUTIONS. (B) ONE 3×1 CONVOLUTION FOLLOWED BY ONE 1×3 CONVOLUTION REPLACES ONE 3×3 CONVOLUTION. [2]	38
FIGURE 4-6 :(A) INCEPTION MODULES. (B) AUXILIARY CLASSIFIER [2]	39
FIGURE 4-8 : INCEPTION MODULE FOR GRID-SIZE REDUCTION. [2]	39
FIGURE 4-9 : SCHEMATIC DIAGRAM OF INCEPTION V3. [46].....	40
FIGURE 5-1 : IMAGE CAPTIONING EXAMPLES. [47]	41
FIGURE 5-2 : A HIGH-LEVEL BLOCK DIAGRAM OF OUR MODEL.	42
FIGURE 5-3 : ENCODER MODEL ARCHITECTURE	42
FIGURE 5-4 : A VISUAL EXAMPLE OF HOW WE USE THE MASK-RCNN MODEL.....	43
FIGURE 5-5 : DECODER MODEL ARCHITECTURE	44
FIGURE 5-6 : AN EXAMPLE OF BAG OF WORDS MODEL. [49]	45
FIGURE 5-7 : SEMANTIC SIMILARITIES USING WORD EMBEDDING. [48]	46
FIGURE 5-8 : ATTENTION MODEL STRUCTURE. [50]	47
FIGURE 5-9 : A TYPICAL ATTENTION MODEL WITH A UNIFORM GRID OF EQUALLY-SIZED IMAGE REGIONS (LEFT). OUR APPROACH ENABLES ATTENTION TO BE CALCULATED AT THE LEVEL OF OBJECTS (RIGHT). [14]	48
FIGURE 5-10 : ATTENTION EXAMPLE	49
FIGURE 5-11 : AN EXAMPLE OF HOW LSTM IS USED FOR GENERATING CAPTIONS	50
FIGURE 7-1 : EXAMPLES OF HOW OUR MODEL FAILS TO GET THE NUMBER OF OBJECTS IN AN IMAGE.	62
FIGURE 7-2 : EXAMPLES OF HOW THE SMALL VARIETY OF WORDS PRODUCED BY OUR MODEL AFFECTS THE CAPTIONS	63
FIGURE 7-3 : AN EXAMPLE OF OUR MODEL BEING BIASED.	64

Chapter 1: Introduction

1.1 INTRODUCTION

Image captioning is a major problem of artificial intelligence, which combines the field of Computer Vision with Natural Language Processing. The ability of a system to be able to automatically generate descriptions of the content of the images by producing correct, syntax and semantically, proposals seem to be a fairly demanding challenge which, however, could have a great effect, such as helping people with impaired vision to better understand the content of images that are available either online or in the real world. This problem is much more difficult than the problem of image classification or of problems that are related to the detection of objects in images, which mainly deals with computer vision problems. This is due to the fact that we not only need to identify the individual objects in an image, but must also express how these objects are interconnected, by determining which features and activities are related. In addition, the above semantic information must be expressed in a natural language such as English, which means that one needs a model of natural language in addition to visual comprehension of the image.

1.2 THESIS CONTRIBUTION

In this thesis we combine deep convolutional networks for image feature extraction and recurrent networks (RNN) responsible for modeling the proposals so that we can create a unified network that will produce descriptions of images. This thesis utilizes some of the state of the art approaches in the fields of object detection and image classification, as well as, attention mechanism that was created for machine translation and used it in order to give more focus on the important parts of the input image, thus producing more accurate results.

Chapter 1 analyzes how neural networks work and how biological networks have been the inspiration behind them, as well as what are some of the main benefits that make them produce best results in a wide variety of problems. Chapter 2 emphasizes in the algorithm used for their training and how it changes the parameters of the network in way that minimizes the cost function. Some of the most well know networks are Recurrent Neural Networks and Convolution Neural Networks, which are described in detail in this thesis and are explained Chapter 3. Then, in Chapter 5, we discuss the image captioning model created for the purpose of this thesis in detail and how we used each part of the model to achieve our goal. COCO dataset is, then ,explained in detain in Chapter 6 and the way it was used to train our model. In Chapter 7, we discuss the metrics we used to evaluate the accuracy of our system, we calculate the result and present some examples of the successful captions. Apart from the accurate descriptions it is important to view some unsuccessful example and figure out what caused this unsuccessful behavior. Finally in Chapter 8, we end this thesis by suggesting some modification that may improve even more the accuracy of the system.

1.3 INTRODUCTION TO NEURAL NETWORKS

Neural nets are a relatively new area in the natural sciences, since they have become known and developed only in the last forty years. Their main feature is that their principles and functions are based on the nervous system of living organisms, but their study and use has gone far beyond biological organisms, and today neural networks are used to solve any kind of problem associated with computers.

Their philosophy, however, is different from the way in which classical computers work. Their function tries to combine the thinking of the human brain with the abstract mathematical way of thinking. Thus, in neural networks we use ideas such as learning, memory etc. things we have so far attributed only to human thought.

The brain consists predominantly of a wide range of neurons (approximately 10,000,000,000), which are mass interconnected, with an average of 1,000 interconnections per neuron. The central building block of the brain is neurons, nerve cells that create a dense communication network between them. Anatomically the neuron consists of the body, the dendrites and the axon. In each dendrite there is an infinitely short void called a synapse. In particular, **dendrites** are the neural entry gates as they receive electrical signals from other neurons. The **axon** is the gate of the neuron. Sends signals to other neurons in the form of electric pulses of fixed amplitude but of variable frequency. The **synapses** are the points in which the branches of the axon of a neuron are joined to the dendrites of other neurons. The percentage of electrical activity finally transmitted to the dendrite is the synaptic weight.

The synapses are divided into excitatory and inhibitory depending on whether the charge released by the synapse irritates the neuron to produce pulses more frequently or suppresses it by preventing it from producing pulses. The most important feature of the neuron is its sensitivity, its ability to react to various external stimuli. This reaction results in the generation of short-duration pulses. The pulses travel on the axis of each neuron and through the synapses propagate to the dendrites of other neurons. Each neuron collects all the electrical charge it receives from each synapse in dendrites. If the sum of the load exceeds a threshold then the neuron's axis begins to generate electrical pulses at a high frequency so we say the neuron shoots. If the load does not pass this limit then the neuron produces very little pulses at random moments, so we say that the neuron is inactive. All empirical knowledge thus acquired by the neural network from the environment is coded in synaptic weights. These are the characteristic that gives the network the ability to evolve and adapt to the environment.

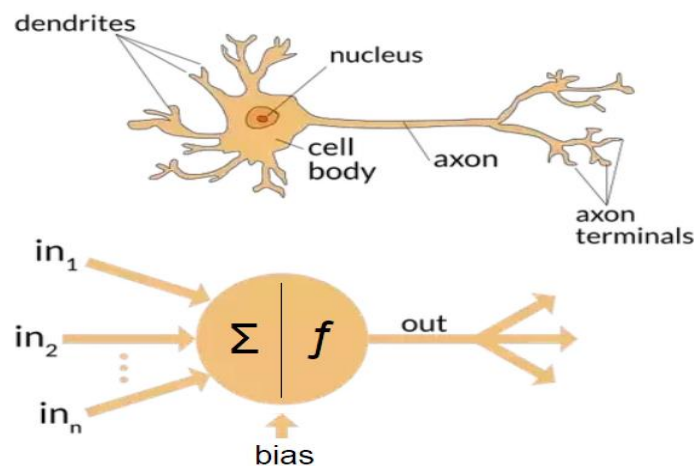


Figure 1-1 : Structure of a real neural network in comparison with an artificial neural network.[51]

In analogy with a network of brain neurons, an artificial neural network consists of a set of artificial neurons that are linked and interact with each other by the so-called synapses. The degree of interaction is different for each pair of neurons and is determined by the so-called synaptic weights. As the neural network interacts with the environment and learns from it, synaptic weights are constantly changing, potentiating or weakening the strength of each bond.

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

1.4 MAIN CHARACTERISTICS OF NEURAL NETWORKS

A neural network owes its computational power firstly to its parallel, distributed structure and secondly to its ability to learn and generalize. The term generalization refers to the production of reasonable outputs for inputs that the neural network has not met during the training. These two features enable neural networks to find good approximate solutions to complex problems.

1.4.1 Advantages of Neural Networks

Neural networks provide the following useful properties:

1. **Non-Linearity:** An artificial neuron can be either linear or nonlinear. A neural network consisting of interconnected non-linear neurons is by nature non-linear. This non-linearity is distributed across the network and is an extremely important property, especially if the physical mechanism for producing the input signals is non-linear.
2. **Adaptability:** Neural Networks have the ability to adapt their weights to changes in their environment. Sometimes adjustments lead to a reduction in system performance.
3. **Indicative Response:** A Neural Network is designed to provide information not only on the specific example chosen but also on the confidence in the decision taken.
4. **Content related information:** Knowledge is represented by the highly structured and active state of the Neural Network. Each neuron in the network may be affected by the overall activity of all other network neurons. This means that a neural network manipulates the content-related information in a natural way.
5. **Tolerance to failures:** Neural Networks have great tolerance for structural errors. This means that the malfunctioning or destruction of a neuron or some connections is not capable of significantly disrupting their operation as the information that it encloses is not localized at a specific point but diffused throughout the network.
6. **Implementability in VLSI:** the massively parallel nature of the Neural Network, makes it possible to implement VLSI technology so that neural networks can be used in very large-scale integration applications.
7. **Analysis and Design Uniformity:** The concept is that the same symbolism is used in all fields that contain neural network implementation, which is indicated in several ways: Neurons define a component common to all neural networks. This property makes it possible to share learning theories and algorithms in different applications of neural networks.
8. **Analogue with Neurobiology:** Neural Network Design is done in analogy with the brain. Neurobiologists see neural networks as the subject of research to explain neurobiological phenomena. Similarly, engineers see neurobiology for new ideas to solve complex problems.

1.4.2 Disadvantages of Neural Networks

Below are listed the disadvantages of Neural Networks and their use:

1. Hardware dependence: Artificial neural networks require processors with parallel computing power. For this reason, the implementation of the equipment depends.
2. Determination the network structure: There is no specific rule on how to build a neural network. Appropriate networking is achieved through experience, testing, trial and error.
3. Unexplained network behavior: This is the most important problem of NN. When the NN produces a solution, there is no explanation of why and how. This reduces the trust of the network.
4. Explanation and translation of weights in Neural Nets are impossible because of their non-linearity.
5. Difficulty in displaying the problem in the network: Neural networks can only process numeric information. Problems must be translated into numerical values before they are entered in the SW. The imaging mechanism to be determined here directly affects the performance of the network.

1.5 LEARNING PROCESS

As there are different ways people learn from their environment, the same goes for neural networks. In a broad sense, we can categorize the learning processes through which neural networks work as follows: supervised learning, unsupervised learning and reinforcing learning.

1.5.1 Supervised Learning

Learning with an instructor is also referred to as supervised learning. The neural network receives pairs of input-desired output vectors and generates with the current weight state an output that initially differs from the desired output. This difference is then calculated and the weights are adjusted based on that error. More specifically, the teacher has knowledge of the environment and this knowledge is represented by a set of input-output examples. However, the environment is unknown in the neural network. If the network is exposed to an environmental training resource, because of his knowledge, the trainer may provide the neural network with the desired response for that particular vector. The desired response represents the optimal action to be performed by the neural network. The network parameters are influenced by the training vector in combination with the error signal. The error signal is defined as the difference between the desired response and the actual network response. This process is repeated in order to bring the network into a situation where it will simulate the teacher's situation.

1.5.2 Unsupervised Learning

In unsupervised learning there is no external trainer to oversee the learning process. Instead, there is a process-independent measure of the representation quality that is required to learn the network and the network parameters are optimized in relation to it. Training samples are now only input samples and do not contain samples of the desired output. Training stops when the network stops changing the weight values.

1.5.3 Reinforcement Learning

In reinforcing learning, learning an output-matching input is performed through continuous interactions with the environment, aiming at minimizing a scalar performance measurement. In this type of learning the network is again fed with sample inputs but is not fed with the desired responses to these outputs. Now we use a general measure of the adequacy of the resulting judgment that can lead the network to the desired behavior. This measure is known as a reinforcement signal and is being fed back to the network in order to reward good behavior and punish the wrong ones. In

summary, reinforcement learning works as follows: Initially, the neural network calculates the outputs produced by the current input with current weight values. The system then evaluates the output and the reinforcement signal is fed into the network. The weights are adjusted based on the reinforcement signal, increasing the weight values that contributed to good behavior or reducing the weight values that caused bad behavior.

Chapter 2: Neural Networks

2.1 NEURON

A neuron is an information processing unit that is fundamental for the operation of a neural network and is the basis for designing a larger network. An artificial neuron is a computational model whose parts are directly matched with those of the biological neuron. Each neuron receives information, processes it and gives an output value. Its inputs are either the outputs of other neurons, or the input signal of the network.

A neuron usually receives many simultaneous inputs. Every input has its corresponding weight. These weights have the same type of operation as synaptic forces of biological neurons. The weights are adaptive coefficients within the network that determine its intensity of the input signal as they enter the artificial neuron. The weight shows us just how important the contribution of this signal is to the configuration of the network structure for the two connecting neurons. Accordingly to how big or small the weight is, the contribution of the signal is great or small respectively. These forces can be modified according to the different training standards and according to the specific network topology or through the training rules.

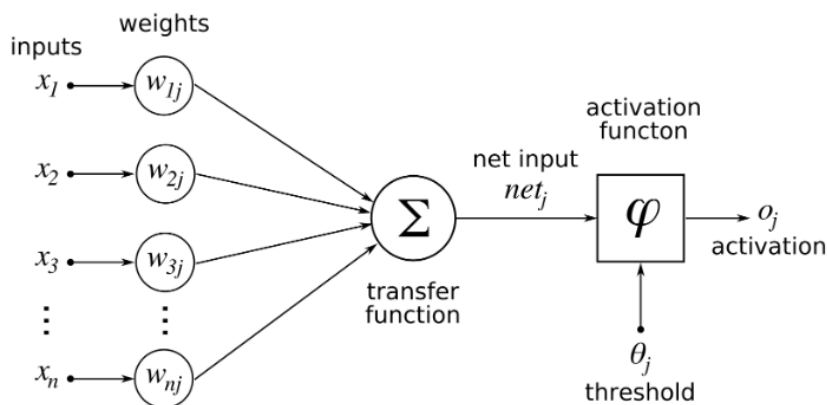


Figure 2-1 : Artificial Neuron model [52]

The basic parts of which a neuron is composed (shown in Figure 2-1) are:

1. A set of synapses each of which is characterized by its own weight. More specifically, a signal x_k at the input of synapse j associated with the neuron j is multiplied by the synaptic weight w_{kj} . The synaptic weight of an artificial neuron can receive both positive and negative values.
2. An adder for summing the input signals weighted by the corresponding synaptic weights.
3. An activation function for limiting the amplitude of the output signal of the neuron.
4. The neuron model also includes an externally applied bias b_j . Bias results in an increase or decrease in network excitation of the activation function, depending on whether it is positive or negative respectively. Bias is an external parameter of the neuron and is equal to the synaptic weight w_{j0} of the fixed input $x_0 = +1$

In mathematical terms we can describe the neuron with the following pair of equations:

$$U_j = \sum_{k=0}^n w_{kj} x_k \quad (2.1)$$

$$y_j = \varphi(U_j) \quad (2.2)$$

Where x_1, x_2, \dots, x_i are the input signals and $w_{j1}, w_{jk}, \dots, w_{ji}$ are the respective synaptic weights of the neuron j , b_j is the bias, $\varphi()$ the activation function and y_j the output signal of neuron j .

2.2 ACTIVATION FUNCTIONS

They are used to convert the input signal of a neuron to a non-linear output signal. If an activation function is not applied then the output signal would be a simple linear function which, although easy to solve, is unable to learn complex mappings from data such as images, video, audio, speech, etc. A neural network should be able to learn and compute not only linear functions but also more complex non-linear functions. Non-linear functions are those that have a degree greater than one and their curve is non-linear.

Another important feature of the activation function is that it should be differentiable so that backpropagation can be applied to optimize the algorithm as we propagate back into the network to calculate the error in relation to weights and gradually optimize the weights using Gradient Descend. Therefore, we conclude that in order for the network to be powerful, to have the ability to learn complicated processes and to be able to process complex data, an activation function must be applied.

Four are the most common functions :

1. Sigmoid or Logist
2. Tanh — Hyperbolic tangent
3. ReLu (Rectified linear units)
4. Softmax

2.2.1 Sigmoid or Logistic

Sigmoid function is the most common activation function. It is defined as a strictly increasing function, receives values from a continuous range of values from 0 to 1 and is differentiable. The sigmoid function is defined as:

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

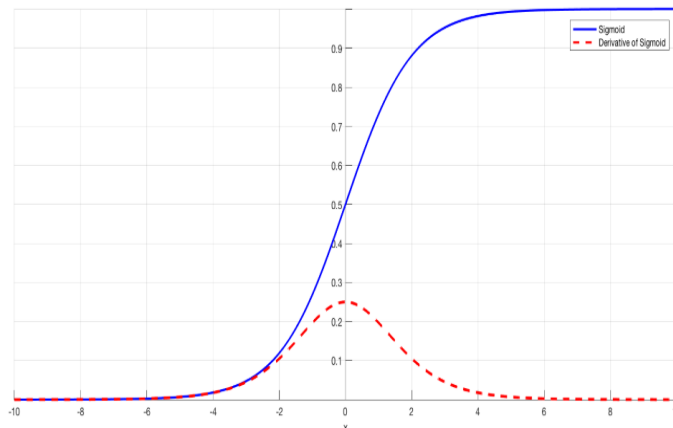


Figure 2-2 : Sigmoid function and its derivative.[30]

2.2.2 Tanh(Hyperbolic tangent)

This function is also a sigmoidal function in terms of its shape, it's difference is in the fact that it gets values in range $(-1,1)$. The fact that the output values are zero – centered make it preferable to the sigmoidal function. The tanh function is differentiable and is given:

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

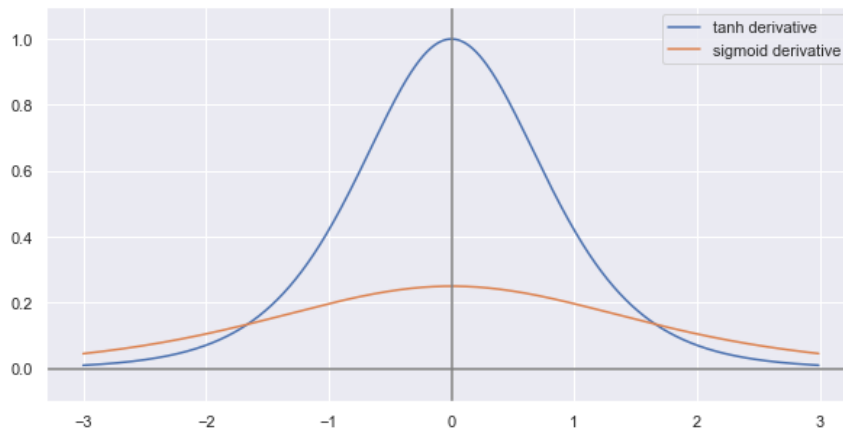


Figure 2-3 : Hyperbolic tangent function. [31]

2.2.3 ReLu (Rectified Linear Units)

The rectifier non-linearity is defined as follows:

$$R(x) = \max(0, x) \quad (2.5)$$

where x is the input to a neuron.

This function ensures that the output will not get a value below zero.

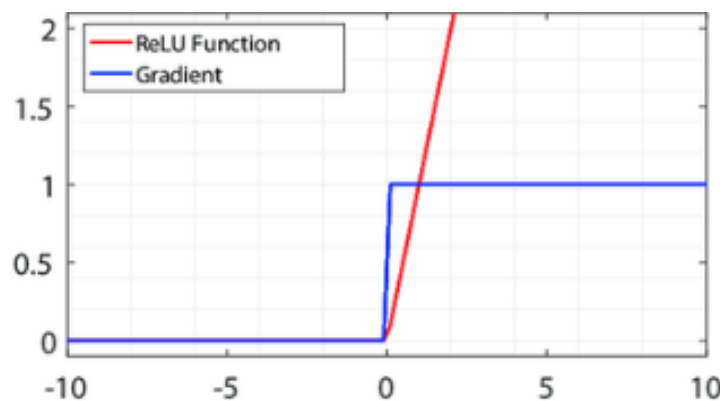


Figure 2-4 : Rectified Linear Units function.[32]

2.2.4 Softmax

The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0, 1] and it is used most cases in the last layer of the networks. The softmax function used for multi-classification model returns the probabilities of each class and the target class will have the high probability.

$$P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^K e^{x^T w_k}} \quad (2.6)$$

Our objective is to predict if the trained set of features x , each with its own set of weights w_j , are a class of j . The formula 2.6 computes the exponential of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.

2.3 NETWORKS ARCHITECTURE

Neural networks are characterized by their architecture, the function they perform and their training method. The network architecture determines the layout of the connections, the number and the type of neurons. The way in which neurons of a network are structured is closely related to the learning algorithm used to train the network. Generally there are three fundamental architectures:

2.3.1 Single layer Feed-Forward Networks

In a neural network, the neurons are organized in the form of layers. In the simplest form there is an input layer, which is directly connected to a layer of output neurons but not vice versa. Such a network is called **single layer** due to the output level. Input layer is not counted because it does not perform any calculations.

The network of this type is also called **feed forward** network. The feedforward neural network was the first and simplest type of artificial neural network. In this network, the information moves in only one direction, forward, from the input nodes to the output nodes. There are no cycles or loops in the network.

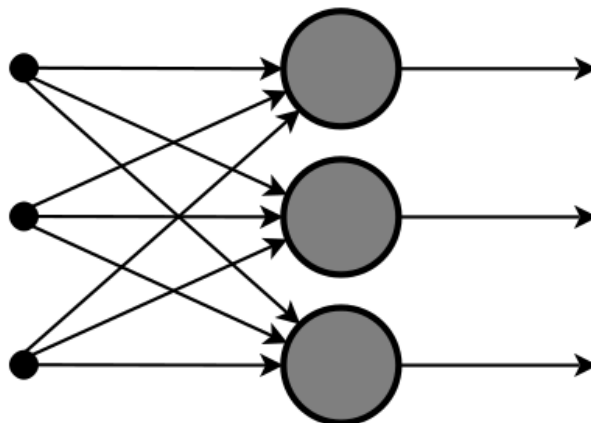


Figure 2-5: Structure of single layer neural network. [33]

The first layer is the d-dimension inputs. In the second layer, the outputs are combined to give the activations of the M output units. The output at this network structure is calculated by the following equations:

$$y_j = f(\sum_{i=0}^d w_{ji} x_i) \quad j=0, 1, \dots, M \quad (2.7)$$

Where, w_{ji} is the weight for input x_i connected to neuron j in the output layer and $f()$ is an activation function like those described in section 2.2.

2.3.2 Multi layer Feed-Forward Networks

This category of neural networks is characterized by the number of hidden layers, whose nodes are called hidden neurons. The operation of these neurons is to interfere between the external input and output of the network by doing some processing.

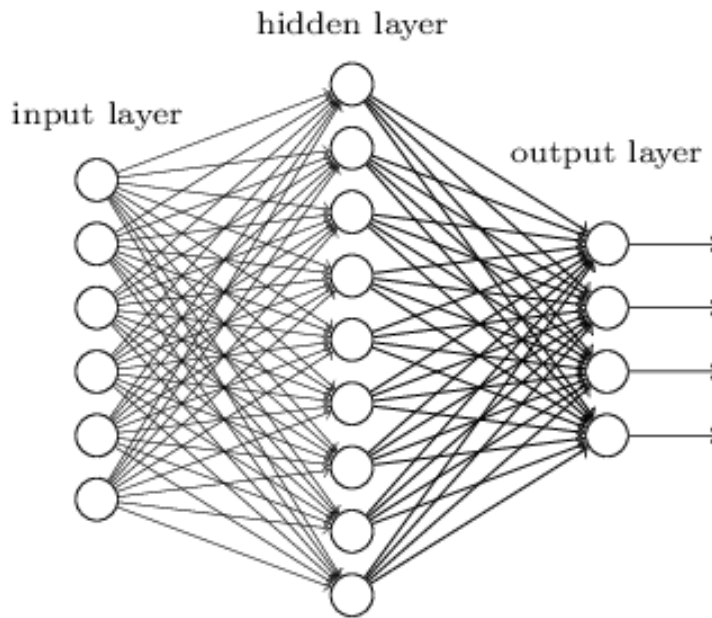


Figure 2-6 : Structure of a multilayered neural network. [34]

The **Input nodes** provide information from the outside world to the network and are together referred to as the “Input Layer”. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes. The **Hidden nodes** have no direct connection with the outside world. They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a “Hidden Layer”. The **Output nodes** are collectively referred to as the “Output Layer” and are responsible for computations and transferring information from the network to the outside world.

The first layer is the d-dimension inputs. In the second layer, the outputs of the hidden units are combined to give the activations of the M output units:

$$h_j = f(\sum_{i=0}^d w_{ji} x_i) \quad j=0, 1, \dots, M \quad (2.8)$$

Then the output of the last layer will be calculated, for K output units, as:

$$y_k = g(\sum_{j=0}^M w_{kj} h_j) \quad k=0, 1, \dots, K \quad (2.9)$$

Or if we combine all the above equations the output of the network is:

$$y_k = g(\sum_{j=0}^M w_{kj} f(\sum_{i=0}^d w_{ji} x_i)) \quad (2.10)$$

2.3.3 Recurrent Networks

A recurrent neural network differs from a feed forward neural network in having at least one feedback loop. The output of each network neuron feeds the input of other neurons of the same layer and, in some cases, even its own input (self-feedback). The presence of feedback loops affects the learning ability of the network and its performance.

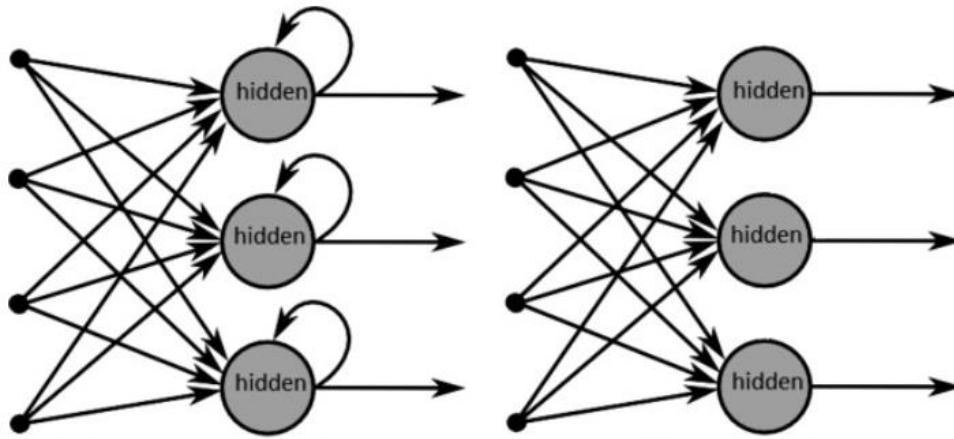


Figure 2-7 : Difference between an RNN (left) and a feed forward network (right). [35]

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1}) \quad (2.11)$$

$$y_t = f(W_{hy}h_t) \quad (2.12)$$

Where, W_{hh} is the array of weights of the hidden state to the hidden state, W_{xh} is the array of weights from the input to the hidden state, W_{hy} is the array of weights from the output to the hidden output and f an activation function applied per element. Recurrent neural networks will be discussed in more detail in Chapter 3.

In all the above cases the networks were **fully connected** because each node of each network layer is connected to each other node of the next level. In the case of missing network connections, then we are talking about a partially connected network.

2.4 NETWORK TRAINING

The main purpose of an artificial neural network is to be able to perform certain processes e.g. to recognize images, but having been first properly trained. Each network receives some inputs and gives some outputs. Learning is done by giving the network as inputs some standards for which we know what the output should be. The network with these data modifies its internal structure to make the same match based on the data given to it. This modification is based on a cost function and a training algorithm which in most cases is gradient descent. The cost functions calculates the deviation from the right response and the training algorithm updates the weights of the models in a way that the cost function is minimized.

After finding the right internal structure, then it can solve other similar problems that it has not seen before. However, these problems should be of the same nature and characteristics as those of training. The training process will be further explained in the next sections.

2.4.1 Cost Function

The cost function of a supervised learning problem computes the compatibility between model prediction and its true value. Depending on the type of problem we need to resolve, we also use a different cost functions. In this case, we have to solve a classification problem, so we chose the **Cross-entropy** cost function. Cross-entropy loss measures the performance of a classification model whose output is a probability between 0 and 1 and expresses how far the prediction is from the actual distribution. The loss of Cross-entropy increases as the predicted probability deviates from the actual label. In case we have a binary classification, where the number of classes M equals 2 Cross-entropy can be calculated from the equation:

$$L = -(y \log(p) + (1 - y) \log(1 - p)) \quad (2.13)$$

Where y is the ground truth and p is the prediction.

If we classify multiple classes, then we calculate separately the error for each class, based on the labels, for one entry, and then we add the results.

$$L = \sum_{c=1}^M y_{o,c} \log(p_{o,c}) \quad (2.14)$$

Where M is the number of different categories, y is a binary marker (0 or 1) on whether the c tag is the correct categorization for observing o and p the probability predicted by the model to be a class c entry.

The total loss for all data takes the form of an average loss from each separate example, ie it takes the form of:

$$J = -\frac{1}{N} \sum_i L_i \quad (2.15)$$

With N the set of training data and L_i the error calculated by the Cross-entropy for the given i .

2.4.2 Gradient Descent

The idea behind the gradient descent [3] is the gradual but repetitive reduction of the error by adjusting the weights. Intuitively, we know that if a change in weight increases (reduce) the error, then we want to reduce (increase) that weight. Mathematically, this is written: $\partial J / \partial w$, which represents the change in error relative to a change of weight. Once we calculate this derivative, we will update the weight through the following equations:

$$w = w - n\nabla J(w) \quad (2.16)$$

Where, n is the learning rate and $\nabla J(w)$ is the gradient of the cost function $J(w)$ with respect to the parameter w .

If we update all weights using equation (2.16) it means that the parameters move in the direction of the steeper descent along the error function - hence the name, gradient descent.

The general idea of Gradient Descent is:

1. Calculate the slope, which is the first derivative of the cost function at the given time.
2. Update parameters by adding the quantity calculated in the previous step towards the negative gradient of the slope.
3. After each weight update the gradient is re-examined for the new weight vector and the procedure is repeated.

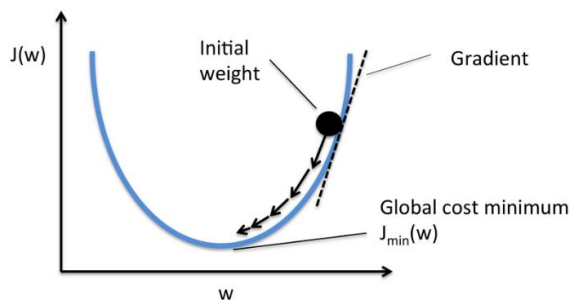


Figure 2-8: Visualization of Gradient Descent. [36]

There are three types of Gradient Descent:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

The pseudocode for stochastic gradient descent (sgd) has the following form:

Algorithm 2.1 : stochastic gradient descent

1. Select initial weight vector w and learning rate n .
2. Repeat until a minimum is reached:
 - a) Randomly mix training the examples
 - b) Calculate the slope: $\partial J / \partial w$
 - c) Calculate the update direction: $\Delta w = -\eta \partial J / \partial w$
 - d) Run a parameter update: $w = w + \Delta w$

2.4.2 Error BackPropagation

Backpropagation algorithms, short for "backward propagation of errors", is used in order to train artificial neural networks following a gradient descent approach. Backpropagation computes the gradients, whereas gradient descent uses the gradients to train the model.

The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient

from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.

The backward propagation process for training a neural network consists of two passes across the different layers of the network: a forward pass and a backward pass. In the forward passage an input vector is applied to the network input neurons and its effect is propagated within the network from one layer to the next and in the direction of the input layer to the output layer. Eventually a set of outputs is generated as the actual network response. Comparing the final output values with the desired result we calculate the error for each of them. During the first pass, the weights of the network remain stable. During the backward pass weights are adjusted according to the error correction rule produced by the forward passage. More specifically, the actual response of the network is subtracted from the desired response to produce an error signal propagating back into the network, on the opposite direction of the connections (direction from the output level to the input level), from which the name 'error back propagation' emerges. The synaptic weights are adjusted to make the actual network response approximate the desired response.

Algorithm 2.2: BackPropagation[\[26\]](#)

1. Initialize weights w , v and learning rate α .
2. Repeat until the number of epochs is reached:
 - I. Repeat for each training pair in the dataset:
 - a) Each input unit receives input signal x_i and sends it to the hidden unit for all $i = 1$ to n
 - b) Calculate the net input at the hidden unit using the following relation:

$$Q_{inj} = b_{0j} + \sum_{i=1}^n (x_i u_{ij}) \quad j=1 \text{ to } p$$

Where b_{0j} is the bias on hidden unit, u_{ij} is the weight on j unit of the hidden layer coming from i unit of the input layer. Now calculate the net output by applying the following activation function

$$Q_j = f(Q_{inj})$$

Send these output signals of the hidden layer units to the output layer units.

- c) Calculate the net input at the output layer unit using the following relation:

$$y_{inj} = b_{0k} + \sum_{j=1}^n (Q_j w_{jk}) \quad k=1 \text{ to } m$$

Where b_{0k} is the bias on output unit, w_{jk} is the weight on k unit of the output layer coming from j unit of the hidden layer. Calculate the net output by applying the following activation function:

$$y_k = f(y_{inj})$$

- d) Compute the error correcting term, in correspondence with the target pattern received at each output unit, as follows:

$$\delta_k = (t_k - y_k) f'(y_{inj})$$

On this basis, update the weight and bias as follows:

$$\Delta u_{jk} = \alpha \delta_k Q_{inj}$$

$$\Delta b_{0k} = \alpha \delta_k$$

Then, send δ_k back to the hidden layer.

- e) Now each hidden unit will be the sum of its delta inputs from the output units.

$$\delta_{inj} = b_{0j} + \sum_{k=1}^m (\delta_k w_{jk})$$

Error term can be calculated as follows:

$$\delta_j = \delta_{inj} f'(Q_{inj})$$

On this basis, update the weight and bias as follows:

$$\Delta w_{ij} = \alpha \delta_j x_i$$

$$\Delta b_{0j} = \alpha \delta_j$$

f) Each output unit y_k , with $k = 1$ to m , updates the weight and bias as follows:

$$u_{jk}(new) = u_{jk}(old) + \Delta u_{jk}(new)$$

$$b_{0k}(new) = b_{0k}(old) + \Delta b_{0k}(new)$$

g) Each output unit z_j , with $j = 1$ to p , updates the weight and bias as follows :

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}(new)$$

$$b_{0j}(new) = b_{0j}(old) + \Delta b_{0j}(new)$$

The steps described above train the neural network, ie all weights and network parameters have been optimized so they can correctly predict an output for a given input. After training if a new input enters the network, it will perform a forward pass from all the individual levels and produce a probability for each unit of the output layer. If the training is done properly then the network will generalize the new data well enough and will produce the desired output even on input data it has never seen.

Chapter 3: Other Types of Neural Networks

Depending on the type of problem and set of parameters required to determine the output, there are different neural networks based on different mathematical functions. Below we will analyze some "special" neural networks that were used to implement our own model.

3.1 RECURRENT NEURAL NETWORKS (RNN)

In a Feed-Forward neural network, the data move only in one direction, from the input layer, through the hidden layers, to the output layer. The information moves directly through the network and for this information passes through every node only once. Feed-forward neural networks have no memory for the input they received before, and therefore cannot predict the data to follow. Thus, it is impossible to use them in cases where the network inputs are sequences. In contrast, in RNNs the information passes through a loop, so that when a decision is made, it will take into account the current input but also what it learned from the inputs it received in the past according to the formula $h_t = f(h_{t-1}, x_t)$, where f is an activation function, h_{t-1} the previous state and x_t the input. Therefore, RNN have two input sources, the current input and the previous hidden state, which are combined to determine how they react to new data.

3.1.1 Architecture

If an RNN [4], [27] unfolds in time it will take the form below:

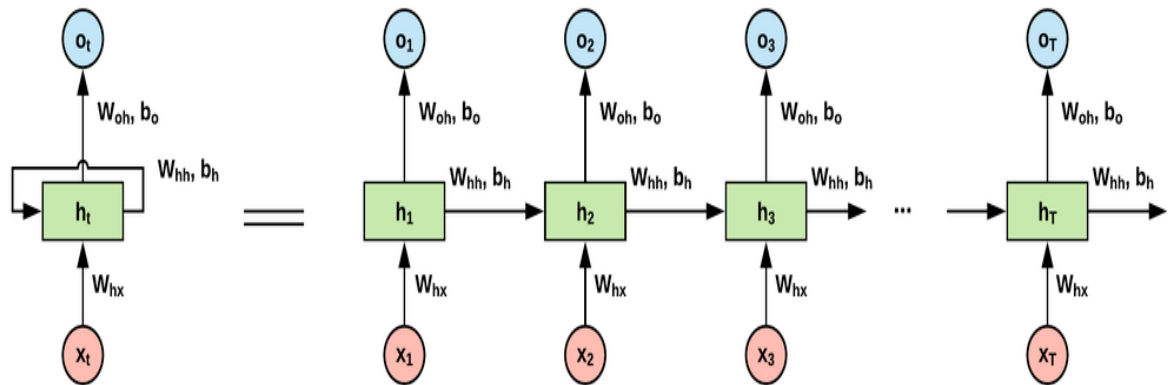


Figure 3-1: Schematic representation of an RNN. [38]

If we denote by (x_1, \dots, x_T) an input sequence, with (h_1, \dots, h_T) the corresponding hidden state sequence and with (y_1, \dots, y_T) the output sequence calculated by network, then an RNN with only one recurrent unit is described by the following equations that are repeated for 1 to T:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1}) \quad (3.1)$$

$$y_t = \text{softmax}(W_{hy}h_t) \quad (3.2)$$

Where, W_{hh} is the array of weights of the hidden state to the hidden state, W_{xh} is the array of weights from the input to the hidden state, W_{hy} is the table of weights from the output to the hidden output and σ , are sigmoid and a softmax function applied per element.

From (3.1) is obvious that the hidden layer depends directly on the input sequence x_1, \dots, x_t :

$$\begin{aligned} h_t &= \sigma(W_{xh}x_t + W_{hh}h_{t-1}) \\ &= \sigma(W_{xh}x_t + W_{hh}\sigma(W_{xh}x_{t-1} + W_{hh}h_{t-2})) \\ &\dots \\ &= \sigma(W_{xh}x_t + W_{hh}\sigma(\dots + W_{hh}\sigma(W_{xh}x_1 + W_{hh}h_0))) \end{aligned}$$

Usually the h_0 is initialized with zeros. The hidden state is a function of the input at the current time on the W_{hx} weight vector which is added to the hidden state of the previous time h_{t-1} which in turn is multiplied by the hidden state weight vector W_{hh} . The weights act as filters that determine the significance to be given at the input and in the previous state. The above sum will pass through an activation function σ . Because of the feedback loop at each time, each hidden state contains information not only of the previous state but of each present state.

It is worth mentioning that while Feed Forward neural networks assign one input to one output, RNNs can assign one input to one output, multiple inputs to one output, one input to multiple outputs, and multiple inputs to multiple outputs.

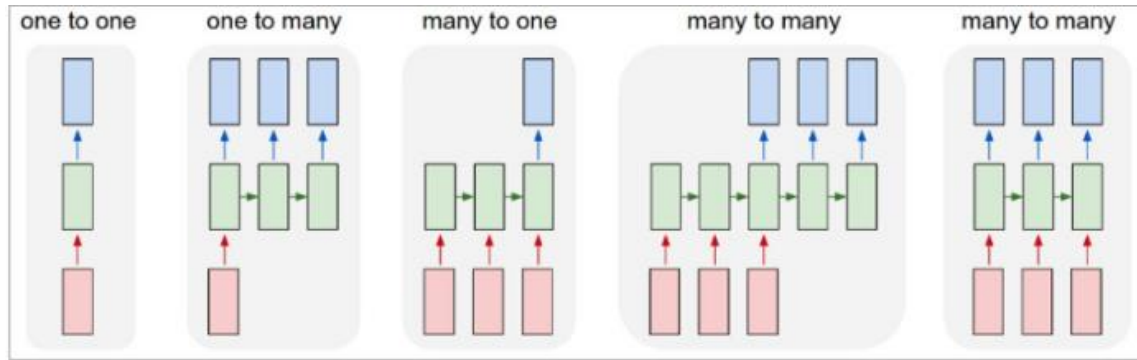


Figure 3-2: Various forms that an RNN network may have. [39]

3.1.2 Backpropagation Through Time

Backpropagation in feedforward networks moves back from the final error through the outputs, weights and inputs of each hidden layer, assigning the responsibility of these weights to a part of the error by calculating their partial derivatives - $\partial E / \partial w$. These derivatives are then used by the gradient descend to adjust the weights up or down, depending on which direction reduces the error. The training of recurrent networks is based on an extension of backpropagation called Backpropagation Through Time (BPTT). Time, in this case, is simply expressed by a well-defined set of parameters linking one step to the next. The BPTT works by unfolding all timesteps. Every step in time has an input step, a copy of the network, and one output. Errors are counted and summed for each time period. The network is folded again and weights are updated, using the following formulas:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \quad (3.3)$$

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} \quad (3.4)$$

3.1.3 Vanishing/Exploding Gradients

Recurrent networks seek to create links between a final output and events that preceded many steps in the past, so it is very difficult to know how much importance should be given to older inputs. This is mainly because the information, flowing through the neural networks, passes through multiple stages of multiplication. Any quantity that is frequently multiplied by an amount slightly larger than one can become incalculably large, and vice versa, a number multiplied by an amount less than one can become incomprehensibly small. If the gradient becomes so small that it disappears we say we have a problem of vanishing gradients. On the contrary, if gradient values are greater than 1, the continuous multiplication of the tables begins to increase the value of the derivatives exponentially. This is defined as an exploding gradients problem.

More specifically, the problem of vanishing gradients is a problem we encounter in artificial neural networks due to the gradient descent learning mechanism. Each weight receives a change according to the error factor in relation to the specific weight at each repetition of the training. The first problem arises when the derivative is small enough. Continuous multiplications due to the chain rule lead to a marginal zeroing of the change, which prevents weight from changing their values and thus stop training. The second problem is occurs in the case where the derivative is of great value. Now the chain rule causes the change to get quite high prices which lead to extreme increases in the value of weights, which also stops the training process.

Below are the results of applying a sigmoid function multiple times, where the slope gradually disappears.

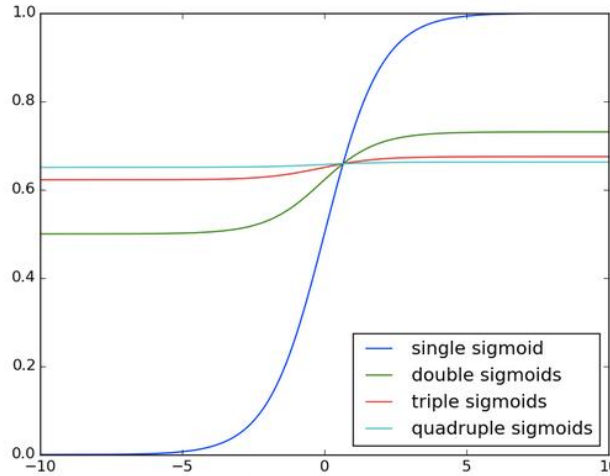


Figure 3-3: Results of applying the sigmoid function multiple times.[40]

3.1.4 Long Short-Term Memory (LSTM)

To overcome the problems mentioned above, Hochreiter and Schmidhuber in 1997 proposed long-short term memory networks (LSTM)[27][28]. Since then, LSTM networks have revolutionized the fields of speech recognition, machine translation, etc. Like conventional RNNs, LSTMs also have a chain-like structure, but the recurrent units have a different structure. Instead of having a single layer of neural network, there are four, which interact in a specific way.

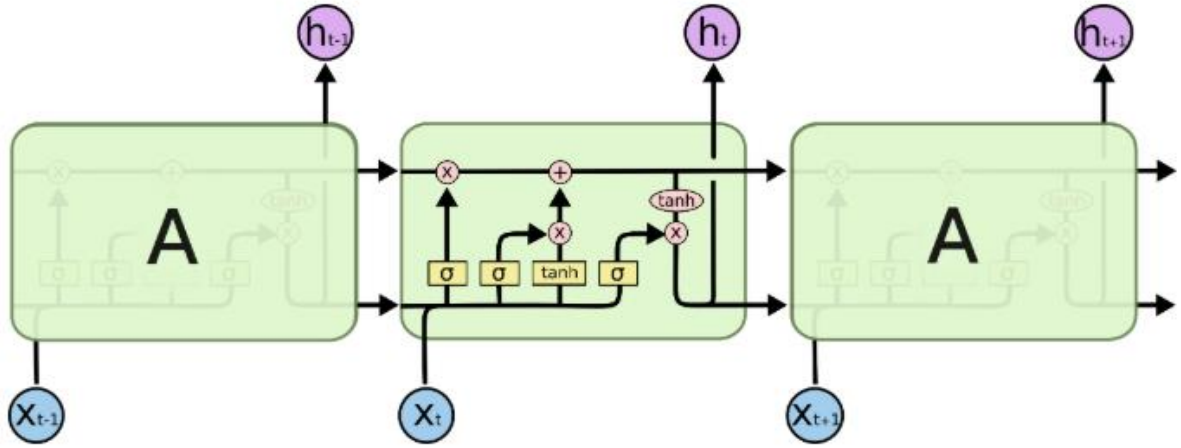


Figure 3-4: Architecture of an LSTM. [41]

The most common architecture consists of a cell and three gates that regulate the flow of information within lstm: an input gate, an output gate, and a forget gate. They are called gates, because the sigmoid function compresses the values of these vectors between 0 and 1 and with the multiplication per element with another vector, they can determine how much information the other vector wishes to hold. The cell is responsible for monitoring the dependencies between the elements in the input sequence. The **input gate** controls the extent to which a new value flows into the cell. The **forget gate** controls the extent to which a value in the cell remains and the **output gate** controls the extent to which the value in the cell is used to calculate the LSTM output. There are connections to and from the LSTM gates, some of which are retrospective. The weights of these connections, which need to be trained during training, determine how the gates work.

To better understand how LSTMs work, let's see how the hidden state h_t is calculated:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (3.5)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (3.6)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (3.7)$$

$$c_t = f_t \circ C_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (3.8)$$

$$h_t = o_t \circ \sigma_h(c_t) \quad (3.9)$$

The key behind the LSTM network is the horizontal line at the top, known as cell state. The cell state passes through all the repeating sections and modifies them from each other with the help of a gate. This causes the maintenance of information in the network. More specifically:

- Forget gate f_t : Decides what information is to be "thrown" by the cell state. This is done by looking at h_{t-1} and x_t and generating a number between 0 and 1 for each number in the cell state C_{t-1} . The number 1 represents "totally keeping the number" while 0 represents "totally forgetting the number".
- Input gate i_t : determines what new information is to be stored in cell state. This process has two parts. First, a sigmoid layer that decides which values will be updated. Then, a tanh layer creates a vector of new candidate values, C_t , which could be added to the state. In the next step, we will combine these two parts to update the new state.
- Output gate o_t : decides how much information from the internal state wants to expose to the external network. The output is based on the cell state, but it is a filtered version of it. We run a sigmoid layer that decides which parts of the cell state will exit.

- Internal memory C_t : We multiply the old state with the f_t , forgetting the things we decided to forget earlier. Then we add $i_t * C_t'$. These are the new candidate values, scaled by how much we decided to update each situation.
- Hidden state h_t : the cell state goes through a \tanh (to push the values to be between -1 and 1) and multiplies with the output gate so that we can only export the parts we have decided.

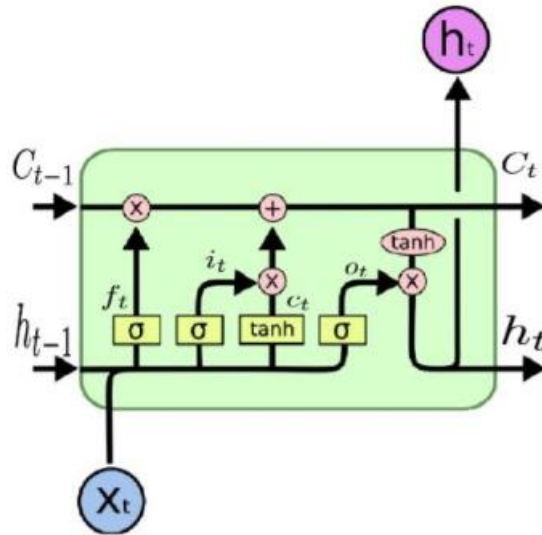


Figure 3-5: Cell structure of an LSTM cell. [41]

3.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional neural networks [1],[5],[6] are deep neural networks that are primarily used to categorize images, detect objects within images, or group them into content-based clusters. It is similar to ordinary neural networks since they are composed of neurons with weights and biases. Each neuron receives inputs, executes an internal product, and optionally uses a non-linear function. They still have a loss function (e.g. Softmax) in the last fully connected layer and all the tricks we have developed for training neural networks continue to apply.

In contrast to traditional neural networks, CNN architectures explicitly admit that the inputs are images that allow us to encode certain properties in architecture. These, therefore, make forward function more efficient to implement and significantly reduce the amount of parameters in the network.

3.2.1 Architecture

First, we will analyze the shape of an image that enters such a network. The input is a 3-dimensional array of the form $H \times W \times 3$, where H (height) corresponds to the number of pixels of the image on the vertical axis, W (width) corresponds to the number of pixels of the image on the horizontal axis and 3 is the 3 RGB color channels (Red, Green, Blue). Each parameter (red, green, blue) determines the color intensity with an integer from 0 to 255. If the images are black and white the input array will be in the form $H \times W \times 1$.

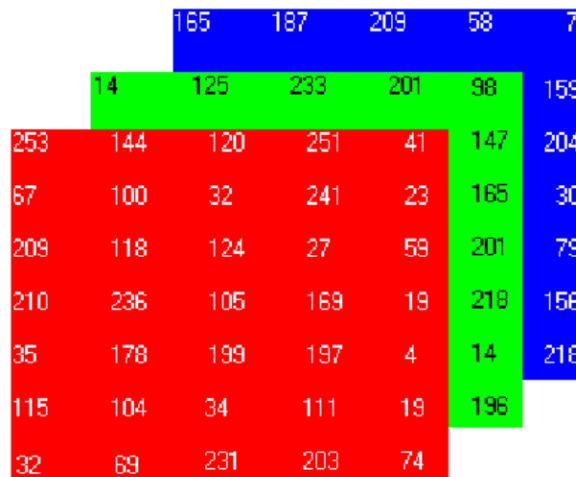


Figure 3-6: Form of a color image. [42]

The input then goes through a series of processing that result from a combination of the following levels: a convolutional layer, a pooling layer, Relu layer and a fully connected layer.

3.2.2 Convolutional Layer

The convolutional layer, as described in [1], is the basic unit of construction of a Convolutional Network, which performs the most demanding calculations. The main purpose of this level is to extract attributes from the input image.

The Convolution layer uses a set of filters that detect the presence of specific features or motifs presented in the original image given in the input. They usually have smaller dimensions than those of the original image, but retain the depth dimension the same as that, i.e. 3. Each filter slides across the input image, and an inner product is calculated to provide an activation map. Different filters that detect different characteristics rotate in the input image and a set of activation maps are the output. The term activation maps refers to image areas that have been mapped to attributes associated with the filters. Intuitively, the network will be trained on filters that are activated when they see a type of visual feature such as the edge of an orientation or the spot of some color on the first level. Thus, we have now acquired a whole set of filters at each convolutional level, each of which will produce a two-dimensional activation map. We will stack these activation maps at the third dimension (depth) and finally get the three-dimensional output.

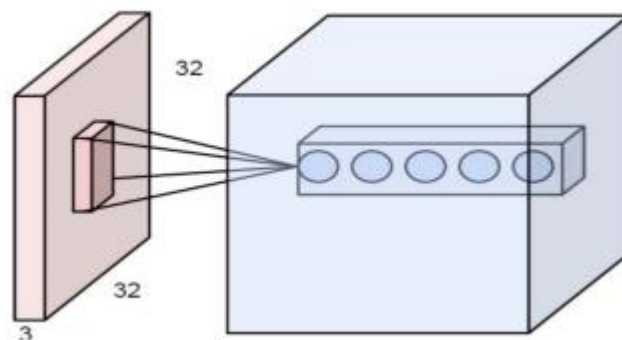


Figure 3-7: A representation of convolutional layer. [1]

. In the example that follows, neurons use different filters but they process the same part of the image.








Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 3-8: Processing an image using different filters. [1]

Before we analyze the remaining levels, we should mention some hyperparameters that control the size of the output:

1. The stride determines how many pixels the filter moves over the image. For example, when the stride has a value of 1, the filter will be moved by one pixel, whereas if it has a value of 2 it will be moved by two, etc. This not only reduces the size of the output, but also the overlap of neighboring levels. Given an image of $N \times N$ dimensions, $F \times F$ dimensional filter, and S the size of output G will be:

$$C = 1 + (N - F) / S$$

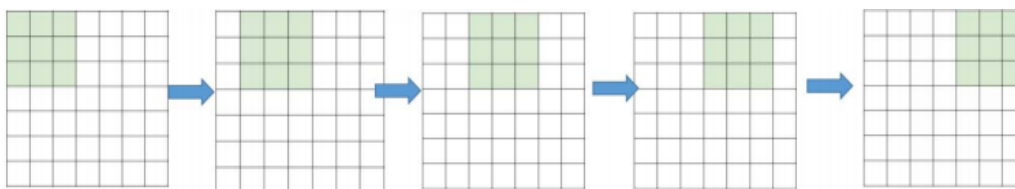


Figure 3-9: Example of a filter with stride = 1. [1]

2. The convolutional layer tend to lose information from the borders of the image. An effective way to solve this problem is to apply zero-padding in order to fill the image borders with zeros.

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Figure 3-10: Application of zero padding at the borders of an image. [1]

Corresponding to the previous case, if P is the number of levels of zero padding, the output will have a size:

$$O = 1 + (N + 2P - F)/S$$

Summarizing, the convolutional layer has the following characteristics: It takes as input an image size $H1 \times W1 \times D1$ and it is determined by the following parameters:

1. Number of filters K
2. Their spatial size F
3. Step S
4. The zero padding number P

When given these parameters, the resulting output will have a size of $H2 \times W2 \times D2$, where

- $W2 = 1 + (W1 + 2P - F)/S$
- $H2 = 1 + (H1 + 2P - F)/S$
- $D2 = K$

3.2.1.2 Relu Layer

The next level after convolution is the implementation of a non-linear function. This level does not affect the size of the output but is applied to saturate or limit the output. Any of the known functions such as tanh or sigmoid can be used as a non-linear function but the one most used is ReLU for the following reasons:

- Due to the simple definition of the function itself and its derivative
 $R(x) = \max(0, x)$ $dReLU(x) / dx = \{1 \text{ } x > 0, 0 \text{ } x \leq 0\}$
- The tanh and sigmoid functions cause backpropagation problems. In deep neural networks, the gradient gradually disappears because the derivative of these functions is close to 0 in almost the whole set of values except the center.

ReLU is an element-wise operation (applied per pixel) and replaces all negative pixel values in the attribute map with zero. The purpose of ReLU is to introduce non-linearity to ConvNet, as most of the real-world data we would like the network to learn will be nonlinear.

3.2.1.3 Pooling Layer

The Pooling Layer is a layer that is usually introduced between successive convolutional levels in Convolutional Network architecture. The idea behind this level is sampling to reduce complexity for

the next layers while retaining important image information. The method may be considered to be equivalent to reducing the resolution of an image. The pooling method used in most cases is max pooling. It separates the image into smaller orthogonal regions and returns from each subregion the pixel with the highest value.

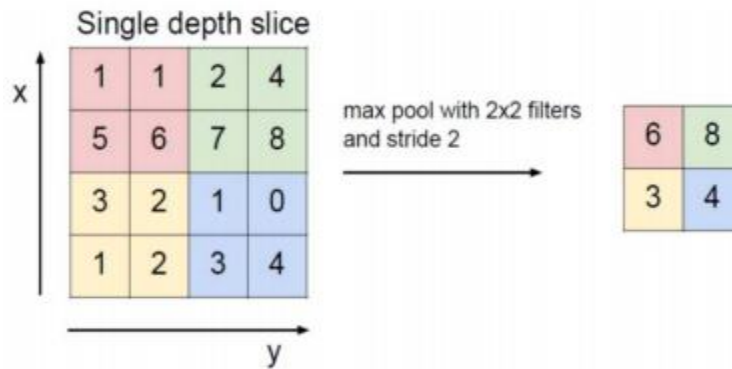


Figure 3-11: Application of a 2x2 Max-pooling. [1]

Instead of taking the largest element, we can also get the Average Pooling or the sum of all items in this window. In practice, Max Pooling has been proven to work best.

The pooling layer has the following characteristics: It accepts an image size $H1 \times W1 \times D1$ and it is determined by the following parameters:

1. Their spatial size F
2. Step S

When given these parameters the resulting output will have a size of $H2 \times W2 \times D2$, where

- $W2 = 1 + (W1 - F)/S$
- $H2 = 1 + (H1 - F)/S$
- $D2 = D1$

3.2.1.4 Fully Connected Layer

This is a common neural network with only one layer. Each node is connected to each node of both the previous and the next layer and uses an activation function at its output. The purpose of the fully connected layer is to use these attributes to classify the input image into different classes, based on the set of data used for training.

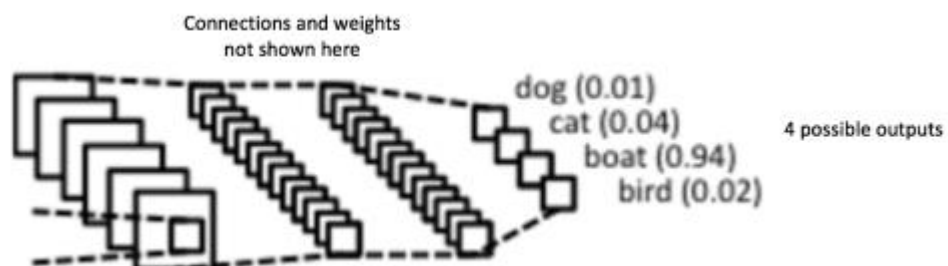


Figure 3-12: Example of a FC layer after a convolutional layer. [1]

3.2.3 Training

The technique used for training convolutional networks [7] is similar to that of feed forward networks. Firstly, there is one forward pass and an output is produced. Then the error is calculated and the error of each weight / filter is updated to pass parameters backwards.

Algorithm 3.2 : BackPropagation for CNN

1. Initialize filters and parameters with random values
2. An image is inserted into the network, passing through all existing levels (convolutional, pooling, etc.) and calculate the output probabilities for each class.
3. Calculate total error at output level.
4. Using the backpropagation method mentioned above, we calculate the error gradients with respect to all the weights of the network and then apply the gradient descent to update all filter values / weights and parameter values to minimize the output error as follows:
 - a) The weights are adjusted according to their contribution to the overall error
 - b) Parameters such as the number of filters, filter sizes, network architecture, etc., are pre-set before step 1 and do not change during training - only filter values and connection weights are updated.
5. Repeat steps 2-4 for each pictures in the training data.

Chapter 4: Related Work

4.1 OBJECT DETECTION

The first efficient algorithm in object detection was one created for face detection in 2001 by Pau Viola and Michael Jones and it was name Viola-Jones Algorithm. They hand-coded features like the location of eyes, nose, mouth and the relations to each other and fed them into a classifier, a support vector machine. In 2005 a paper published by Navneet Dalal and Bill Triggs featuring the Histograms of Oriented Gradients (HOG) outperformed any algorithm created until that moment.

Deep Learning algorithms became widely used in computer vision with its resounding success at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2012, where they outperformed all other algorithms.

The most advanced methods solving the task of object detection using again CNNs is Faster R-CNN [8]. Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the basis of several 1st-place entries in the tracks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. RPNs completely learn to propose regions from data, and thus can easily benefit from deeper and more expressive features. These results suggest that this method is not only a cost-efficient solution for practical usage, but also an effective way of improving object detection accuracy. For this thesis we used Mask RCNN which is an extension of Faster RCNN, explained in detail in the next section.

4.1.1 Mask RCNN

Mask RCNN [8] (Regional Convolutional Neural Network) is the most modern approach that instance segmentation It consists of two stages: one generates regions where it is likely to be an object and second it generates masks in pixel level.

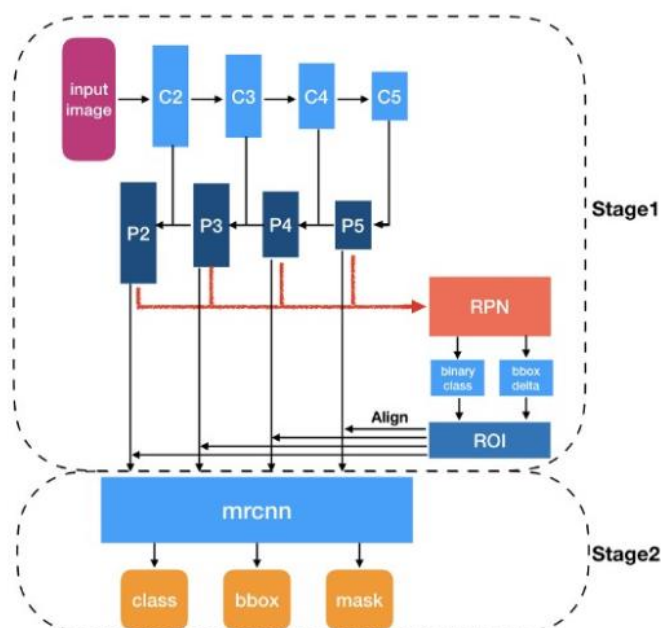


Figure 4-1: Mask R-CNN structure. [43]

Backbone is the first stage of the mask-RCNN and it is a convolutional neural network typically, ResNet50 or ResNet101 that extracts features. The early layers detect low level features (edges and corners), and later layers successively detect higher level features (car, person, sky). The backbone can be improved using the Feature Pyramid Network known as FPN which is created by adding a second pyramid in the standard feature extraction pyramid that takes the high level features from the first pyramid and passes them down to lower layers, it allowing features at every level to have access to both, lower and higher level features.

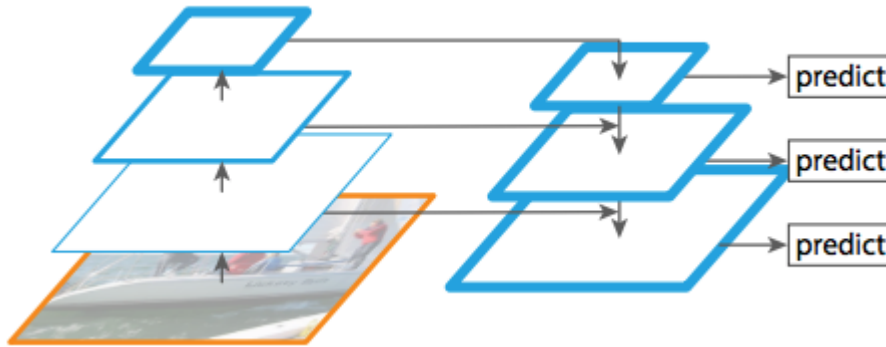


Figure 4-2: Feature Pyramid Networks. [44]

The first part of the network is a Region Proposal Network. The RPN is a lightweight neural network that scans all FPN top-bottom pathway in a sliding-window fashion and finds areas that contain objects. The regions that the RPN scans over are called anchors, which are boxes distributed over the image area. The RPN generates two outputs for each anchor a binary anchor class and does bounding box refinement.

Using the RPN predictions, we pick the top anchors that are likely to contain objects and refine their location and size. One of the great advantages of the RPN is that it does not scan the actual image, the network scans the feature map, making it much faster.

Next is the ROI Classifier. This stage runs on the regions of interest (ROIs) proposed by the RPN and produces a class of the 80 total classes that it is trained to recognize, does some more refinement on the bounding boxes and produces the final results. At this stage ROIAlign is used to have all bounding boxes the same size.

At the last stage the regions suggested by the previous layer are used from the model in order to create masks. Masks are low 28x28 pixels resolution and are represented with float numbers.

4.1.2 Image Classification

Image Classification is an important task within the field of computer vision. Image classification refers to the labelling of images into one of a number of predefined categories. Classification includes image sensors, image pre-processing, feature extraction and object classification. Many classification techniques have been developed for image classification like Artificial Neural Network (ANN), Decision Tree (DT), Support Vector Machine (SVM) and Fuzzy Classification.

For the purpose of this thesis we will focus on approaches based on Artificial Neural Networks. In the most recent work in computer vision, variations of neural networks trained with stochastic gradient descent are mostly used. One of the state of the art pre-trained models for this task is VGG-16 [12] proposed by Karen Simonyan & Andrew Zisserman from the University of Oxford in 2014. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second

convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

Another state of the art model is Inception V3[2]. Inception-v3 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 48 layers deep and can classify images into 1000 object categories. Inception-v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years.

These models are trained to classify an image, i.e. assign a class label to it. It proved to be very efficient to utilize a pre-trained image classification model in similar tasks. From the great variety of pre-trained models, VGG-16 and Inception V3 achieve the state-of-the-art performance. We choose to implement Inception V3 in our model for the task of feature extraction from the images.

4.1.3 Inception_V3

The convolutional neural network discussed in this section is the next generation of GoogleNet by Christian Szegedy, Google's researcher on Machine Learning, Artificial Intelligence and Computer Vision, through deep learning. Inception_V3 is a widely used image classification model that has been proven to achieve a precision of more than 78.1% in the ImageNet data set.

The model itself consists of symmetrical and asymmetric structural elements such as convolutions, average pooling, max pooling, concatenations, dropouts and fully connected layers. The training of this network has been done in a thousand classes, meaning it is capable of recognizing a thousand different objects.

The basic principles governing this network and making it effective compared to other similar networks are:

- Avoid bottleneck, especially at the beginning of the network.
- High-resolution rendering as it is easier to move locally to a network.
- Implementation of spatial aggregation in small dimensional implants. For example, before implementing a 3×3 convolution, we can reduce the dimensions of the representation of the entrance, without having to deal with some serious consequences.
- Balance between the width and depth of the network

The core concept behind this model is the inception module shown in Figure 4-4. Inception module was firstly introduced in Inception-v1 / GoogLeNet. The input goes through 1×1 , 3×3 and 5×5 convolution, as well as max pooling simultaneously and concatenated together as output. The 1×1 Convolutional layers before applying another layer, is used for dimensionality reduction.

The aim of factorizing Convolutional is to reduce the number of connections/parameters without decreasing the network efficiency. Convolutions with larger spatial filters tend to be computationally expensive.

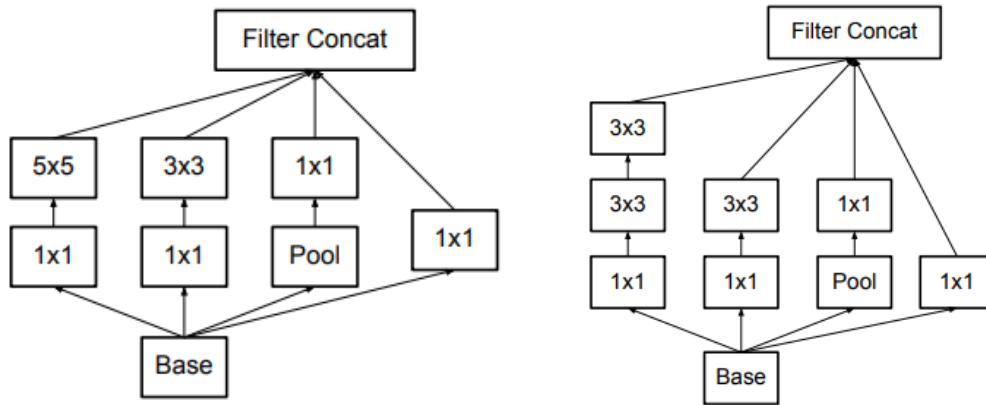


Figure 4-3: (a) Original Inception module (b) 5×5 convolution is replaced by two 3×3 convolution. [2]

For example, by using 1 layer of 5×5 filter the number of parameters for this layer is $5 \times 5 = 25$ and using 2 layers of 3×3 filters the number of parameters is $3 \times 3 + 3 \times 3 = 18$, the number is reduced by 28%.

These suggest that the convolutions with filters larger than 3×3 might not be useful as they can always be replaced into a sequence of 3×3 convolutional layers. A bigger improvement it would be to use asymmetric convolutions For example using a 3×1 asymmetric convolutions followed by a 1×3 convolution.

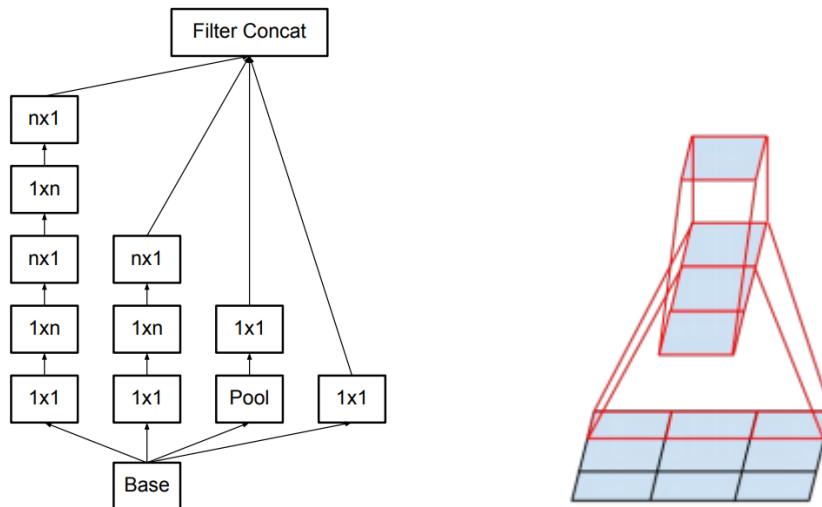


Figure 4-4: (a) Factorization of the $n \times n$ convolutions. (b) One 3×1 convolution followed by one 1×3 convolution replaces one 3×3 convolution. [2]

For example, by using 3×3 filter, the number of parameters is $3 \times 3 = 9$, whereas by using 3×1 and 1×3 filters, the number of parameters is $3 \times 1 + 1 \times 3 = 6$. If we use two 2×2 filters, the number of parameters is $2 \times 2 \times 2 = 8$, number of parameters is only reduced by 11%.

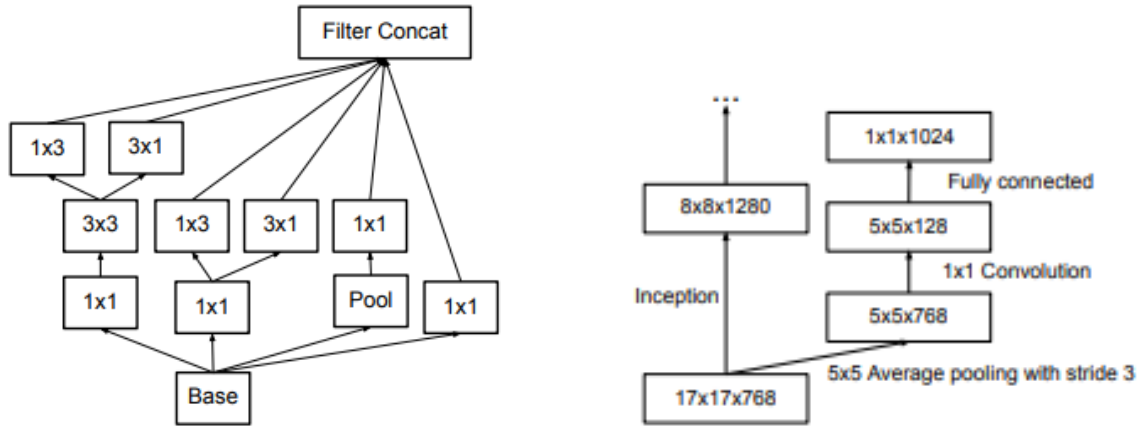


Figure 4-5:(a) Inception modules. (b) Auxiliary classifier [2]

The architectures shown in Figure 4-5(a) and Figure 4-6(a) are the 2 different inception modules suggested by the author. With factorization, number of parameters is reduced for the whole network, it is less likely to be overfitting.

Auxiliary Classifier are used to push useful gradients to the lower layers in order to make them immediately useful and improve the convergence during training by combating the vanishing gradient problem in very deep networks. In GoogLeNet / Inception-v1, auxiliary classifiers are used for having deeper network. In Inception-v3, auxiliary classifier is used as regularizer. An example is shown in Figure 4-6 (b) where one auxiliary classifier is used on the top of the last 17×17 layer.

In most cases, convolutional networks use the pooling operation to decrease the grid size of the feature maps. In [2], an efficient grid size reduction is proposed as follows: with the efficient grid size reduction, 320 feature maps are done by convolution with stride 2. 320 feature maps are obtained by max pooling. And these 2 sets of feature maps are concatenated as 640 feature maps and go to the next level of inception module. With this approach the model is less expensive but still efficient.

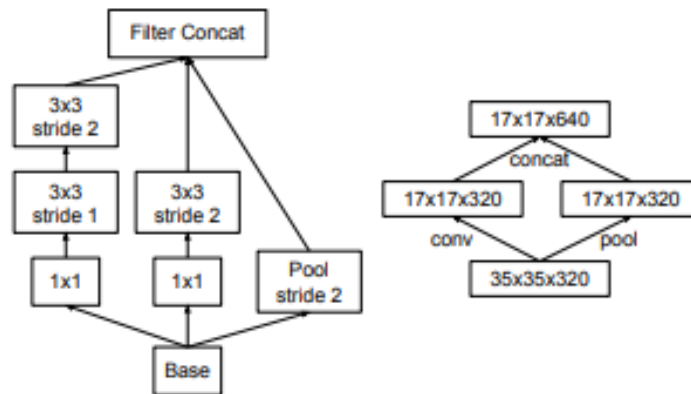


Figure 4-6: Inception module for grid-size reduction. [2]

Inception v3 network stacks 11 inception modules where each module consists of pooling layers and convolutional filters with rectified linear units as activation function.

All the above modules are combined to create the total inception_v3 model, shown below. The model has 3 modules like the ones shown in Figure 4-5(a) at the beginning of the model. Then there is a grid size reduction layer, and after there are again 4 inception modules like in Figure 4-5 (a) followed by

another grid size reduction module and an auxiliary classifier (shown in Figure 4-8 at the bottom right). Last, two modules from Figure 4-6(a) are connected to the model and at the end a fully connected layer with a softmax classifier for the classification task.

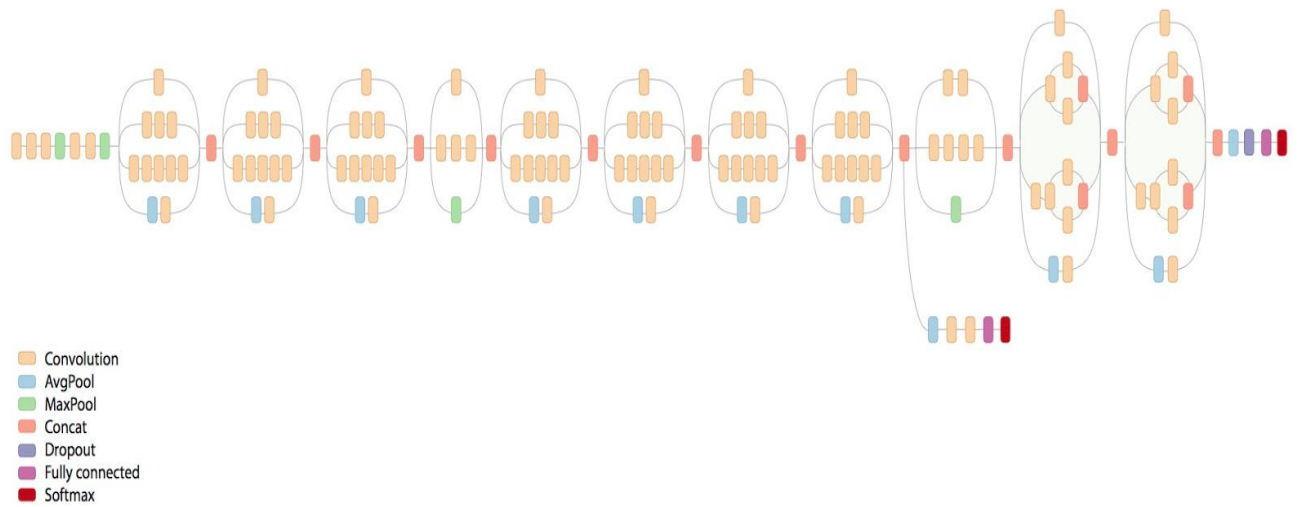


Figure 4-7: Schematic diagram of Inception V3. [46]

4.2 IMAGE CAPTIONING

One of the first approaches in image captioning, using attention, is the one explained in the paper “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention” [13]. This approach utilizes the encoder-decoder architecture with another mechanism in between, the attention mechanism. This paper, describe approaches to caption generation that attempt to incorporate a form of attention with two variants: a “hard” attention mechanism and a “soft” attention mechanism. It shows how we can gain insight and interpret the results of this framework by visualizing “where” and “what” the attention focused on. Before this work attention was only used for machine translation.

Another state of the art approach is the one described in the “Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering”[14], where our model is based on. The innovation about this model is that typically, attention models operate on CNN features corresponding to a uniform grid of equally-sized image regions. In this approach attention is calculated at the level of objects and other salient image regions. Applying this approach to image captioning, the results on the MSCOCO test server establish a new state-of-the-art for the task.

For the purpose of this thesis, we also examined other similar implementations like [53], in order to compare our methods and results .

Chapter 5: Our Image Captioning Model

In this chapter we will analyze the neural model we used to produce descriptions of the content of images. For this we use recurrent and convolutional neural networks to encode the input image into a vector and use this encoding to decode the desired output sequence. Some examples are shown in Figure 5.1.

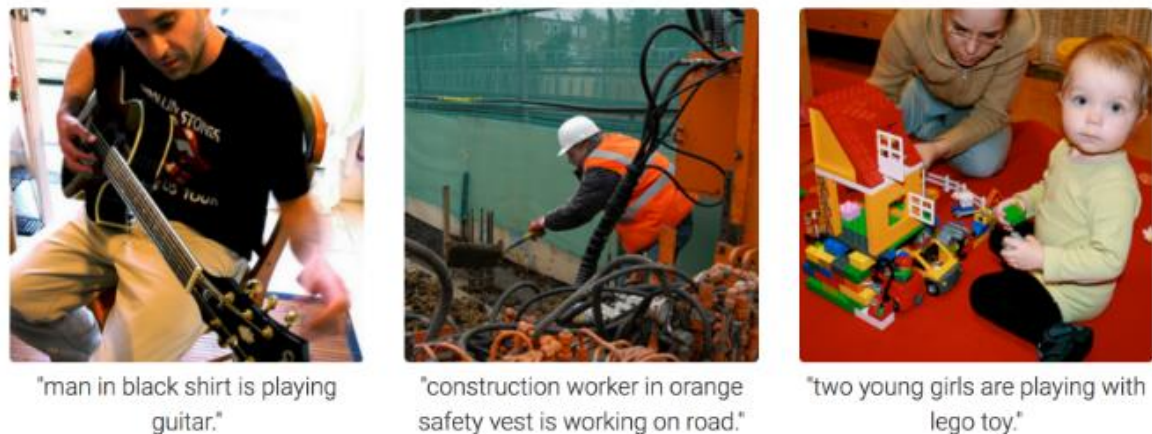


Figure 5-1: Image Captioning Examples. [47]

5.1 OVERVIEW OF MODEL ARCHITECTURE

The model we developed it is based on sequence to sequence model [9]. Introduced for machine translation in 2014 by Google, a sequence to sequence model aims to map a fixed length input with a fixed length output where the length of the input and output may differ. In its general form the model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

The **Encoder** consists of several recurrent units where each accepts a single element of the input sequence, collects information for that element and propagates it forward. The **Encoder Vector** is the final hidden state produced from the encoder part of the model. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions a. The **Decoder** also consists of several recurrent units where each predicts an output for each time step.

Our approach maintains a similar structure. First, the image enters as an input to the encoder. The encoder undertakes to extract features from the image and represent them in the form of vectors so that they can be processed by the next part of the network, the decoder. The decoder receives this information and tries to express it in natural language. The difference is that our encoder model does not contain an RNN network. Instead, the encoding of the image is a combination of two pre-trained models: Inception V3 and Mask-RCNN. More analytically, the encoder consists of a convolutional network, as those discussed in Chapter 2. These networks are widely used in image detection and recognition. The object detection has been greatly emphasized and models have been developed, such as those described here, which perform best compared to other methods.

The decoder is an LSTM network. Words are entered into the network through an Embedding Layer, coding words into vectors that neural networks can process. These vectors then enter as an input into the LSTM, which given a word undertakes to predict the next. In contrast to the decoder mentioned above, our decoder has an attention layer. The attention layer will process different parts of the image and decide where more emphasis should be placed.

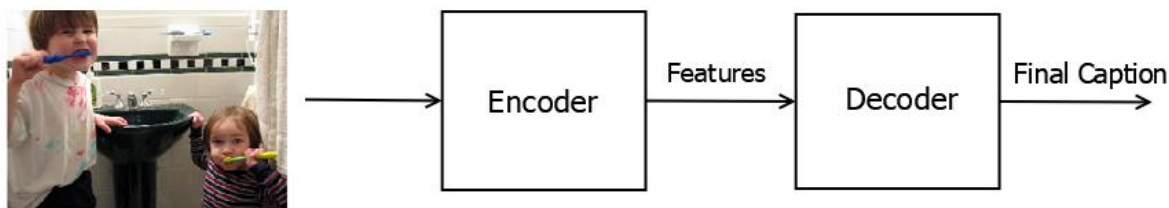


Figure 5-2: A high-level block diagram of our model.

In the following sections we will analyze step by step each component of the above model.

5.2 ENCODER

In recent years, the performance of convolutional neural networks in classification problems has risen steeply due to continued improvements in the field of deep learning, particularly in neural networks. Most importantly, these improvements have not only been made by the development of hardware and larger data sets, but also by new ideas and algorithms that improve network architecture. The Encoder model consists of the following parts:

- A mask R-CNN for detecting important regions from the input image.
- Inception_V3 model for extracting features from the whole image and each region.
- Another dense layer for reducing the size of the feature vector.
- Dropout layers for preventing overfitting.

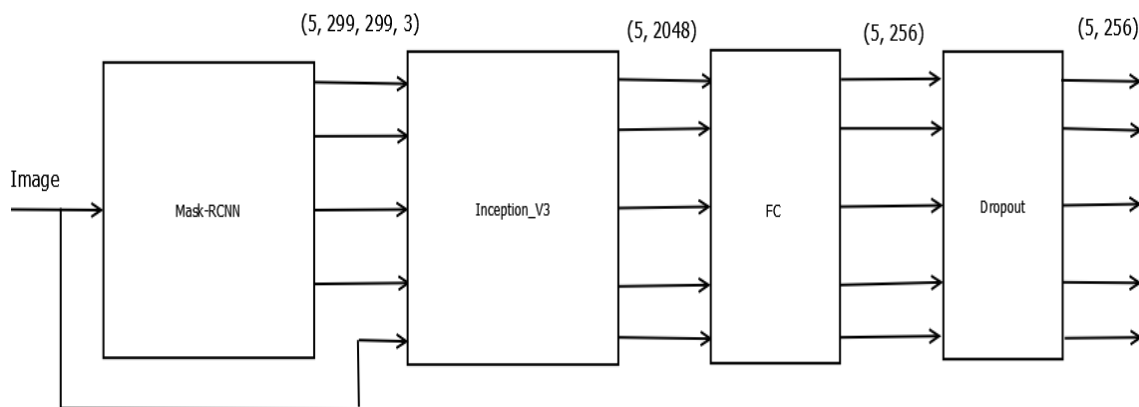


Figure 5-3: Encoder model architecture

5.2.1 Mask R-CNN and Inception_V3 in our System

The input image first enters the mask-RCNN model. The purpose of this model is to scan the image and then detect the different regions of interest containing objects. These objects must belong to the 80 class objects that the model is trained to detect. From the regions that are produced by this model we keep only the top-4 regions. By top-4 we mean the 4 regions with the highest probabilities to contain an object that belong to one of the 80 classes mentioned above. After these regions are produced we use the bounding boxes to crop the initial image at the places suggested by the mask-RCNN model. Now instead of only one image, we have also another 4 that contain the most interesting part of it. The reason we choose number four is because we calculated that the mean of the number of objects in the training images detected is 3.9.



↓ Mask-RCNN



↓ keep the 4 regions with higher likelihood



Figure 5-4: A visual example of how we use the Mask-RCNN model

These regions are then passed through Inception_V3 model for feature extraction. As previously mentioned, it is the second part of the model where five 299x299x3 images enter, pass through some processing steps and a feature vector is created for each one of them that contains useful image characteristics. It is worth noting, that we are not interested in what is contained in the pictures explicitly, as at this point it is not an image classification problem. What interests us is to get some important image features that will be promoted to the LSTM. For this reason, we do not take the output from the last layer of the network where the softmax is located because it will give us a probabilistic distribution among the thousands of classes it has been trained on. Instead, the output will result from the third from the end layer, which is a trigger restriction level. From this layer, we get a tensor of size 1x2048 for each image region and the whole image. So in total 5x2048 feature vectors are computed.

5.2.2 Image Embedding

The outputs of the Inception V3, as we have already mentioned, will be the input to the LSTM network, which is responsible for generating reasonable suggestions for describing images. For this, we must do some modifications. In order to reduce computational costs and for better data management, we should change the size of the 5 parts of the images from 2048 to 256. For this transformation, we will need to add a fully connected layer, as the ones mentioned in Chapter 2, immediately after producing the feature vectors. We also added a dropout layer to increase the generalization power of the encoder and prevent overfitting. The output of this layer will be a tensor of size (batch_size x 5 x 256). Now we have created the encoder vector that will be propagated to the decoder.

5.3 DECODER

The purpose of this model is to decode the feature vectors, produced by the encoder, into rational sentences. The decoding of the images and the words goes through the following stages:

- Word Embedding for encoding every word to a vector that can be processed by the neural network.
- Attention for focusing on specific parts of the input image.
- LSTM for combining all the above to produce a word at each time step.

The full decoder model is shown in the next figure.

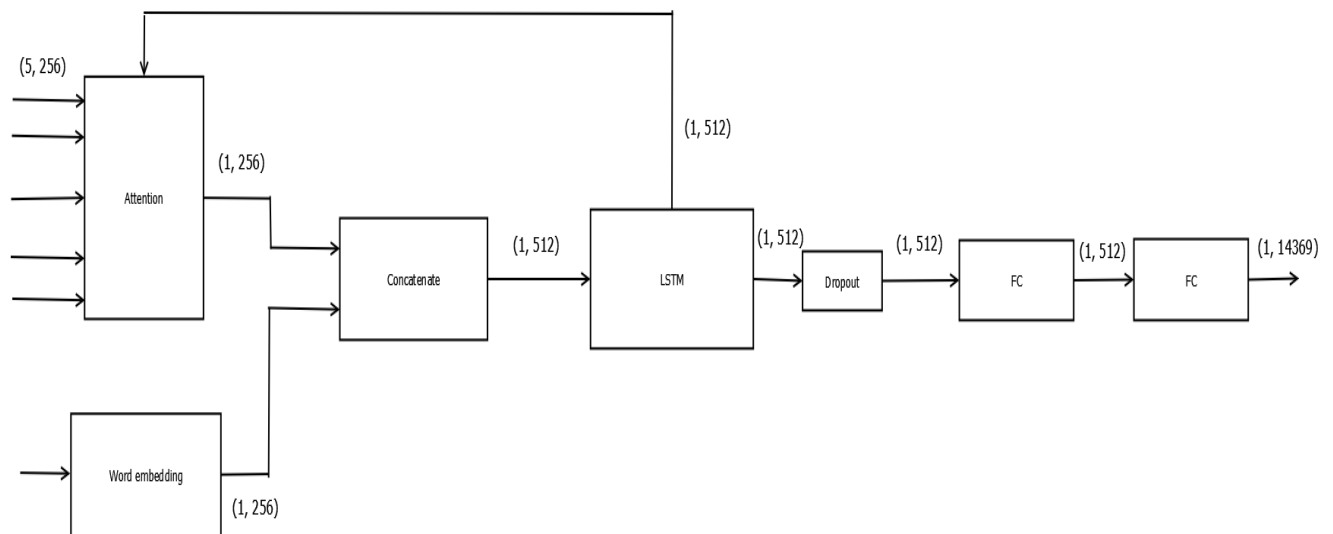


Figure 5-5: Decoder model architecture

5.3.1 Word Embedding

Neural networks, by their nature, are so constructed that they can process only numbers. As a result, the representation of words as a string is impossible and therefore it is necessary to find a mechanism that will represent each word with a single vector made up only of numbers.

The first step in the representation process is to determine how many different words our model can detect, i.e. to create a vocabulary. The set of training data that we have at our disposal, which will be discussed in more detail in the next chapter, consists of a number of images and their respective descriptions in character sequences in English. The vocabulary will be produced from all the different

words that appear in the descriptions of the images of the training dataset. To do this, we first need to do a Tokenization, in order to divide the sentences into tokens, in the way illustrated below:

<start> two cars parked on the sidewalk on the street <end> →
[<start>, two, cars, parked, on, the, sidewalk, on, the, street, <end>]

Two additional tokens have been added to each sentence. A token <start> that marks the beginning of the sentence and a token <end> that marks the end and are also included in our vocabulary. We also added a token <unk> for the words that are in the training sentences but not in our vocabulary and the <pad> token for the zeros produced by the zero padding. In all, the vocabulary consists of words, sorted in descending order, based on their frequency of occurrence in the descriptions of the training dataset.

The next step is to encode words in numbers in a unique way, so that each number that corresponds to a word is not used to encode any other word. The method followed is the representation of each word with a number corresponding to the position of that word in the vocabulary. Thus the word ‘<pad>’ as the most common in descriptions is first in the vocabulary and for this it has the number 0. Correspondingly, ‘<unk>’ has the number 1, ‘a’ has the number 2, ‘<start>’ has the number 3, ‘<end>’ has the number 4 etc. Now each sentence can be represented by a vector of integers as shown below:

<start> two cars parked on the sidewalk on the street <end> →
[<start>, two, cars, parked, on, the, sidewalk, on, the, street, <end>] →
[3, 16, 204, 68, 5, 7, 193, 5, 7, 25, 4]

Many Natural Language Processing systems and techniques handle words as individual units, without the concept of similarity between words, as they are represented as indicators in a dictionary, in the way we have described the process of representation of words so far. Below are some of these methods and the reasons why we had to reject them.

One way would be to apply the bag of words (BoW). It is called bag of words because in this approach we only concern about the number of words that appear, not the order in which they appear. In more detail, after first tokenizing the sentences in individual words and constructing a vocabulary, we calculate the number of occurrences of each word in the sentence we look at each time as shown in Figure 5-6. So in the case of the first sentence, where every word is unique, each word that appears will have the number one, and in the remaining positions it will have zero. Similarly, in the fourth sentence there are two words that appear twice, is and a. There will be number two.

The bag of words model is used as a tool for generating attributes. After we turn into a bag of words we can calculate frequency to characterize it.

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

Figure 5-6: An example of bag of words model. [\[49\]](#)

We notice that in this case there is no information about the semantics of words, and if two words are semantically identical.

Another method would be to apply One-Hot coding. Each word is encoded by a vector as long as the vocabulary, with zeros in all positions except the position corresponding to that word. We see that in this case there is no information about the semantics of the words and the resulting vectors are quite large.

The above methods, although simple, present several problems that come to solve the embedding vectors. The basic idea of these vectors is to convert words into fixed length vectors, which contain real numbers. The purpose is to create representations of words whose size will be independent of the size of the vocabulary and also have a much smaller number of dimensions. This conversion is necessary as most engineering learning algorithms require inputs to be true value vectors. Two are the main advantages of using word embedding:

- **Dimensionality Reduction.** In the case that we would use one-hot encoding for word representation, each word would be encoded by a vector of numbers of which in positions would have been zeroes and only in one place there would be 1. That as it is obvious is not computationally efficient.
- **Contextual Similarity.** This means that the representation will be in such a way that words with common semantics should have similar representations.

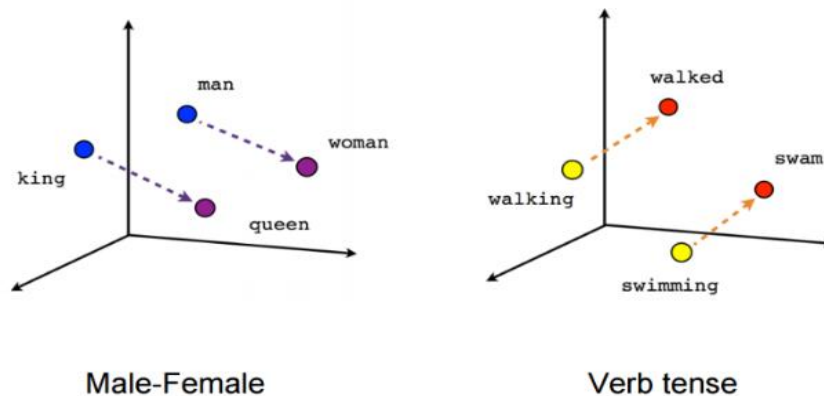


Figure 5-7: Semantic similarities using word embedding. [48]

In Figure 5-7, we can see some semantic similarities between words. For example, the word "king" and the word "queen" will be close enough and will be as far away as the words man and woman. Also, the words "swimming" and "swam" will be in close proximity, since this model will have identified the relationship between the two verbs, that is, they are the same verb at another time.

In our model, the dimensions of the embedding vectors were set to 256. This means that each word - which we refer to as an integer - is represented by a 256 size vector, which contains real values that express the semantic similarity of that word with some other words. Thus, the set of word depictions in vectors is implemented by using a two-dimensional array that has dimensions 14369 x 256, where the number 14369 refers to the number of words in the vocabulary, while 256 is the dimensions of the embedding vectors.

5.3.2 Attention

Attention mechanism is an architecture that enables a neural network to concentrate on the most important parts of the image. A neural attention mechanism equips a neural network with the ability to

focus on a subset of its inputs (or features). Its purpose is to apply the same tactics followed by people. Human visual attention allows us to focus on a certain region with “high resolution” while perceiving the surrounding image in “low resolution”, and then adjust the focal point or do the inference accordingly.

Attention in deep learning can be interpreted as a vector of importance weights: in order to predict or infer one element, such as a pixel in an image or a word in a sentence, we estimate using the attention vector how strongly it is correlated with other elements and take the sum of their values weighted by the attention vector as the approximation of the target.

By utilizing this mechanism, it is possible for the decoder to capture somewhat global information rather than solely to infer based on one hidden state.

Attention can be applied to any kind of inputs, regardless of their shape. In the case of matrix-valued inputs, such as images, we can talk about visual attention. The attention mechanism is better explained in the next algorithm.

Algorithm 4.1 : attention

1. Given a feature vector C of an image and a previous hidden y_i vector
2. Multiply C and y_i with weights $W1$ and $W2$, respectively
3. Add the results
4. Pass the new vector through a \tanh activation function
5. Multiplied the result vector with V weights
6. Then pass this vector from a softmax function
7. Multiply the output of the softmax again with feature vector C
8. Add the results

The attention model can be represented with the structure shown in Figure 5-8.

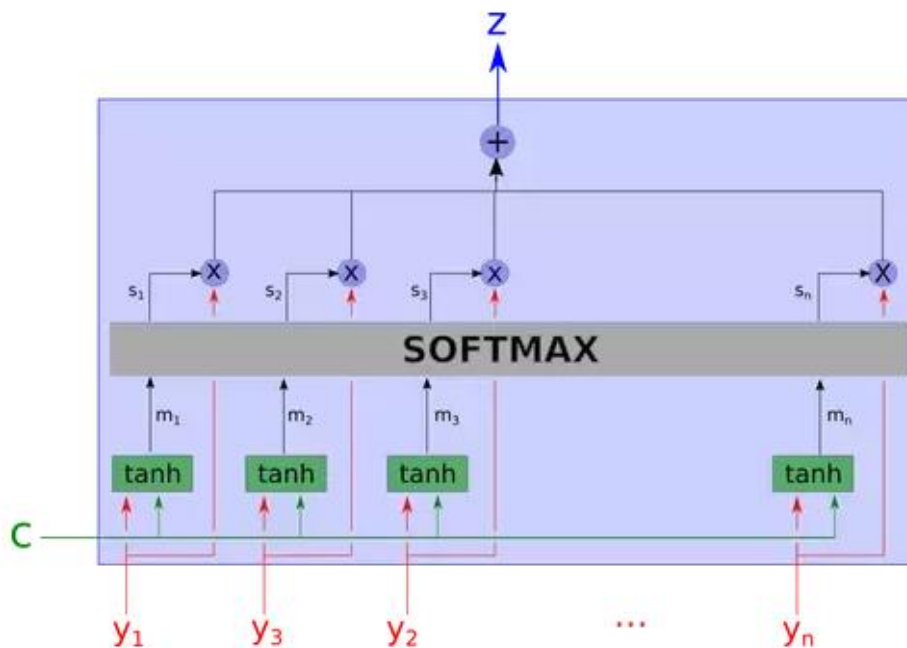


Figure 5-8: Attention Model Structure. [50]

The algorithm can be described with the following mathematical equations:

$$m_i = \tanh(y_i W_{y_i} + C W_c) \quad (5.1)$$

$$s_i = \frac{e^{m_i}}{\sum_n e^{m_n}} \quad (5.2)$$

$$z = \sum_n s_n y_n \quad (5.3)$$

Where y_i is the feature vector, C is the previous hidden state of the LSTM, s_i the attention weights and z the context vector.

In our system attention mechanism follows the same structure. The 5x256 feature vector produced by the image embedding is the input to the attention model, which based on the previous hidden state of the LSTM, with size 1x512, will determine where it must give the most attention among the input vectors. The output of this model is a 1x256 context vector and it is concatenated with the embedding vector of the word at the current time step, producing a vector of size 1x512. This is the input to the LSTM unit. The attention model has another output, called attention weights. These weights are used to visualize the attention mechanism on the image.

At this point it is worth mentioning that attention models operate on CNN features corresponding to a uniform grid of equally-sized image regions, irrespectively of the content of the image. As proposed by [14] to generate more human-like captions we use a different approach where attention is calculated at the level of objects, as shown in Figure 5-9, like the ones suggested by the Mask-RCNN model, as they are a much more natural basis for attention.

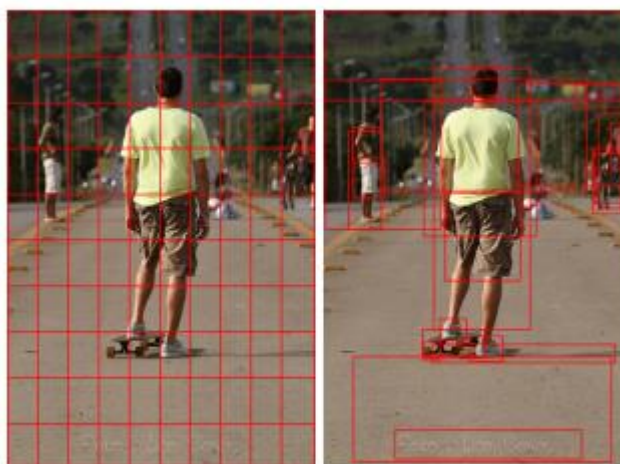


Figure 5-9: A typical attention model with a uniform grid of equally-sized image regions (left). Our approach enables attention to be calculated at the level of objects (right). [14]

An example of what attention does in our model is the following. In the Figure 5-10 below we see how our model focuses on specific part of the image in order to produce a word at each timestep. More examples are in the appendix section.

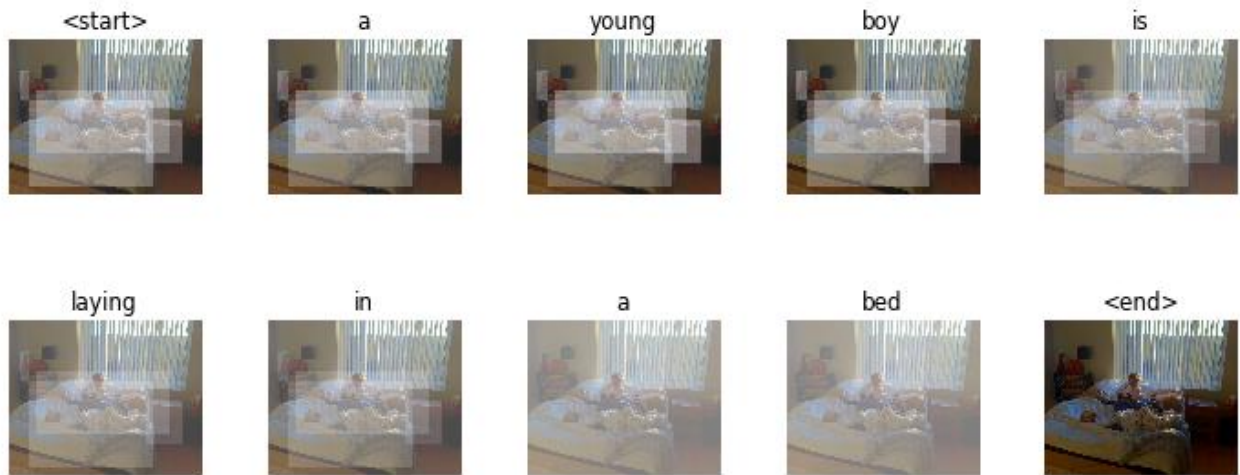


Figure 5-10: Attention Example

5.3.3 Caption Generator Based on LSTM

In this paragraph we will analyze the most important part of our system, which is the LSTM network, as those described in the Chapter 3, which is responsible for creating reasonable suggestions for describing the content of the images. As mentioned in Chapter 3, they not only have a memory, so there is a continuity between the foregoing and subsequent costs, but they also have the advantage of dealing effectively with the problem of vanishing and exploding gradients. The LSTM network takes as input a vector that contains the representation of both the image and a word, represented by an embedding vector, from the known sentences describing the image during training. For this image, a word enters at each timestep the LSTM and it outputs the word with the highest probability. Apart from the embedding vector LSTM takes as input the hidden state of the previous timestep. This processing is repeated until all words in the sentence describing the image are passed through the LSTM.

During training we used the teacher forcing method. Teacher forcing works by using the actual or expected output from the training dataset at the current time step as input in the next time step, rather than the output generated by the network. This method is used in order to reduce learning errors and improve the model skill and stability.

The LSTM unit has an input of size (1, 512) and so will have its output. As we have already mentioned we want to make prediction of a word and for this we need a probabilistic distribution among all words in the vocabulary in order to choose the one with the highest probability. For this we added two more fully connected layers at the end of our model. The first one is used in order to make the model more complex and have more parameters describing the output of the previous layer and the last one for giving the output the size of our vocabulary in order to predict a specific word. At the last layer we use a softmax classifier for creating probabilistic distribution among the words in the vocabulary and then choose the one with the highest probability.

In the figure below we can see an example of how the process of creating a caption works in detail. The sentence describing the image is “<start> A man is sitting on a chair <end>”. The tokens <start> and <end> was added by us, as mentioned before. The input sentence, which will be fed into the LSTM word by word is “<start> A man is sitting on a chair” and the target sentence that our model should be trained to predict is “A man is sitting on a chair <end>”. This is due to the fact that our model must be trained to stop when it meets the <end> token, so there is no need for it to exist in the input sentence and also the <start> token marks the start of the prediction so there is no need for it to be in the output.

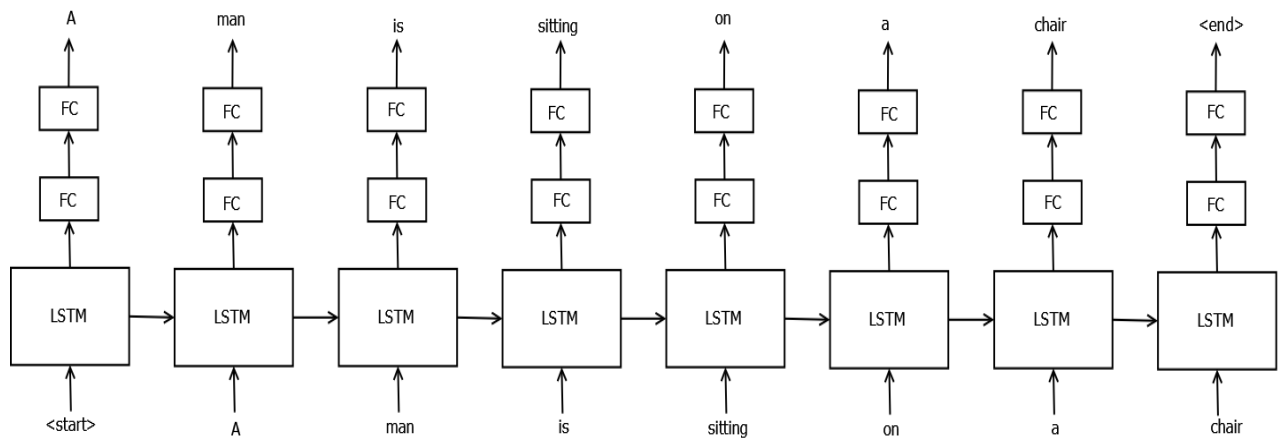


Figure 5-11: An example of how LSTM is used for generating captions

The code used for creating the total model of image captioning can be found here:
https://github.com/NadiaFrh/Diploma_thesis

Chapter 6: Model Training

In this chapter we will analyze the sources from which we derived our data, algorithms and the techniques used to train our model, as well as the results presented by our system.

6.1 DATABASE MSCOCO 2015

The set of data on the database of which we have implemented the training functions of the evaluation and operational control is the MSCOCO 2015 dataset [10]. This is a dataset used for the COCO competition Captioning 2015. The selection of this particular dataset was due to its large number of training examples. More specifically, the training dataset included:

- 80000 training images
- 40000 validation images
- 80 categories of common objects

The data, as officially provided by the COCO website, consists of the images that are in JPEG format, along with their descriptions that are in JSON files. Each image has 5 descriptions. The form in which descriptions are provided in the JSON file is a collection of “info”, “licenses”, “images”, “annotations”, “categories” (in most cases), and “segment info” (in one case).

```
1. {  
2.   "info": {...},  
3.   "licenses": [...],  
4.   "images": [...],  
5.   "annotations": [...],  
6.   "categories": [...],  
7.   "segment_info": [...]  
8. }
```

For the object detection model we used the file `instances_train2014.json` and `instances_val2014.json` for training and validation respectively, which has the above format. From this file for the object detection model we needed only the “categories” and “annotations” object.

```
1. "categories": [  
2.   {  
3.     "supercategory": "person",  
4.     "id": 1,  
5.     "name": "person",  
6.     "keypoints": [  
7.       "nose", "left_eye", "right_eye", "left_ear", "right_ear",  
8.       "left_shoulder", "right_shoulder", "left_elbow", "right_elbow",  
9.       "left_wrist", "right_wrist", "left_hip", "right_hip",  
10.      "left_knee", "right_knee", "left_ankle", "right_ankle"  
11.     ],  
12.     "skeleton": [  
13.       [16,14],[14,12],[17,15],[15,13],[12,13],[6,12],[7,13],[6,7],  
14.       [6,8],[7,9],[8,10],[9,11],[2,3],[1,2],[1,3],[2,4],[3,5],[4,6],[5,7]  
15.     ]  
16.   }  
17. ]
```

There are 80 different categories contained in the dataset plus the one category given by the Mask-RCNN for the background:

```

1. Class Count: 81
2. {0.BG 1.person 2.bicycle 3.car 4.motorcycle 5.airplane
3. 6.bus 7.train 8.truck 9.boat 10.traffic light

4. 11.fire hydrant 12.stop sign 13.parking meter 14.bench 15.bird
5. 16.cat 17.dog 18.horse 19.sheep 20.cow

6. 21.elephant 22.bear 23.zebra 24.giraffe 25.backpack
7. 26.umbrella 27.handbag 28.tie 29.suitcase 30.frisbee

8. 31.skis 32.snowboard 33.sports ball 34.kite 35. baseball bat
9. 36.baseball glove 37.skateboard 38.surfboard 39.tennis racket 40.bottle

10. 41.wine glass 42.cup 43.fork 44.knife 45.spoon
11. 46.bowl 47.banana 48.apple 49.sandwich 50.orange

12. 51.broccoli 52.carrot 53.hot dog 54.pizza 55.donut
13. 56.cake 57.chair 58.couch 59.potted plant 60.bed

14. 61.dining table 62.toilet 63.tv 64.laptop 65.mouse
15. 66.remote 67.keyboard 68.cell phone 69.microwave 70.oven

16. 71.toaster 72.sink 73.refrigerator 74.book 75.clock
17. 76.vase 77.scissors 78.teddy bear 79.hair drier 80.toothbrush
18. }

```

The annotations object contains the following objects:

1. Segmentations : are regions of interest which are usually a list of polygon vertices around the object.
2. is_Crowd : specifies if there is a single object or a group of objects.
3. image_id : a specific image in the dataset.
4. category_id : a single category specified in the categories section.

```

1. "annotations": [
2.   {
3.     "segmentation": [[510.66,423.01,511.72,420.03,...,510.45,423.01]],
4.     "area": 702.1057499999998,
5.     "iscrowd": 0,
6.     "image_id": 289343,
7.     "bbox": [473.07,395.93,38.65,28.67],
8.     "category_id": 18,
9.     "id": 1768
10.  },
11.  ...
12.  {
13.    "segmentation": {
14.      "counts": [179,27,392,41,...,55,20],
15.      "size": [426,640]
16.    },
17.    "area": 220834,
18.    "iscrowd": 1,
19.    "image_id": 250282,
20.    "bbox": [0,34,639,388],
21.    "category_id": 1,
22.    "id": 900100250282
23.  ]

```

For training our captioning model we used the files `annotations_train2014.json` and `annotations_val2014.json` and it is only the `annotations` object as shown below. There is all the information needed for the captioning process. For each image there is an identifier called `"image_id"`, which is unique for every one of them in our dataset, an identifier `"id"` which is the identification number of the caption and the identifier `"caption"` which is the description of the image. The following is an example of the format of this file:

```
1. "annotations": [  
2.   {  
3.     "image_id": 289343,  
4.     "id": 433580,  
5.     "caption": "A person riding a very tall bike in the street."  
6.   },  
7.   ...  
8.   {  
9.     "image_id": 250282,  
10.    "id": 511309,  
11.    "caption": "A group of school children posing for a picture. "  
12.  },  
13. ]
```

6.2 MODEL TRAINING

In this section we will deal with the process of training our system, analyzing the exact algorithm used for training, but also the choice of hyperparameters we have done so to optimize this process.

6.2.1 Training Variables

Having described all the components of the system we implemented, it's time to see which of these pieces were parts of the whole system training. The challenges of designing and selecting the training variables we faced during the creation of our system were several and they had to do mainly with the problem of overfitting. Nevertheless, we applied techniques that were capable of dealing with this problem.

First of all, the most obvious way to deal with this problem was to initialize the weights of our Inception V3 Neural Network, with weights of an already trained model into quite large datasets, such as ImageNet, which is capable of identifying and categorizing objects that are in pictures. We applied it to all our examples and it helped quite a lot, mainly in the field of generalization. This means that the weights of Inception V3 were initialized with pre-trained weights and did not change at all during the training process. On the contrary, the weights of the fully connected layer after the inception model took part in the training process and so were all the parts of the decoder model coming after. More specifically, the attention mechanism needed to adjust its weights during the training in order to learn where are the most important features in the input images in relation to the expected output. Even though there are pre-trained models capable of representing millions words with fixed length vectors, such as `word2vec` or `glove`, we observed that the use of such a model did not show any improvement, and therefore for simplicity reasons, it was preferred not to include them in our system, but to train the weights of embedding layer normally. Finally, the weights of the two fully connected layers, which are at the output of the model and at the input of the Softmax classifier after LSTM, were variables that needed training, as it did, of course, the weights of LSTM network.

At this point we must mention that Mask-RCNN model was initialized with pre-trained weights as well but then it was fine-tuned using COCO dataset.

In summary, the weights that were trained were the following:

- Mask-RCNN model in order to be fine-tuned with our model.

- The weights of the fully connected layer, which is at its output Inception_V3 network and matches the dimensions of the table it contains image features from 2048 to 256.
- The weights of the attention model, which decide what part of the input image are the most important.
- The weights of the word embedding matrix in fixed length vectors 256, which converts words into real value vectors.
- The weights of the LSTM network, with size 512, which is responsible for making proposals descriptions of pictures.
- The weights of the fully connected layers, located between the LSTM and the Softmax classifier, which corresponds the 512 dimensions output of LSTM, firstly at size 512 and then at the size of the vocabulary.

6.2.2 Training Algorithm

The general purpose of our system is to train the LSTM network so that it can predict every word of a sentence describing a picture, having first seen the picture as well as all the previous words which are represented by the embedding vectors. Attention model has a very important role to this, because of its ability to focus on the most important parts of an image and as a result it can help the LSTM to make more accurate predictions. The input of the LSTM is a combination of the context vector produced by the attention model and a word. At each timestep a word from the training sentences enters the embedding layer and a new one is produced. In more detail the train process of our model given an input image I and the training caption $S = (S_0, \dots, S_N)$ is:

$$obj_i = Mask_RCNN(I), \quad i = 1, \dots, n, \text{ corresponding to } n \text{ different objects in an image} \quad (6.1)$$

$$A_i = CNN(obj_i), \quad (6.2)$$

$$z_t = ATT(A_i, h_{t-1}), \quad t \in \{0 \dots N - 1\} \quad (6.3)$$

$$x_t = W_e S_t, \quad (6.4)$$

$$o_t = [z_t; x_t], \quad (6.5)$$

$$p_{t+1}, h_t = LSTM(o_t) \quad (6.6)$$

The image I first enters the mask-RCNN where n objects are created. These objects are then passed through a CNN, in this thesis an Inception_V3 model, for feature extraction and the attention mechanism uses these features to produce the context vector z_t , where t is the timestep. Each word is represented with an embedding vector W_e . Then the whole model is trained to minimizing the cost function as follows

$$L(I, S) = - \sum_{t=1}^N \log p_t(S_t)$$

But let's look at the steps that are performed in a more detail.

Algorithm 6.1 Training of the total model

1. Given an input image I , pass it through Mask-RCNN and get the top 4 regions of interest.
2. Combine these 4 images with the total image and pass them through the Inception_V3 model. Get a feature vector for each of the 5 images.
3. Promote the Inception_V3 outputs to the fully connected layer in order to transform the dimensions of the images feature vector.
4. Repeat, until a complete sentence for the description of image I is produced:
 - a. If $t=0$:
Initialize the hidden state that passes to both LSTM and attention at zero
Else:
Use the previous hidden state
 - b. Pass the feature vectors as well as the hidden state of the LSTM into the attention model and create a context vector.
 - c. Pass one word through the embedding layer.
 - d. Concatenate the output of the attention and the embedding layer.
 - e. Use the new vector as input to the LSTM unit.
 - f. Pass the output of the LSTM through two fully connected layers and make a prediction for a word.
 - g. Calculate the value of the loss function $L(I, S)$ between the probability provided by the LSTM network and the actual word.
 - h. Through the technique of error backpropagation and with the help of the Adam Optimizer update the values of the trained variables, so they make a more accurate prediction next time.

6.2.3 Hyperparameter Selection

In order for our system to produce the best possible results, it should be trained in the best possible way. For this to happen, an important role in system training plays the right choice of hyperparameters. In order to come up with the specific choices we will present later, we had to try different values, search for values used in corresponding systems and have shown satisfactory results. Thus, the values of the hyperparameter we ended up using are:

- The number of images that will be processed simultaneously by our system (batch size) is equal to 64.
- As an optimizer of our training, we chose Adam.
- As already mentioned, the number of LSTM network units as 512
- The output dimensions of the embedding table were 256.
- The learning rate of our system in the first stage started at 0.001 and gradually reached almost zero using a decreasing function.
- Since our training data set consisted of 80000 images with their descriptions, the epochs in which they were trained our system was about 20.

Hyperparameter	Value
Learning rate	0.001
Batch size	64
Epochs	20
Dropout rate	0.3
Attention size	256
Embedding size	256
LSTM size	512

Table 6-1: The hyperparameters used in our model.

During the inference, there are several techniques that can be used to create captions. The first and most common technique is Sampling, where we chose the word with higher probability and then we input this word to the embedding layer at the next step and the process continues until we meet the <end> token or the proposal reaches a maximum length that we have set. The second technique is called Beam Search. In this method, we consider the set of k better propositions as candidates to create proposals and we only hold the k better. In our implementation we used the sampling method.

6.3 METRICS

To evaluate our system we used four of the most common metrics used for image captioning. Those metrics are BLEU [11], ROUGE-L[21] and METEOR [17] which were originally created for evaluating machine translation. We also used CIDEr [16] that is the only metric created specifically for image caption. In the next section we will analyse in detail how those metrics work.

6.3.1 BLEU

BLEU or Bilingual Evaluation Understudy[11] is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. First we count the number of matches between n-grams of candidate and reference sentences and then we divide by the total number of n-grams in the candidate sentences.

The precision score is p_n is then calculated:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram' \in C'} Count(n-gram')} \quad (6.1)$$

Then a penalty is computed for short candidate sentences:

$$BP = \begin{cases} 1, & \text{if } c > r \\ e^{(1-r/c)}, & \text{if } c \leq r \end{cases} \quad (6.2)$$

Where c is the length of the candidate and r is the length of the reference

The cumulative scores are produced by calculation individual n-gram scores at all orders from 1 to n and weighting them by calculating the weighted geometric mean.

$$BLEU = BP * \exp(\sum_{n=1}^N w_n \log p_n) \quad (6.3)$$

Where w_n are positive weights summing to one

6.3.2 ROUGE

ROUGE or Recall-Oriented Understudy for Gisting Evaluation [21] finds the common subsequences (LCS) between the reference and candidate sentences and calculates the recall and precision:

$$R_{lcs} = \frac{LCS(Cand, Ref)}{Referense\ length} \quad (6.4)$$

$$P_{lcs} = \frac{LCS(Cand, Ref)}{Candidate\ length} \quad (6.5)$$

Then uses this values to compute the term $\beta = P_{lcs}/R_{lcs}$ and the final score:

$$ROUGE - L = F_{\beta} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \quad (6.6)$$

6.3.3 METEOR

Metric for Evaluation of Translation with Explicit Ordering[17] is a metric for the evaluation of machine translation output. Given a pair of translations, METEOR creates an alignment between the two strings by mapping unigrams, such that every unigram in each string maps to zero or one unigram in the other string, and to no unigrams in the same string. Once a final alignment has been produced the Meteor score for this pairing is computed by calculate unigram precision $P = m/t$ and unigram recall $R = m/r$ where:

m: unigrams found between the reference and candidate

t: total number of unigrams in the candidate

r: total number of unigrams in the reference

We then compute the harmonic mean of P and R:

$$F_{mean} = \frac{10P*R}{R+9P} \quad (6.7)$$

METEOR, also, computes a penalty for a given alignment as follows:

$$Penalty = 0.5 * \left(\frac{\#chunks}{\#unigrams_mached} \right)^3 \quad (6.8)$$

Chunks are the unigrams in adjacent positions in the candidate that are also mapped to unigrams that are in adjacent positions in the reference. Finally, the METEOR Score is computed as follows:

$$Score = F_{mean} * (1 - Pealty) \quad (6.9)$$

6.3.4 CIDEr

Consensus-based Image Description Evaluation[16] is used to evaluate how well a candidate sentence c_i matches a descriptions $S_i = \{s_{i1}, \dots, s_{im}\}$ of an image. This metric is the only one that uses the stems of the words. It produces the score by calculating the TF-IDF score, calculating the frequency an n-gram occurs in the reference sentence and uses a penalty for n-grams that commonly occur across all images in the dataset.:

$$g_k(s_{ij}) = \frac{h_k(s_{ij})}{\sum_{\omega_l \in \Omega} h_l(s_{ij})} \log \left(\frac{|I|}{\sum_{l_p \in I} \min(1, \sum_q h_k(s_{pq}))} \right) \quad (6.10)$$

where :

Ω is the vocabulary of all n-grams

$h_k(s_{ij})$ a reference sentence s_{ij}

$h_k(c_i)$ a candidate sentence c_i

I the set of all images in the dataset.

Then we compute the average cosine similarity between the candidate sentence and the reference sentences:

$$CIDEr(c_i, S_i) = \frac{1}{m} \sum_j \frac{g^n(c_i)g^n(s_{ij})}{\|g^n(c_i)\| \|g^n(s_{ij})\|} \quad (6.11)$$

Where:

$g^n(c_i)$ is a vector corresponding to all n-grams of length n

$\|g^n(c_i)\|$ is the magnitude of the vector $g^n(c_i)$.

Similarly we compute for $g^n(s_{ij})$.

At the last step we combine the scores from n-grams as follows:

$$CIDEr(c_i, S_i) = \sum_{n=1}^N w_n CIDEr_n(c_i, S_i) \quad (6.12)$$

The authors of the paper propose to use uniform weights $w_n = 1/N$ as they produce the best results.

Chapter 7: Results

In the previous chapter we explained in detail how the model we created worked. In this chapter we will analyse the COCO dataset that it was used, at the first step, for training our system and then for evaluation using the metrics BLEU, METEOR, ROUGE-L and CIDEr, described in the next section. Our baseline model is the one proposed in [29]. In this approach the image is divided into 64 parts of the same size and shape. Attention is then split among these 64 parts and a caption is produced with help of an LSTM.

In the table below we present our results compared to the baseline model.

Metric	Our Model	Baseline
BLEU-1	65.9	51.8
BLEU-2	44.1	31.0
BLEU-3	29.1	17.9
BLEU-4	19.1	10.4
METEOR	24.5	11.3
CIDEr	54.6	34.5
ROUGE-L	51.0	44.6

Table 7-1: Evaluation Results

Our approach is an improvement compared to the baseline and we observe a significant increase at the value of each metric. As we can see from the table 6-2 the increase is 14.1 for BLEU-1, 13.1 for BLEU-2, 11.2 for BLEU-3, 8.7 for BLEU-4, 13.2 for METEOR, 20.1 for CIDEr and 6.4 for ROUGE-L.

We present the captions with the highest Bleu_4 score:



Score: 0.999

Caption: A baseball player swinging a bat at a ball.



Score: 0.999

Caption: A group of people standing around a table.



Score: 0.967

Caption: A man is playing tennis on a tennis court.



Score: 0.919

Caption: A fire truck driving down a street.



Score: 0.863

Caption: A herd of sheep grazing in a field.



Score: 0.84

Caption: A group of people standing on a beach.



Score: 0.819

Caption: A man riding a wave on a surfboard.



Score: 0.817

Caption: A group of people sitting at a table.



Score: 0.813

Caption: A group of zebras are standing in a field.



Score: 0.813

Caption: A clock tower with a clock on it.

7.1.1 Qualitative Examples and Discussion

So far, we evaluated our models using the automatic metrics, but the metrics are only approximate in judging the correctness of the captions. In the previous section we saw some sample captions generated by our model on images from the validation set. It is obvious that the generated captions are often fairly accurate, but still sometimes contain errors.

One of the errors which still persist is in counting, where the captions tend to get the number of objects wrong and in some cases it repeats the words referring to objects instead of using numerals. This can be seen in the Figure 7-1, where, even though the caption accurately describes the image, it produces two times the word ‘clock’ because it was two times located in the picture by the image detection model, or in the second picture where it understood that the picture contained a lot of wine glasses but it output two times the phrase ‘wine glasses’.



Caption: A clock with a clock in a building



Caption: A table topped with wine glasses of wine glasses.



Caption: A bed with a bed and a bed.

Figure 7-1 : Examples of how our model fails to get the number of objects in an image.

Another important error in the function of our model is that, even though it was trained to recognize almost 14300 words, the number of words in the vocabulary, during the testing process we calculated that it was able to produce only 837. This is a problem occurring in most cases of similar problem, even in the state of the art approaches, due to the fact that the model is removing secondary information during the processing of the image. This problem becomes apparent due to the small length of the predicted captions but also from the repeating of the same degraded n-gram distribution. Some examples are shown in the next figure. We can see that for pictures with similar content the model produces almost the same caption.



Caption: A baseball player is swinging a bat on a field.



Caption: A baseball player is holding a bat on a field.



Caption: A baseball player holding a bat at a baseball game.



Caption: A train is coming down the tracks.



Caption: A train on a track.



Caption: A train is going down the tracks.

Figure 7-2: Examples of how the small variety of words produced by our model affects the captions

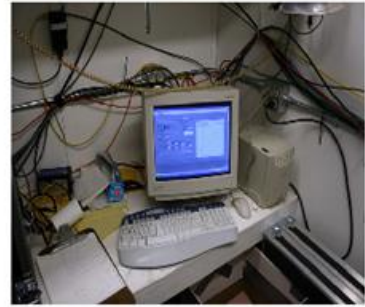
Of courses some of the errors occur because the model produces captions based on similar examples in the training set. For example, our model has trouble distinguishing between laptops and computers. This happens due to the fact that in the dataset there more pictures featuring laptops than computers. In Figure 6-3 we can see that our model either ignores completely the computer in the image or confuses it with a laptop.



Caption: A desk with a laptop.



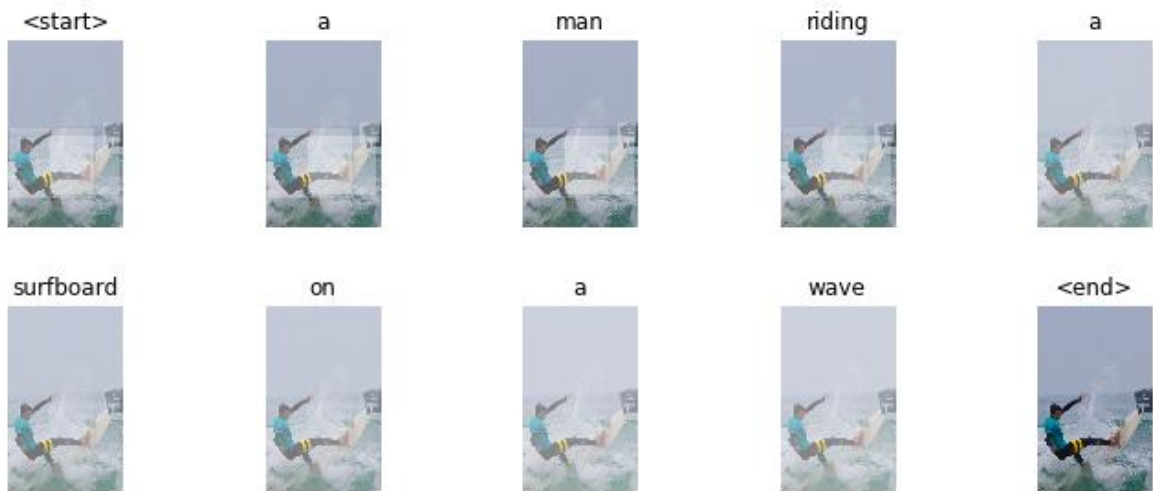
Caption: A desk with a laptop and a computer.



Caption: A laptop computer sitting on a desk.

Figure 7-3: An example of our model being biased.

Here we present in detail the output of our model, both the word produced in each timestep and the regions in the image where attention was given in order to produce that specific word. The following are some of the successful results:



<start>



a



vase



with



flowers



in



it



<end>



<start>



a



pizza



on



a



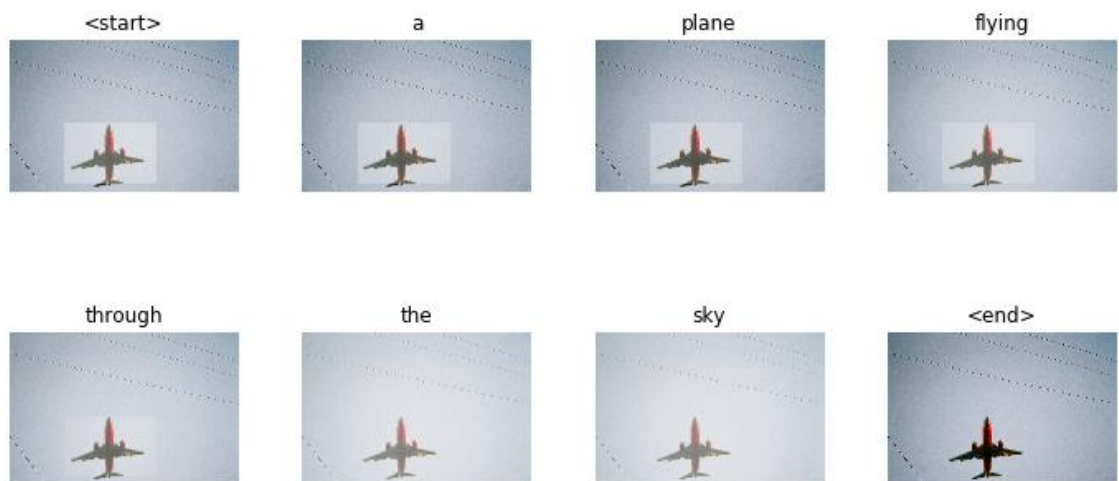
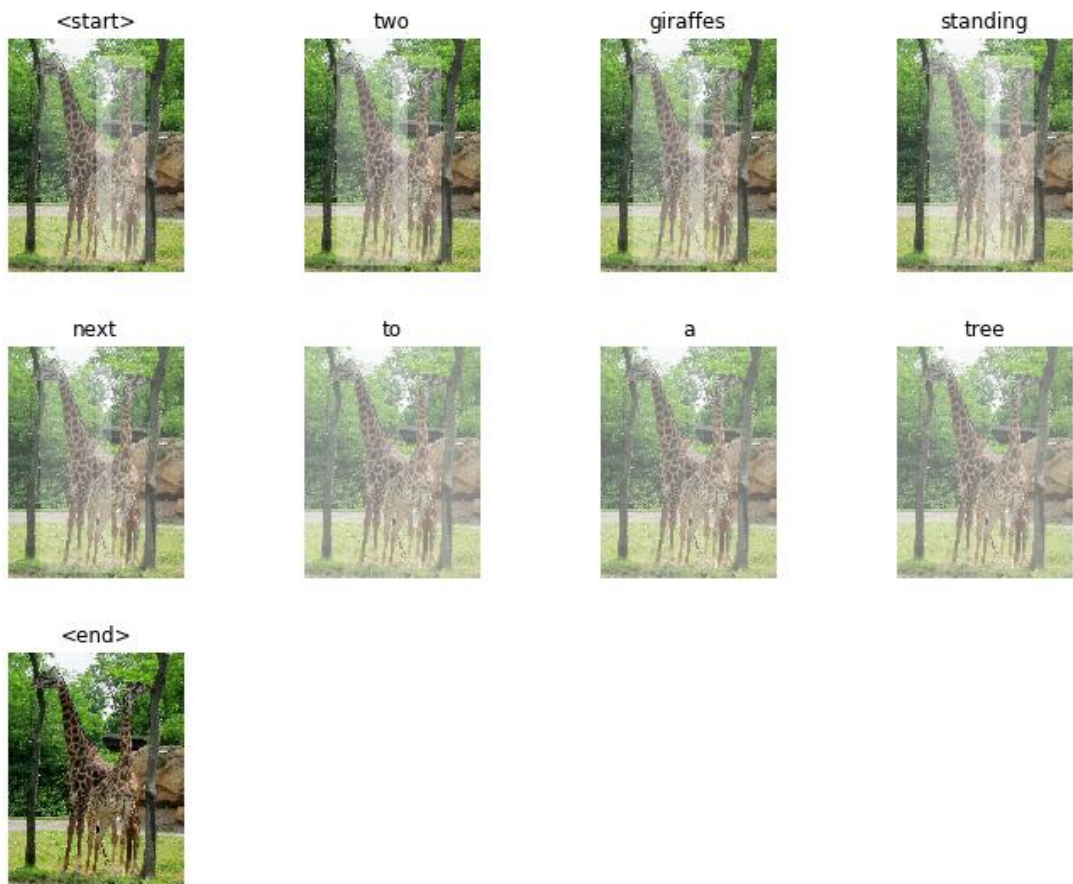
plate

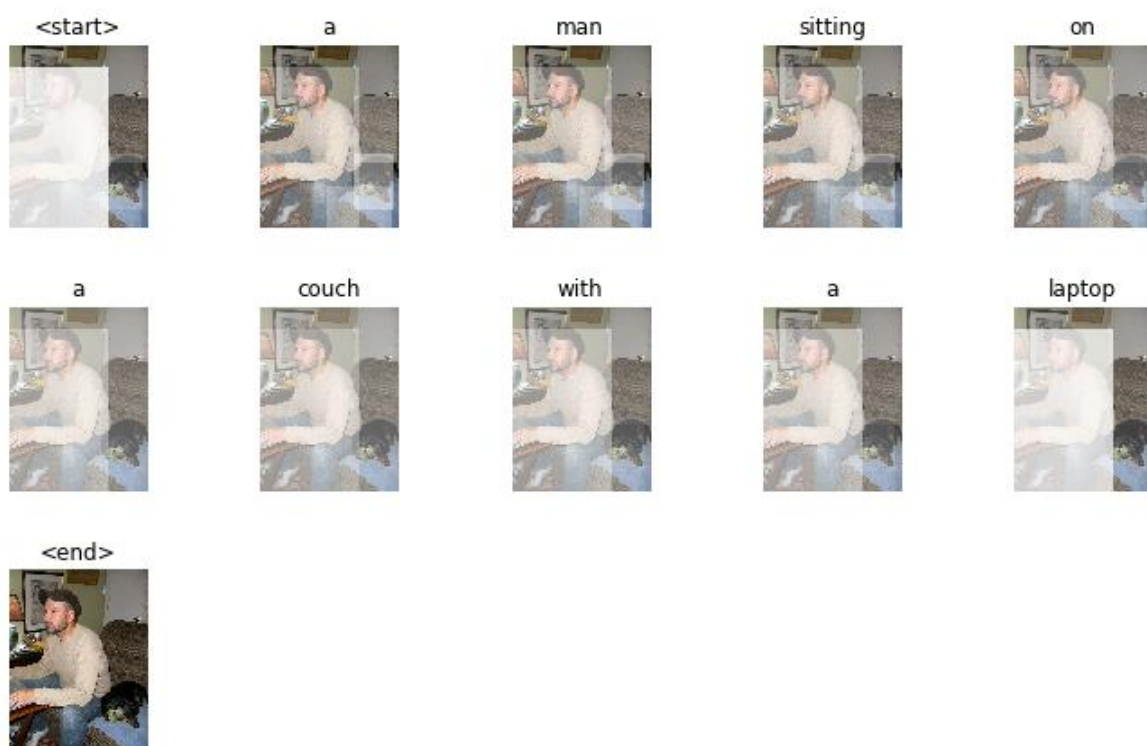
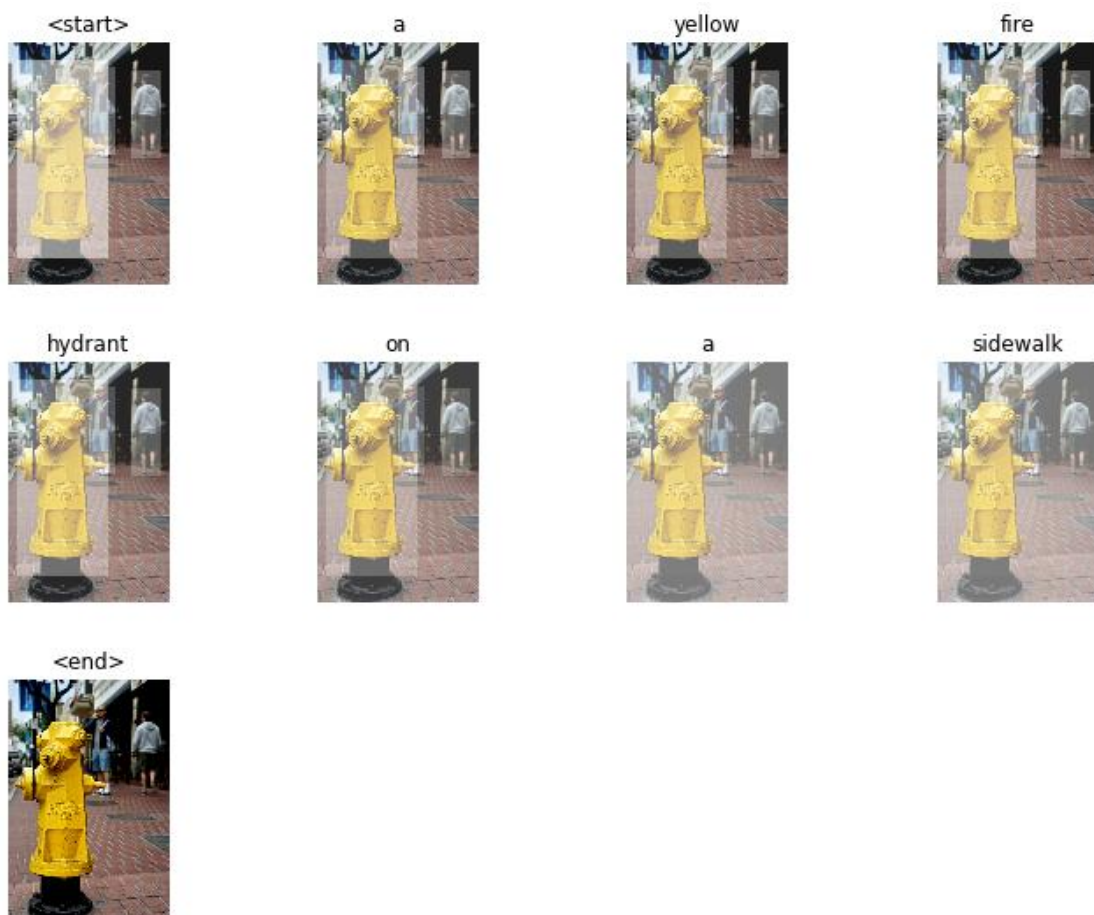


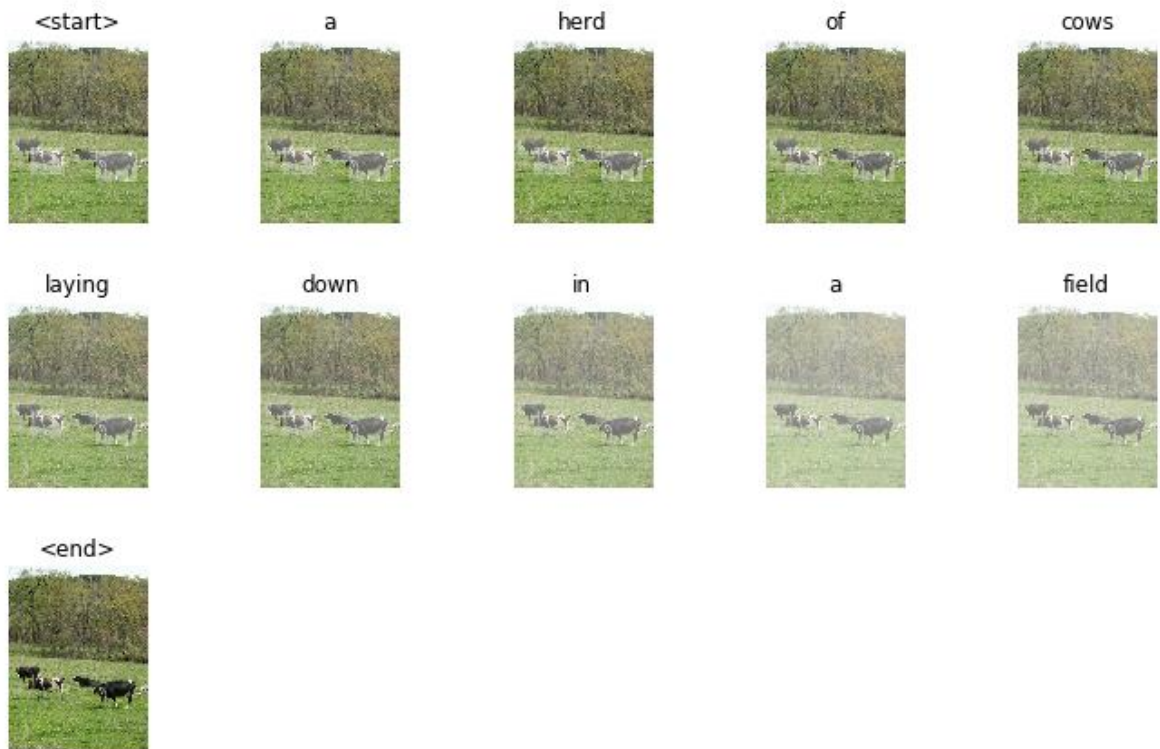
<end>



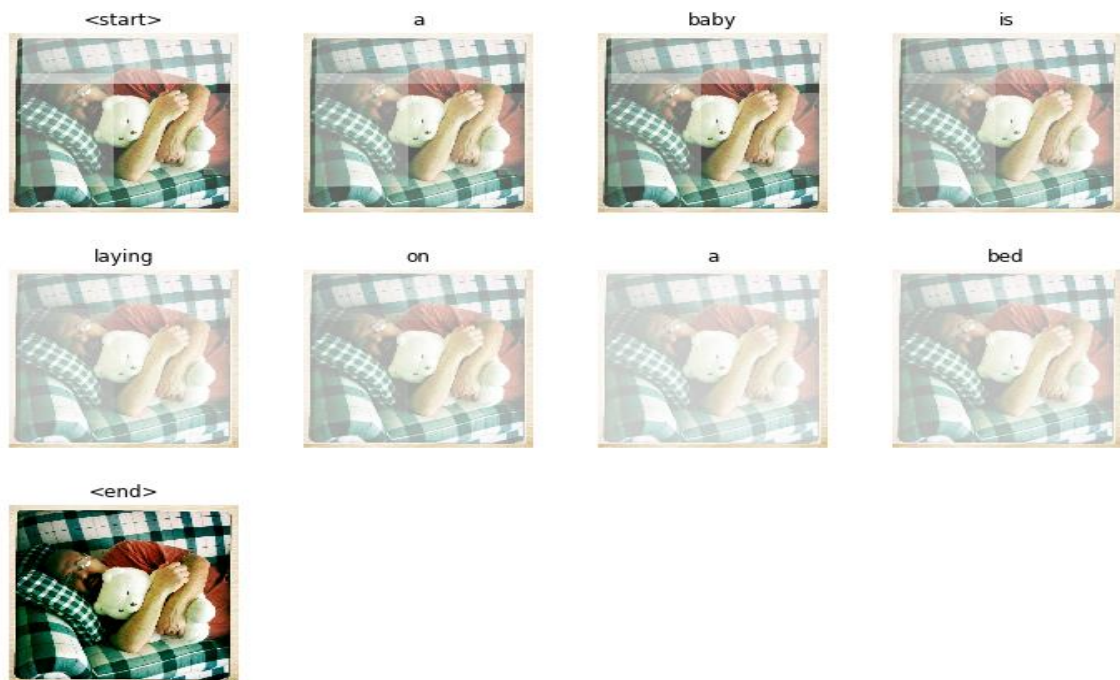








Of courses there where some images where the caption didn't accurately describe their content:



<start>



a



boat



sitting



on



a



wooden



dock



<end>



<start>



a



man



standing



next



to



a



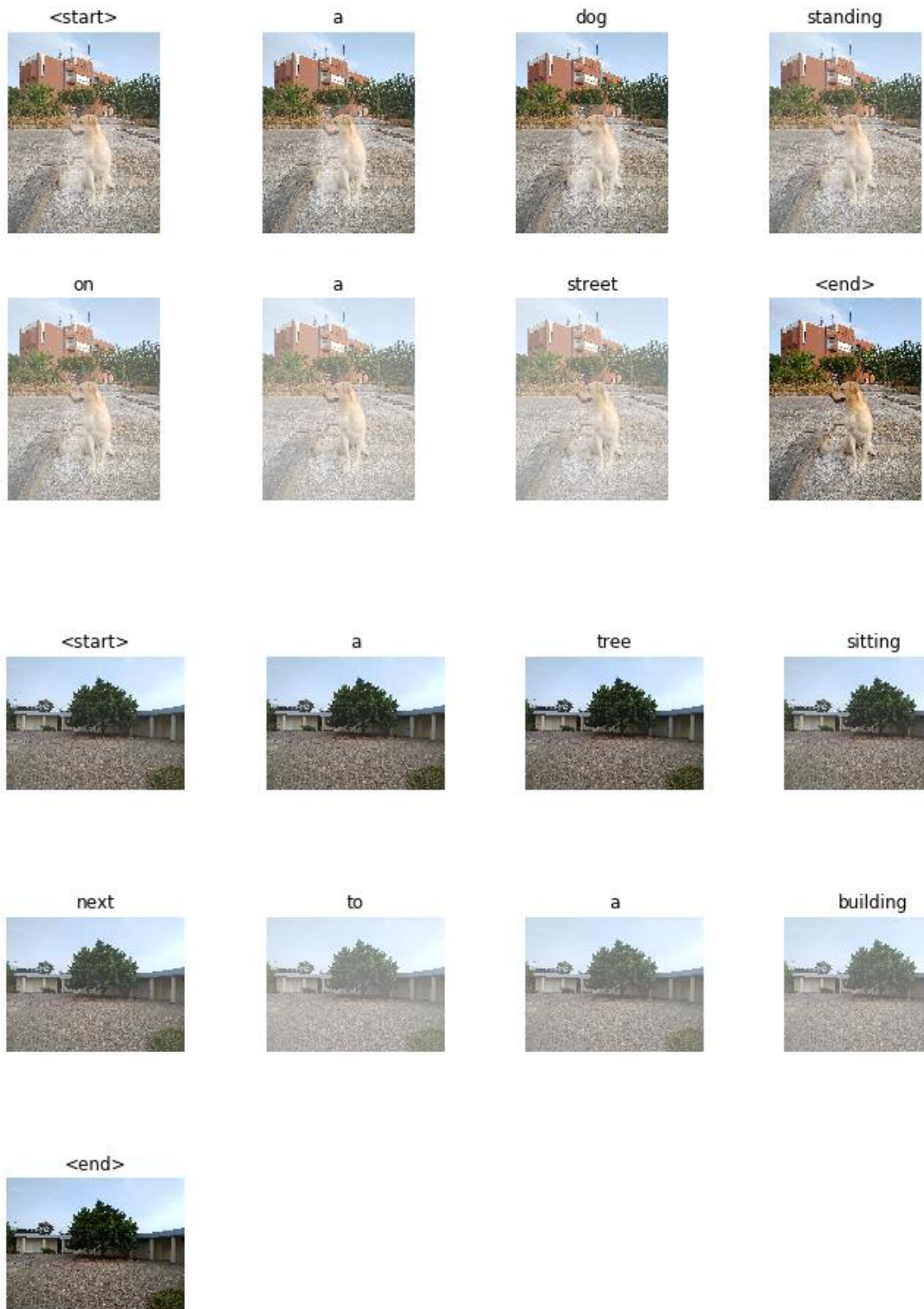
tree



<end>



Now we present the output of our model not from using COCO dataset, but from images taken by us inside the Technical University of Crete:



<start>



a



man



and



a



dog



standing



in



the



grass



<end>



<start>



a



woman



sitting



on



a



bench



<end>



<start>



a



man



standing



next



to



a



sign



<end>



Chapter 8: Epilogue

8.1 SUMMARY AND CONCLUSIONS

In this diploma thesis, we presented a system of neural networks which can automatically see an image and create a rational description of it in English language. Our system is based on a convolutional neural network which encodes the input image in a representation of a fixed-length vector, then followed by a recurrent neural network, responsible for creating the corresponding description of the image. The model was trained so as to maximize the likelihood of producing a rational description for a given image.

In Chapter 1 we discussed the source of inspiration behind neural networks, some of their wide variety of use, the characteristics that makes them very useful and efficient but also some of problems resulting from their use.

In Chapter 2 we studied in detail the structure and the function of both single and multilayer networks and how the training algorithms, like backpropagation can modify the network in a way that generalizes and makes reasonable prediction for every unknown input.

In Chapter 3 the thesis focuses on some specific networks like Recurrent Neural Networks and Convolutional Neural Networks and how they work in detail.

In Chapter 4 we discussed the evolution of object detection, image classification and image captioning in general and some of the pre-trained model we used and was the inspiration behind our system.

Chapter 5 is the most important of this thesis. We discussed the model we developed, the idea behind every layer we added and how are all these combined to achieve our goal.

In Chapter 6 we analyzed the training dataset and how it was used for the purpose of our model. We also described the training method and the final results during testing.

The purpose of this thesis was to study and implement some of the state of the art mechanisms developed in computer vision and Natural Language Processing. This includes models like Mask-RCNN and Inception_V3, which are state of the art in terms of object detection and image classification respectively, and mechanisms like attention which, as we concluded, gave a significant improvement to our results. Another very important aspect of our approach is the fact that attention is calculated at the level of objects and not in a uniform grid of equally-sized image regions as most similar approaches suggest [13].

8.2 FUTURE WORK

In any machine learning problem it is necessary to present the scope for improvement of our approach as well as future issues related to this particular problem. As it makes sense, there are many ways in which one can improve the results and one of them is training. When a network is as deep as Inception-v3, then we are given the opportunity to train it with more and more data while of course requiring more training iterations and therefore more time. So, simply by a large amount of data, as long as the network will be trained to more features, it will succeed and be accurate to random images.

Another key change that can lead to greater accuracy is the change in the CNN we are training. Inception-v3 is a state-of-the-art CNN, but other CNNs are published, either new or improved versions of Inception, such as Inception-v4 and Inception-ResNet. The comparison of these networks with Inception-v3 is presented in [18]. Also, the same can be applied to the object detection model we used. Other approaches like the YOLO [20] have very satisfying results and it could be an improvement after being trained with our dataset.

One idea that can be implemented is to produce expressions using the Bidirectional Recurrent Neural Networks (BRNN) according to [19] instead of using simple LSTM units. Bidirectional Recurrent

Neural Networks connect two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously.

Another method proven to improve the performance of an NLP problem is beam search algorithm. Instead of greedily choosing the most likely next step as the sequence is constructed, the beam search expands all possible next steps and keeps the k most likely, where k is a user-specified parameter and controls the number of beams or parallel searches through the sequence of probabilities.

Finally, it is very common in cases of transfer learning, for the pre-trained models to be fine-tuned. This refers to the process of training some of the layers of pre-trained models that were frozen during training, so that the model can adjust better to the specific problem.

References

- [1] Saad ALBAWI , Tareq Abed MOHAMMED, Saad AL-ZAWI Understanding of a Convolutional Neural Network, The International Conference on Engineering and Technology 2017, At Antalya, Turkey
- [2] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna Rethinking the Inception Architecture for Computer Vision 2015 arXiv:1512.00567v3 2015
- [3] <https://folk.idi.ntnu.no/keithd/classes/advai/lectures/backprop.pdf>
- [4] George Saon, Hagen Soltau, Ahmad Emami, Michael Picheny Unfolded Recurrent Neural Networks for Speech Recognition, IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598 2014
- [5] Tianyi Liu, Shuangfang Fang, Yuehui Zhao, Peng Wang, Jun Zhang Implementation of Training Convolutional Neural Networks University of Chinese Academy of Sciences, Beijing, China I
- [6] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton ImageNet Classification with Deep Convolutional Neural Networks
- [7] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner Gradient-Based Learning Applied to Document Recognition Proc. Of the IEEE, November
- [8] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick Facebook AI Research (FAIR) Mask R-CNN arXiv:1703.06870v3, 2018
- [9] Ilya Sutskever, Oriol Vinyals, Quoc V. Le Sequence to Sequence Learning with Neural Networks arXiv:1409.3215v3 2014
- [10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar Microsoft COCO: Common Objects in Context arXiv:1405.0312v3 2015
- [11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu IBM T. J. Watson Research Center Yorktown Heights BLEU: a Method for Automatic Evaluation of Machine Translation Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 311-318.
- [12] Karen Simonyan, Andrew Zisserman Visual Geometry Group, Department of Engineering Science, University of Oxford VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION arXiv:1409.1556v6 2015
- [13] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, Yoshua Bengio Show, Attend and Tell: Neural Image Caption Generation with Visual Attention arXiv:1502.03044v3 2016

- [14] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, Lei Zhang Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering arXiv:1707.07998v3 2018
- [15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks arXiv:1506.01497v3 2016
- [16] Ramakrishna Vedantam, C. Lawrence Zitnick, Devi Parikh CIDEr: Consensus-based Image Description Evaluation arXiv:1411.5726v2 2015
- [17] Satanjeev Banerjee, Alon Lavie METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments
- [18] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. «Inceptionv4, inception-resnet and the impact of residual connections on learning». arXiv:1602.07261 (2016).
- [19] Andrej Karpathy and Li Fei-Fei. «Deep visual-semantic alignments for generating image descriptions». In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
- [20] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi You Only Look Once: Unified, Real-Time Object Detection arXiv:1506.02640v5 2016
- [21] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. Text summarization branches out: Proceedings of the ACL-04 workshop.
- [22] Diederik P. Kingma and Jimmy Lei Ba A method for stochastic optimization. CoRR, arXiv:1412.6980v9 2017
- [23] John Duchi, Elad Hazan, Yoram Singer , Adaptive Subgradient Methods for Online Learning and Stochastic Optimization , Journal of Machine Learning Research 12 (2011)
- [24] Matthew D. Zeiler, ADADELTA: AN ADAPTIVE LEARNING RATE METHOD, arXiv:1212.5701v1 2012
- [25] Alex Graves Department of Computer Science University of Toronto, Generating Sequences With Recurrent Neural Networks arXiv:1308.0850v5 2014
- [26] https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_supervised_learning.htm
- [27] Alex Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network, arXiv:1808.03314v4 2018
- [28] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber LSTM: A Search Space Odyssey arXiv:1503.04069v2 2017
- [29] https://www.tensorflow.org/tutorials/text/image_captioning

- [30] <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>
- [31] <https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d>
- [32] https://www.researchgate.net/figure/5-Activation-functions-in-comparison-Red-curves-stand-for-respectively-sigmoid_fig10_317679065
- [33] https://commons.wikimedia.org/wiki/File:Single-Layer_Neural_Network-Vector-Blank.svg
- [34] <https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network-and-w>
- [35] <https://www.sciencedirect.com/science/article/pii/S088523081400093X#fig0010>
- [36] <https://hackernoon.com/gradient-descent-aynk-7cbe95a778da>
- [38] <https://medium.com/ai-journal/lstm-gru-recurrent-neural-networks-81fe2bcd1f9>
- [39] <https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN?focusedCommentId=25007595>
- [40] <https://skymind.ai/wiki/lstm>
- [41] <https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4>
- [42] https://www.researchgate.net/figure/A-three-dimensional-RGB-matrix-Each-layer-of-the-matrix-is-a-two-dimensional-matrix_fig6_267210444
- [43] <https://medium.com/@alittlepain833/simple-understanding-of-mask-rcnn-134b5b330e95>
- [44] <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-rcnn-and-tensorflow-7c761e238b46>
- [45] <https://modelzoo.co/model/mask-r-cnn-keras>
- [46] <https://sefiks.com/2017/12/10/transfer-learning-in-keras-using-inception-v3/>
- [47] <https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2>
- [48] <https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12>
- [49] <https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/ch04.html>
- [50] <https://lab.heuritech.com/attention-mechanism>
- [51] <https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7>

[52] http://uc-r.github.io/ann_fundamentals

[53] Nikolaos Panagiaris. "Natural Language Description of Images : A Qualitative Analysis.", Master's Thesis, Department of Informatics, University of Piraeus, October 2018.