AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND BIOMEDICAL ENGINEERING

DEPARTMENT OF APPLIED COMPUTER SCIENCE

Master of Science Thesis

Photo captioning with deep learning networks
Podpisywanie zdjęć za pomocą głębokich sieci neuronowych

Author: *Bartosz Tyński*
Degree programme: *Computer Science*
Supervisor: *PhD Adrian Horzyk*

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Contents

1. Introduction

The purpose of this project is to show the complete image captioning application programming interface. Image Captioning is a process of generating a textual description of an image by a computer. To achieve this, I will study the current state of the art, explain theoretical issues, and demonstrate a practical solution to an image captioning problem. Nowadays, the image is one of the dominant mediums. I believe that developing the domain of Image Captioning will bring researchers valuable information on artificial intelligence.

1.1. Motivation

Automatically describing the content of an image is a fundamental problem in artificial intelligence. Creating a textual description based on a given image is a quite complex activity. For us, humans, it is easy to do so because our perception allows us to catch essential elements in the picture, and based on them, construct the description. Though if we think about it more deeply, it turns out that our brain subconsciously performs a complex operation. If we would like to make a list of tasks for a computer to do so, it will look as follows:

1. Take a look at the picture.
2. Preprocess the picture.
3. Extract the most important features of the picture.
4. Convert visual features into words.
5. Based on received words create the correct description in terms of semantics and language.

As we can see, Image Captioning is quite an interdisciplinary task. It connects Computer Vision and Natural Language Processing. It is a hard task for the computer to make a sentence which is understandable for human. The primary challenge towards this goal is in the design of a model that is rich enough to simultaneously reason about the contents of images and their representation in the domain of natural language. Additionally, the model should be free of assumptions about specific hard-coded templates, rules, or categories. Instead, it should rely on rules learned from the training data.

Moreover, this project will help me to obtain skills not only in the deep learning domain but also as a software developer and software architect. I will have to make the most appropriate choice on my own in fields of:

- technology stack,
- project structure,
- pipeline for image captioning,
- suitable dataset,
- appropriate deep learning model and architecture,
- correct hyperparameters.

It is not so easy to prepare a solid project. There are many possible pitfalls during developing software that can consume our time in an unproductive and unsuccessful way. I wish to learn as much as possible to avoid them.

1.2. Problem statement

Being able to automatically describe the content of an image using properly formed English sentences is a very challenging task, but it could have a great impact, for instance, by helping visually impaired people better understand the content of images on the web. This task is significantly more difficult, for example, than the well-studied image classification or object recognition tasks, which have been a central focus in the computer vision community [1]. A description must capture not only the objects contained in an image, but it also must express how these objects relate to each other as well as their attributes and the activities they are involved in. Moreover, the above semantic knowledge has to be expressed in a natural language like English, which means that a language model is needed in addition to visual understanding.

The purpose of the work is to analyze the input image by a prepared algorithm. Based on the prepared model and the chosen hyperparameters, I will try to build the best and the most detailed description of photos. In this project, I am presenting a generative model based on a deep recurrent architecture empowered by attention. The presented solution combines recent advances in computer vision and machine translation, which can be used to generate natural sentences describing an image. Additionally, I aimed to design the solution, which is:

- Easy to use. That means the application programming interface should be clear and readable.
- Modular. The software should consist of parts that are easy to replace and integrate.
- Portable. Possibility of use trained models on other devices to obtain high-quality image caption.

1.3. Methodological approach

The algorithm will process the given image in a way to provide the reader with the basic textual information necessary to understand it. The scope of work includes:

- Obtaining and developing a set of learning data (input image and output caption).
- Preprocessing the dataset in such a way to be possible to be processed by a computer.
- Design of the architecture of the deep learning model. The model consists of two neural networks: a convolution network and a recursive neural network.
- Train prepared model with the training data.
- Verify if the predictions of the trained model. Validate the system to evaluate the trained model.

1.4. Structure of the work

The next Chapter 2 is dedicated to the history and current trends in Image Captioning. It will allow us to study and reflect on achievements in the field of describing images. The different existing approaches will be analyzed and compared to each other. In the end, a summary of existing methodologies will be presented. I believe preparing Chapter 2 will be a precious experience and will allow me to build a successful project. After becoming acquainted with the history and current state of the art, Chapter 3 will cover every aspect of my methodology and how I intend to solve the problem of image captioning based on thoughts from Chapter 2. Chapter 4 will present the suggested solution to the image captioning problem. It will show every step of the suggested image captioning pipeline. This chapter is a presentation of my practical solution to a given problem. There will be a place for verification and validation of the proposed system. Right after achieving the results, should the analysis and reflection come, Chapter 5 is the place to do so. It will cover the thoughts and conclusions about the obtained results. Additionally, this chapter will include the discussion of open issues ensuing during work. Last but not least is a summary of the work done. Chapter 6 describes in a big picture of the whole project. It covers the author's thoughts and plans for future work.

2. State of the art

Image captioning research has been around for a number of years, but the efficacy of techniques was limited, and they generally were not robust enough to handle the real world. Largely due to the limits of heuristics or approximations for word-object relationships [2] and [3]. However, in 2014 a number of high-profile AI labs began to release new approaches leveraging deep learning to improve performance. One of the first papers to apply neural networks to the image captioning problem was [4], who proposed a multi-layer perceptron that uses a group of word representation vectors biased by features from the image, meaning the image itself conditioned the linguistic output. From that moment, the researchers have started thinking about and improving the textual output given by the network. I believe that it is quite fascinating for researchers, that computer can do such sophisticated task as describing a photo.

2.1. Literature studies and analysis

Researchers start to think that it is possible to squeeze more information, which is easier to interpret, out of Convolutional Neural Network. Looking closer at how humans would complete the task, they would notice the important objects, parts, and semantics of an image and relate them within the global context of the image. All before attempting to put words into a coherent sentence. Similarly, instead of 'just' using the encoded vector representation of the image, we can achieve better results by combining information contained in several regions of the image. One can make use of two independent networks, one for text and one for image regions, that create a representation within the same image-text space. An example of such an approach is seen in the work of Karpathy, and Fei-Fei [5]. "Deep Visual-Semantic Alignments for Generating Image Description" [5], which utilizes convolutional neural network and recurrent neural network for caption generation, is one of the most responsible for popularising image captioning in the media. A large proportion of articles on image captioning tend to borrow from their excellent captioned image examples. But more impressive than capturing the public's attention with their research, were the strides made by Johnson, Karpathy, and Fei-Fei later that year [6].

The authors of work "Densecap: Fully convolutional localization networks for dense captioning" [6] asked the question, why are we describing an image with a single caption, when we can use the diversity of the captions in each region of interest to generate multiple captions with better descriptions than an individual image caption provides? The authors introduce a variation to the image captioning task called dense captioning where the model describes individual parts of the image (denoted by bounding boxes)

Fig. 2.1 presents an example of their idea. This approach produces results that may be more relevant and accurate when contrasted with captioning an entire image with a single sentence.

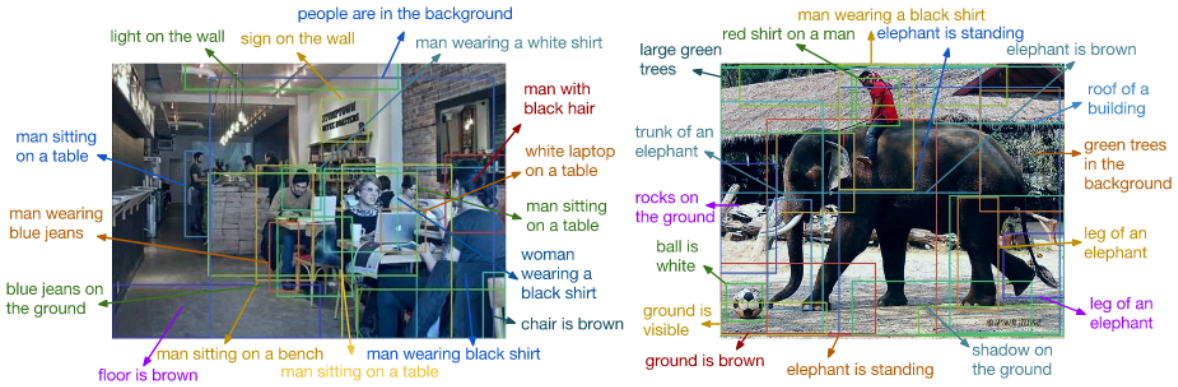

Fig. 2.1. Example captions generated and localized by models on test images proposed in [6].

One of the most prominent papers is "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" [7]. This work introduces the concept of attention to image captioning. The work takes inspiration from attention's application in other sequences and image recognition problems. Building on seminal work from [4] and [8], which incorporated the first neural networks into image captioning approaches, the impressive research team implemented hard and soft attention for the first time in image captioning (Fig. 2.2).

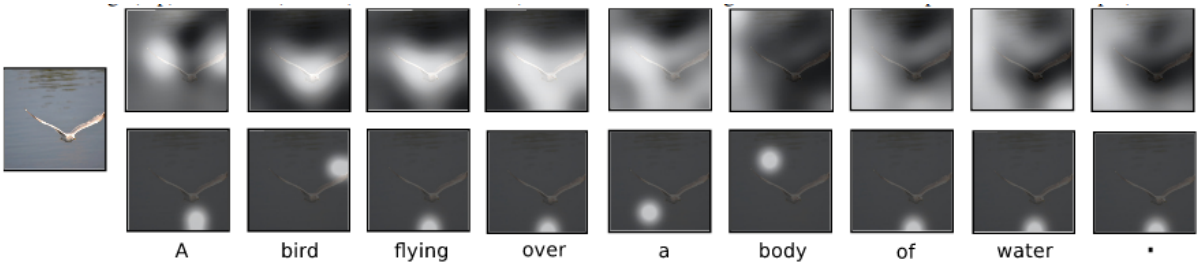

Fig. 2.2. Visualization of the attention for each generated word. The rough visualizations obtained by up-sampling the attention weights and smoothing. (top)"soft" and (bottom) "hard" attention [7].

Paper [9] note that traditional approaches to image captioning are either *top-down*, moving from a gist of an image that is converted to words, or *bottom-up*, which generate words describing various aspects of an image and then combine them. However, their contribution is the introduction of a novel algorithm that combines both of the aforementioned approaches and learns to attend selectively. This is achieved through a model of semantic attention, which combines semantic concepts and the feature representation of the image. Semantic attention Fig. 2.3 refers to the technique of focusing on semantically meaningful concepts, i.e. objects or actions which are integral to constructing an accurate image caption.

In spatial attention, the focus is placed on regions of interest, but semantic attention relates attention to the keywords used in the caption as it's generated.

Fig. 2.3. Qualitative analysis of the impact of visual attributes. The left six examples (solid green box) shows that the visual attributes help generate more accurate captions. The right two examples (red dashed box) indicate that incorrect visual attributes may mislead the model [9].

A very interesting and informative example is the history of Google Brain Team [10]. In 2014 their system used the Inception V1 image classification model to initialize the image encoder, which produces the encoding that is useful for recognizing different objects in the images [11]. This was the best image model available at the time, achieving 89.6% top-5 accuracy on the benchmark ImageNet 2012 image classification task[1]. They replaced this in 2015 with the newer Inception V2 image classification model, which achieves 91.8% accuracy on the same task [12]. The improved vision component gave their captioning system an accuracy boost of 2 points in the BLEU-4 metric (which is commonly used in machine translation to evaluate the quality of generated sentences) and was an essential factor of its success in the captioning challenge. Today's code release initializes the image encoder using the Inception V3 model [13], which achieves 93.9% accuracy on the ImageNet classification task. Initializing the image encoder with a better vision model gives the image captioning system a better ability to recognize different objects in the images, allowing it to generate more detailed and accurate descriptions. This provides an additional 2 points of improvement in the BLEU-4 metric over the system used in the captioning challenge. To conclude, the quality of features extracted from the image is crucial. It has a great impact on overall accuracy.

Another key improvement to the vision component comes from fine-tuning the image model. This step addresses the problem that the image encoder is initialized by a model trained to classify objects in images, whereas the goal of the captioning system is to describe the objects in images using the encoding produced by the image model. For example, an image classification model will tell you that a dog, grass, and a Frisbee are in the image, but a natural description should also tell you the color of the grass and how the dog relates to the Frisbee. In the fine-tuning phase, the captioning system is improved

[1] http://image-net.org/challenges/LSVRC/2012/index

by jointly training its vision and language components on human-generated captions. This allows the captioning system to transfer information from the image that is specifically useful for generating descriptive captions, but which was not necessary for classifying objects. In particular, after fine-tuning, it becomes better at correctly describing the colors of objects. Importantly, the fine-tuning phase must occur after the language component has already learned to generate captions - otherwise, the noisiness of the randomly initialized language component causes irreversible corruption to the vision component [1]. The Google Brain Team boasts of developing the ability to generate accurate new captions when presented with entirely new scenes, indicating a deeper understanding of the objects and context in the images (Fig. 2.4).

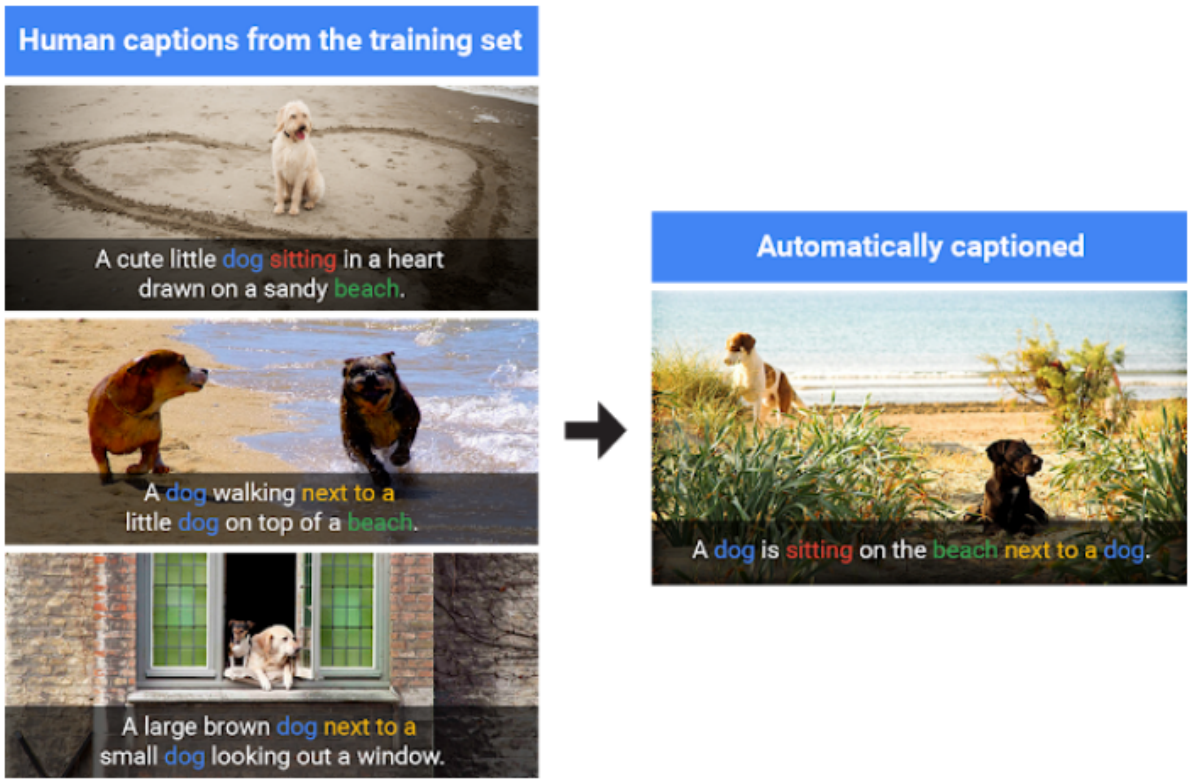

Fig. 2.4. Google Brain Team's model generates an entirely new caption using concepts learned from similar scenes in the training set [10].

One more equally important factor is sentence representation. Authors of this work [5] established the inter-modal relationships, to represent the words in the sentence in the same *h-dimensional embedding space* that the image regions occupy. The most straightforward approach might be to project every individual word directly into this embedding. However, this approach does not consider any ordering and word context information in the sentence. An extension of this idea is to use word bigrams or dependency tree relations. However, this still imposes an arbitrary maximum size of the context window and requires the use of Dependency Tree Parsers that might be trained on unrelated text corpora. To address these concerns, authors propose to use a Bidirectional Recurrent Neural Network to compute the word representations.

Due to the fact that Image Captioning is quite challenging tasks, many competitions were created. Their aim is to push current sate of the are to the highest possible level. One of the most prestigious is the COCO 2015 Image Captioning Task[2]. The COCO Captioning Challenge is designed to spur the development of algorithms producing image captions that are informative and accurate. The COCO leader board[3] is a great source when looking for an idea for your own system. For example, the TencentVision team is leading the COCO captioning leader board. Unfortunately, it is hard to find a publication detailing their work. All can be found is that they use a multi-agent Reinforcement Learning model. This is quite surprising fact comparing to the rest of the works in the Image Captioning field.

2.2. Comparison and summary of existing approaches

A computer does not know our language and its rules. That is why researchers are using Deep Learning. The computer has to be trained to be a good descriptor. It has to learn the way we speak. As we can see, there is no only one suitable approach to the captioning task. There are a variety of approaches for preparing a model architecture. People are choosing different architectures for extracting and decoding features. I believe that it only depends on our imagination how we will deal with this problem. Some are using Deep Learning others trying with Reinforcement Learning. They are both right and score high! It is amazing that one task can be solved in such a wide spectrum of ways!

Scores of the current state of the art solutions are very close to each other, which means that the authors are very near to the optimal solution. Image descriptions given by their systems are excellent, and it is difficult to distinguish whether they were created by a computer or by man. Although success has been achieved in recent years, there is still a large scope for improvement. Generation based methods can generate novel captions for every image. However, these methods fail to detect prominent objects and attributes and their relationships to some extent in generating accurate and multiple captions. In addition to this, the accuracy of the generated captions largely depends on syntactically correct and diverse captions, which in turn rely on powerful and sophisticated language generation models.

[2] *http://cocodataset.org/#captions-2015*

[3] *http://cocodataset.org/#captions-leaderboard*

3. Methodology

After in-depth studies in Chapter 2 I decided to use commonly used approach encoder-decoder to image captioning process. This concept is based on two deep learning architectures. First, the Convolutional Neural Network is used to detect objects on an input image. Then detected objects correspondences are treated as input data for a second, Recurrent Neural Network model that learns to generate the image description (Fig. 3.1). The task of this chapter is to lead the reader into the theory and concept of the image captioning process. Here will be given the definitions of used approaches, used models, data models, and analysis methods.

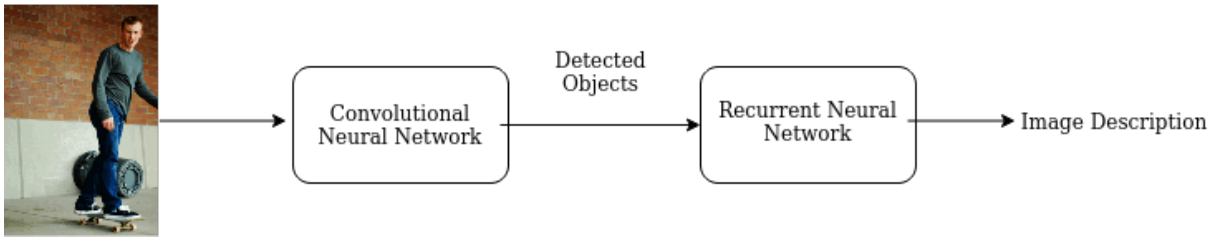

Fig. 3.1. Overview of the encoder-decoder approach to image captioning.

3.1. Used concepts

This section is dedicated to the introduction and description of the concepts used during the image captioning system preparation. The definitions of terms will be introduced, explanation of specific solutions and their application in the project will be presented. The purpose of this section is to familiarize the reader with the project concepts so that he can understand the proposed solution.

3.1.1. Convolutional Neural Network (CNN)

Firstly it is worth mentioning why CNN is better than simply a Neural Network model in case of processing images. The classical Neural Network approach receives an input (a single vector) and transforms it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully-connected to all neurons in the previous layer, and where neurons in a single layer function entirely independently and do not share any connections. The last fully-connected layer is called the 'output layer', and in classification settings, it represents the class scores.

Regular Neural Nets don't scale well to full images. Let's consider that the chosen input image size is 299x299x3 (299 wide, 299 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have $299*299*3 = 268203$ weights. That is plenty of data, and this is only the first layer! As we can see the fully-connected structure does not scale well, the number of neurons increases exponentially relative to the dimensions of the image. Moreover, we would almost certainly want to have several such layers of neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful, and the huge number of parameters would quickly lead to overfitting.

The Convolutional Neural Networks take advantage of the fact that the input consists of the images, and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. Worth mentioning is the fact that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network. The intuition behind CNN is simple. There exist three major parts (layers) of it:

– Convolutional layer. It will compute the output of neurons that are connected to the local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

– Activation layer. It will apply an elementwise activation function (e.g. RELU).

– Pooling Layer. It will perform a down-sampling operation along the spatial dimensions: width and height.

– Fully-connected layer. It is a kind of encoder for results from the convolutional layer. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

The given elements can be stacked in different combinations. Fig. 3.2 proposes the most simple approach to the CNN architecture model, which contains only one hidden layer: convolutional layer + activation layer + pooling layer. The fully-connected layer is usually the last layer; its task is to produce the desired output.

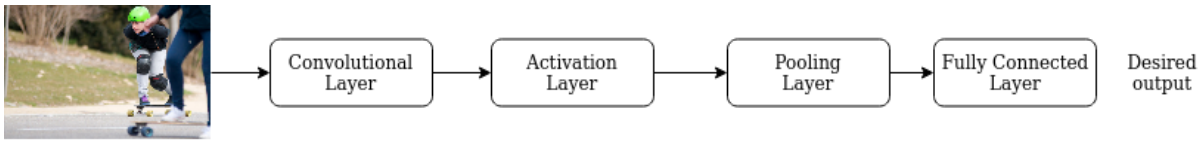

Fig. 3.2. Example of the simple architecture CNN with one hidden layer.

3.1.2. Recurrent Neural Network (RNN)

The Recurrent Neural Networks [14] are a family of neural networks for processing sequential data. Much as a convolutional network is a neural network that is specialized for processing a grid of values

X such as an image, a recurrent neural network is a neural network that is specialized for processing a sequence of values $x^1, .., x^n$. Just as convolutional networks can scale to images with large width and height, and some convolutional networks can process images of variable size, recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialization. Most recurrent networks can also process sequences of variable length. The RNN introduces the idea of sharing parameters across the different parts of a model. Parameter sharing makes it possible to extend and apply the model to the examples of different forms or lengths and generalize across them. If we had separate parameters for each value of the time index, we could not generalize to the sequence lengths not seen during training, nor share statistical strength across the different sequence lengths and across the different positions in time. Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence. For example, consider the two sentences, 'I went to Nepal in 2009' and 'In 2009, I went to Nepal.' If we ask a machine learning model to read each sentence and extract the year in which the narrator went to Nepal, we would like it to recognize the year 2009 as the relevant piece of information, whether it appears in the sixth word or in the second word of the sentence. Suppose that we trained a feedforward network that processes the sentences of a fixed length. A traditional, fully-connected feedforward network would have separate parameters for each input feature, so it would need to learn all the rules of the language separately at each position in the sentence. By comparison, a recurrent neural network shares the same weights across several time steps [15].

There exist the various types of ways to process the input sequence, here are given the common design patterns of them: (Fig. 3.3)

– One to one. The vanilla model of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).

– One to many. Fixed-sized input to sequence output (e.g. an image captioning system takes an image and outputs a sentence of words).

– Mane to one. Sequence input to fixed-size output (e.g. a sentiment analysis where a given sentence is classified as expressing a positive or negative sentiment),

– Many to many. Sequence input and sequence output (e.g. Machine Translation, an RNN reads a sentence in English and then outputs a sentence in French),

– Many to many. Synced sequence input and output (e.g. a video classification where we wish to label each frame of the video)

The sequence regime of operation is much more powerful compared to fixed networks and is more appealing for those of us who aspire to build more intelligent systems. Moreover, RNN combines the input vector with its state vector with a fixed (but learned) function to produce a new state vector. This can in programming terms be interpreted as running a fixed program with certain inputs and some internal

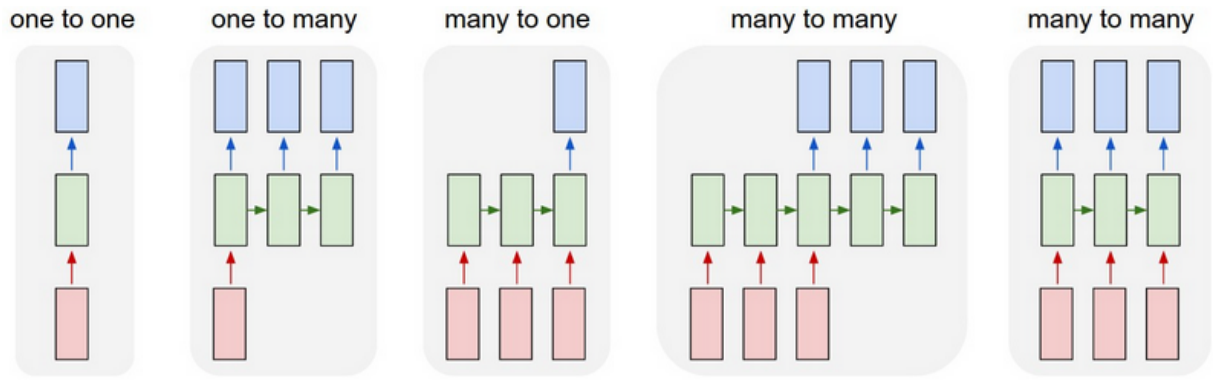

Fig. 3.3. The most common RNN design patterns. [16]

variables. Viewed this way, RNN essentially describes programs. In fact, it is known that RNN is Turing-Complete [17] in the sense that they can simulate arbitrary programs (with proper weights).

3.1.3. Long Short-Term Memory (LSTM)

The most effective sequence models used in practical applications are called gated Recurrent Neural Networks (RNNs). These include the Long Short-Term Memory and networks based on the Gated Recurrent Unit (GRU) [15]. Like leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode. Leaky units did this with connection weights that were either manually chosen constants or were parameters. Gated RNNs generalize this to connection weights that may change at each time step. Leaky units allow the network to accumulate information (such as evidence for a particular feature or category) over a long duration. Once that information has been used, however, it might be useful for the neural network to forget the old state. For example, if a sequence is made of subsequences and we want a leaky unit to accumulate evidence inside each subsubsequence, we need a mechanism to forget the old state by setting it to zero. Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it.

3.1.4. Feature extraction

A feature is an individual measurable property or characteristic of a phenomenon being observed [18]. The features are crucial for an effective machine-learning algorithm. However, the image itself does not have the features, which are informative enough. Therefore, the loaded image has to be handled in such a way as to receive the characteristics of the image. This step is called feature extraction. To do so, the desired portions or shapes of a digitized image (features) have to be detected and isolated. This process is classified by a computer vision as an object detection task. Most works described in Chapter 2 use the Convolutional Neural Networks for a feature extraction task. The way the CNN works fits perfectly in an object detection task, a detected object becomes features (Fig. 3.4). I decided to utilize the pre-trained InceptionV3 model for feature extraction [13], as the quality of the features is one of the most important factors influencing the accuracy of the deep learning model. I find the informativeness of

input data as one of the most important factors for my work pipeline, thus it is not worth risking using models prepared by myself for an object detection task.

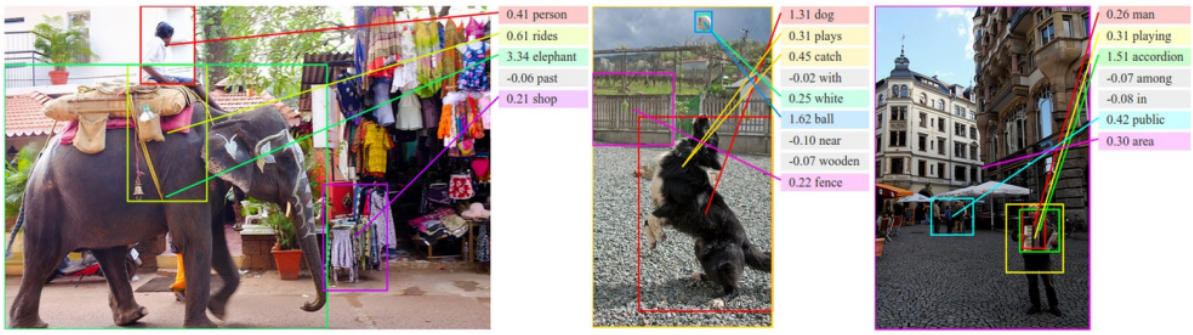

Fig. 3.4. Example of features extraction from given images. [9]

3.1.5. Attention

Incorporating attention allows the decoder to focus on specific parts of the input representation for each of the outputted words. Meaning, that in converting aspects of the image to captions, the network can choose where and when to focus in relation to specific words outputted during sentence generation. Such techniques improve not only the network performance but also aid interpretability; we have a better understanding of how the network determined its answer. The attention can enable our inspection and debug of the networks. It can provide functional insights, i.e. which parts of the image the network is 'looking at'. Each form of attention has its own unique characteristics.

There are multiple ways to implement the attention. The work [7] divide the image into a grid of regions after the CNN feature extraction, and produce one feature vector for each. These features are used in different ways for soft and hard attention:

– In the soft attention variant, each region's feature vector receives a weight (can be interpreted as the probability of focusing at that particular location) at every time step of the decoding RNN, which signifies the relative importance of that region in order to generate the next word. The neural network followed by a softmax, which is used to calculate these weights, is a deterministic part of the computational graph and therefore, can be trained end-to-end as a part of the whole system using backpropagation as usual.

– With hard attention only a single region is sampled from the feature vectors at every time step to generate the output word (using probabilities calculated similarly as mentioned before). This prevents network training by backpropagation due to the stochasticity of sampling.

This work will utilize the soft attention variant proposed in paper: "Neural machine translation by jointly learning to align and translate."[19].

3.1.6. Training

Every machine learning solution, especially the deep learning model has to be trained. At the very beginning without the proper training, the deep learning model in most cases is inaccurate and highly biased. However, its true power lies in a learning capability. If hyperparameters were picked wisely and the model was properly calibrated, the solution could achieve very satisfying results. Therefore, the training set should be prepared. This set will be iterated many times by a computer, as many as a number of epochs are set (number of epochs is one of the mentioned hyperparameters, which has to be picked by the researcher). The present project is utilizing supervised learning. The supervised learning task is to find the model that maps an input to an output based on example input-output pairs. In other words, during the training process, the model will see the true captions. Why are we letting know computer the desired output? Because based on the model prediction and the true caption, the loss can be computed. The loss is a result of the loss function and is a crucial element in a model optimization part. To conclude, the training set is well known for the model and iterated many times, both input images and output captions. Its role is to be the basis of model teaching. Without training, the model will be useless and ineffective. However, during the training process, the researcher has to be careful, because the model should score high not only on the training set but also on new not known input data. This means that under no circumstances should the model be overfitted to training data.

3.1.7. Loss Function

The main objective of the loss function is to deliver information on how the prediction differs from the real description. The value computed from loss function is used by the optimizer for founding the most optimal solution. It tells the optimizer how good is our model at predicting, and thus how it will be able to steer itself in the right direction. As the optimizer, I decided to pick Adam [20], an algorithm for first-order gradient-based optimization of stochastic objective functions based on adaptive estimates of lower-order moments.

3.2. Model workflow

An encoder-decoder model workflow is simple. Feed the image into a Convolutional Neural Network (CNN) for encoding, and run this encoding into a decoder Recurrent Neural Network (RNN) to generate an output sentence. The network backpropagates based on the error of the output sentence compared with the ground truth sentence calculated by a loss function like cross-entropy/maximum likelihood. Finally, one can use a sentence similarity evaluation metric to evaluate the algorithm.

Based on studies in Chapter 2.1, most Image Captioning systems are utilizing two neural networks in an encoder-decoder model architecture. First is the Convolutional Neural Network, which plays the role of the encoder; its job is feature extraction and encoding. The next is the Recurrent Neural Network - decoder. The decoder is creating textual descriptions based on encoded features. The common practice

is to extend the RNN decoder with LSTM. LSTM is a kind of RNN, but in most cases, it works better in natural language processing tasks [21].

3.3. Technology Stack

The entire project was written using Python, therefore all issues and formalism are connected with this specific technology. I decided to pick Python for this project because it is a very popular interpreted, high-level, general-purpose programming language. Python is one of the most frequently picked technology for machine learning projects [22]. Due to the fact that deep learning is a very complicated and sophisticated field I have decided to back my project with open-source python library - TensorFlow. TensorFlow makes it easy for beginners and experts to create machine learning models. It has great tutorials[1], which allow one to quickly dive into the deep learning state of the art.

To sum up. Both technologies are very popular; there are a lot of people working with them. Python is very friendly in the case of prototyping and Tensorflow is a very famous deep learning framework. These factors have a great impact on efficient software development. In case of any problems, I can refer to open issues for a specific technology. I think, that picking a technology stack with strong community backup is very reasonable, and will reduce the number of problems I have to face.

During work, the GNU/LINUX operating system was utilized, so all practical examples and formalism connected with the operating system are connected with it. Moreover, when referring to paths, I will be using the POSIX notation. In order to speed up and automate some processes commanding and operating shell Bash language was used; shell scripting was utilized for project maintenance and data management.

3.4. Software design methods

I decided to develop this project application programming interface in a Python package fashion. This means that it is enough to acquire and import the `image_captioning` package prepared by me, to enjoy fully functional the image captioning system. The package structure was created according to official Python documentation[2] and looked as follows:

```
image_captioning
    models
        __init.py
        endcoder.py
        decoder.py
        attention.py
    data
        __init__.py
```

[1] *https://www.tensorflow.org/tutorials*

[2] *https://docs.python.org/3/tutorial/modules.html#packages*

```
    data_processing.py
__init__.py
model_manager.py
```

3.5. Deep learning design methods

Additionally, to traditional software design methods, a typical deep learning project extends of a few new concepts. To prepare a proper deep learning model, it is crucial to follow a few design patterns. The most important is a dataset split. The rule of thumb in terms of splitting the data is to make a train, validation, and test set. Their ratio looks as follows:

- train set = 60%
- validation set = 20%
- test set = 20%

3.5.1. Training set

This part of the dataset is well known to the trained model. The model is acquainted with not only input data but also output results. Due to this fact training set cannot be a base for validation of results achieved after training. Due to the fact that the model knows the true results, it can try to fit the training data without generalization. This is a very serious problem in deep learning research. One of the main objectives is to make a model, which is not over-fitted; this means that the given model receives similar results for training and validation sets.

3.5.2. Validation set

After training, the accuracy of the model has to be validated. To do so, the validation set is prepared. In contrast to the training set, its captions are not known to the model. The validation set is iterated once after training. As its name implies, its purpose is to validate the trained model. Base on it, the researcher can calibrate the model or tune the hyperparameters. The validation set can be used many times to test the model, and its goal is to give the information to the researcher how well-calibrated his model is. However, it can not be used for the final evaluation of the model. For this purpose, the test set is prepared.

3.5.3. Test set

The role of the test set is to give possibly the fairest and the most independent judgment. Therefore, under no circumstances should the test set be used during calibration nor tuning the model; for this purpose, the validation set is prepared. The test set has to be used only once! These precautions make it possible to achieve a real and reliable assessment of the quality of model preparation. Just like in the case of the validation set the true captions are not known to the model.

3.6. Data models

The data models section introduces two concepts of the data, which has a great impact on the whole pipeline for the image captioning process. The first of them is a collection of the input images and output sentences. My dataset is the COCO dataset, I decided to pick this specific one because it has great API for the data managing and validation. The data set is the most important factor for each machine learning solution, especially those in the deep learning field, which highly depends on training data. The second data model is a repository made up by myself to store crucial information computed during the preprocessing and training process.

3.6.1. COCO dataset

COCO is the dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context. Objects are labeled using per-instance segmentations to aid in precise object localization. The dataset contains photos of 91 objects types that would be easily recognizable by a 4-year-old. With a total of 2.5 million labeled instances in 328k images, the creation of dataset drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting, and instance segmentation [23]. The COCO dataset is a great source of high-quality images and captions (Fig. 3.5).

This dataset has a version specially dedicated to the Image Captioning task[3]. It consists of 80,000 training images and 40,000 validation images, each annotated with 5 captions written by workers on Amazon Mechanical Turk. The annotations are stored using JSON. This project uses annotations for image captioning; their data structure with the data types is described below:

```
{
    "info": info,
    "images": [image],
    "annotations": [annotation],
    "licenses": [license],
}

info
    {
        "year": int,
        "version": str,
        "description": str,
        "contributor": str,
        "url": str,
        "date_created": datetime,
    }
```

[3] *http://cocodataset.org/#captions-2015*

```
image
    {
        "id": int,
        "width": int,
        "height": int,
        "file_name": str,
        "license": int,
        "flickr_url": str,
        "coco_url": str,
        "date_captured": datetime,
    }

license
    {
        "id": int,
        "name": str,
        "url": str,
    }
```

Fig. 3.5. An example picture with proposed captions from the COCO dataset[4].

3.6.2. Repository

The main role of the repository is to store computing information during an executing pipeline. It accumulates everything necessary to train and validate the model. This solution not only allows us to avoid unnecessary computing but also allows us to unload computer memory. Moreover, this approach allows creating a robust and easy to restore system, while preparing this work it was very important for me to prepare the solution, that is portable and easy to reproduce on other machines. The repository is storing information about:

- The paths to the NumPy files with extracted image features prepared for the training, validation, and test process.
- The preprocessed caption vectors for each image prepared for the training process.
- The tokenizer used for image captions preprocessing.
- The longest caption record.
- The start epoch record.

The repository is stored thanks to the Python Pickle module[5]. This module implements binary protocols for serializing and de-serializing a Python object structure. It is a very convenient tool for storing Python dictionaries. Dictionary data structure designated for storing my repository. The repository dictionary structure with corresponding data types:

```
{
    "train_feature_path_list": list of str,
    "val_feature_path_list": list of str,
    "test_feature_path_list": list of str,
    "train_caption_list": list of str,
    "val_img_id_list": list of int,
    "tokenizer": tensorflow.keras.preprocessing.text.Tokenizer,
    "max_length": int,
    "start_epoch": int
}
```

3.7. Analysis methods

An analysis is one of the key parts during a deep learning project research. It is worth to track a loss obtained during training because a loss plot can be constructed. The loss plot is one of the most important factors during looking for an optimal solution. It can tell how the process of training looks like; on its basis, a skilled researcher is able to assess the quality of the solution proposed and, if necessary, make the necessary corrections. However, the desirable training loss plot curve shape is not enough for assessing

[5] *https://docs.python.org/3/library/pickle.html*

the quality of the obtained model. The high accuracy during training can be misleading and can lead to over-fitting of the model, thus after training the validation phase should come. The validation step tells how the model performs on a fresh not known input data, thus it allows to assess the quality of the predicted output. Moreover, it allows checking, how well the model is generalizing (check whether the model has not over-learn on a training set). If the validation and training results are comparable, the model is well trained and reliable. Another form of model analysis is the attention plot. The attention plot allows seeing how the computer predicted each specific word in relation to the region of the give input image.

3.7.1. Validation

The validation is the process of comparing the output of the prepared model with the ground-truth label. In this scenario, the generated sentence by a computer is compared with the caption given in the COCO dataset. This process allows answering the question of how good the model is in predicting a textual description close to one prepared by the human hand. Here is presented the sample of five predictions received from COCO API:

```
A line of people in ski gear prepare to ski.
A group of children and adults in skis are standing on a snowy hill.
The group of people are standing while wearing skis and ski poles.
A group of people in gear standing in the snow.
a group of people lined up  in the snow standing on some skis
```

They are the basis for the validation of each predicted text sentence. The predicted caption is compared with the corresponding five ground truth labels. The question arises, how the result of being a sentence can be evaluated. I decided to pick the metrics used by the COCO evaluation server [24]. This work utilizes the following metrics: BLEU-1, BLEU-2, BLEU-3, BLEU-4, ROUGE-L, METEOR, and CIDEr-D. The details of these metrics are described next.

3.7.2. BLEU

Bilingual Evaluation Understudy algorithm or BLEU score [25]. The BLEU score comes from work in machine translation, which is where image captioning takes much of its inspiration; as well as from image ranking/retrieval and action recognition. BLEU was one of the first metrics to claim a high correlation with human judgments of quality [25], and remains one of the most popular automated and inexpensive metrics. BLEU has shown good performance for corpus-level comparisons over which a high number of *n-gram* matches exist. However, at the sentence-level then-gram matches for higher *n* rarely occur. As a result, BLEU performs poorly when comparing individual sentences.

Understanding the basic BLEU score is quite intuitive. Scores are calculated for individual translated segments by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account.

3.7.3. ROUGE

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation [26]. It includes measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. The measures count the number of overlapping units such as *n-gram*, word sequences, and word pairs between the computer-generated summary to be evaluated and the ideal summaries created by humans. ROUGE is a set of evaluation metrics designed to evaluate text summarizing algorithms.

3.7.4. METEOR

The Meteor automatic evaluation metric scores machine translation hypotheses by aligning them to one or more reference translations. Alignments are based on exact, stem, synonym, and paraphrase match between words and phrases. Segment and system-level metric scores are calculated based on the alignments between hypothesis-reference pairs. Meteor has extended support (paraphrase matching and tuned parameters) for the following languages: English, Czech, German, French, Spanish, and Arabic. Meteor consistently demonstrates a high correlation with human judgments in independent evaluations such as EMNLP WMT 2011 [27].

METEOR [28] is calculated by generating an alignment between the words in the candidate and reference sentences, with the aim of 1:1 correspondence. This alignment is computed while minimizing the number of chunks of contiguous and identically ordered tokens in the sentence pair. The alignment is based on exact token matching, followed by WordNet synonyms [29], stemmed tokens, and then paraphrase.

3.7.5. CIDEr

CIDEr (Consensus-based Image Description Evaluation) [30] is an automatic consensus metric for evaluating image descriptions. Most existing datasets have only five captions per image. Previous evaluation metrics work with these small number of sentences and are not enough to measure the consensus between generated captions and human judgment. However, CIDEr achieves human consensus using the term frequency-inverse document frequency [31].

3.7.6. Attention plot

The last part of my verification process is the model's prediction decision explanation. This process provides information on how the model predicted each specific word based on the input image region. It allows us to look at how the computer has constructed an output sentence. Each word of a sentence has a designated part of an image. The attention plot is fully made by computer. Based on this, the researcher can decide whether the whole system is working correctly. It is a very high-level test, thanks to it we can find out how well the computer copes with signing photos and compare it with the human-based process of creating the image caption. Fig. 3.6 presents an example part of the attention plot. It is an

input image with a single predicted word. I have added attention weights returned as an extra parameter from the decoder; they are small rectangles on the input. Attention weights represent, which parts of an input image were responsible for the prediction of a specific word.

Fig. 3.6. An example part of the attention plot: the input image with corresponding the attention weights and predicted word.

4. Suggested solution and implementation

A starting point for my project was the work created and shared by the Google [32]. Their work gives me an insight how the example pipeline and workflow looks like in case of the image captioning task. It shows me how to prepare and develop a project capable of providing an accurate image description. Equally important honorable acknowledgement is a paper "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" [1]. This work proposes the model architecture, which was an inspiration to me.

This chapter will show and describe in detail the proposed solution, along with the implementation details of the complete image captioning project. I would like to emphasize that some solutions are the reason for my use of a virtual machine. Preprocessing images and training image-captioning models are very resources demanding processes. Moreover, Tensorflow is very memory-intensive because it is designed to load entire batches of data. So I had to rent a machine designed for such a task. I had to face a real challenge. Fortunately, thanks to this, my project is portable and ready to use on every other machine. The whole code is available on my GitHub repository[1]. The repository contains the Jupyter Notebook `ImageCaptioning.ipynb`[2] with an end-to-end example usage of the image captioning API proposed by me. The next parts of this chapter are practical walk through the software API prepare by me with results at the end. To be able to accompany me fully please clone my GitHub repository:

```
git clone https://github.com/tynski/ImageCaptioning.git
```

Here are the parameters used by me during the learning process:

```
batch_size = 64
top_k = 5000
epochs = 20
train_samples = 30000
val_samples = 10000
test_samples = 10000
```

[1] *https://github.com/tynski/ImageCaptioning*

[2] *https://github.com/tynski/ImageCaptioning/blob/master/ImageCaptioning.ipynb*

4.1. Acquire the dataset

The basis and beginning is the data set. The crucial element of each machine learning solution is an input data. Moreover, the image captioning is a task categorized as supervised learning. This means that besides an input, the dataset model should also contains an expected prediction. This work utilizes the COCO dataset, more fully described in Section 3.6. I decided to pick this specific dataset, because it has great API for managing this huge portion of data.

Firstly the dataset has to be downloaded and stored. In order to make program run correctly I have designed the following data director hierarchy:

```
coco/ (dataset directory)
    train2014/ (training set)
    val2014/ (validation set)
    test2014/ (test set)
    ann2014/
        annotations/
            captions_train2014.json (training captions)
            captions_val2014.json (validation captions)
            captions_test2014.json (test captions)
```

I prepared a dedicated Bash script to execute whole operations connected with preparing datasets. However, due to the fact that it is a big amount of data, it requires some extra GNU/Linux dependencies: gsutils, unzip. With all necessary tools equipped, the dataset can be acquired by simply running Bash script:

```
./getdataset.sh
```

The script will acquire the data and store it in the right location.

4.2. Prepare the repository

However, the just downloaded the data is not ready to use. Before the training process can be initialized it should be appropriately preprocessed. The right pre-processing algorithms of the captions and features of an image are essential to training a good model. Because this is an important and memory demanding process, I decided to preprocess the whole picked dataset and create a repository to store the necessary information used during the training, validation, and testing phase. The idea and data model are more deeply cover in Section 3.6. The code bellow will handle all the preprocessing crucial for the training, validation, and testing:

```
import pickle
from image_captioning.data import data_processing

repository = data_processing.prepare_repository(train_samples, val_samples, top_k)

with open('image_captioning/data/repository.pkl', 'w+b') as file:
```

```
pickle.dump(repository, file)
```

The results will be stored in the repository dictionary. Below is described the processing of the information included in the repository.

4.2.1. Prepare the input data

To extract features from each selected input image the pre-trained convolutional neural network InceptionV3 [13] is used. Fig. 4.1 presents the architecture of used pre-trained CNN model, InceptionV3. The input images have to be appropriately preprocessed in a way to match the InceptionV3's expected format:

– resized to 299 pixels by 299 pixels,

– normalized so that it contains pixels in the range of -1 to 1,

– the shape of each image has to be: width = 299, height = 299, depth = 3.

After transformation images to the format, which matches the requirement of the InceptionV3. Each image will be processed in a way to detect objects on a picture and extract them. The shape of each extracted feature vector is (64, 2048). Features are extracted and stored as the NumPy file in such hierarchy:

```
coco/
    train2014_features/
    val2014_features/
    test2014_features/
```

This allows avoiding repeating the same activity in the future. The Python list of strings containing the full path to each NumPy file with image features is saved in the repository dictionary.

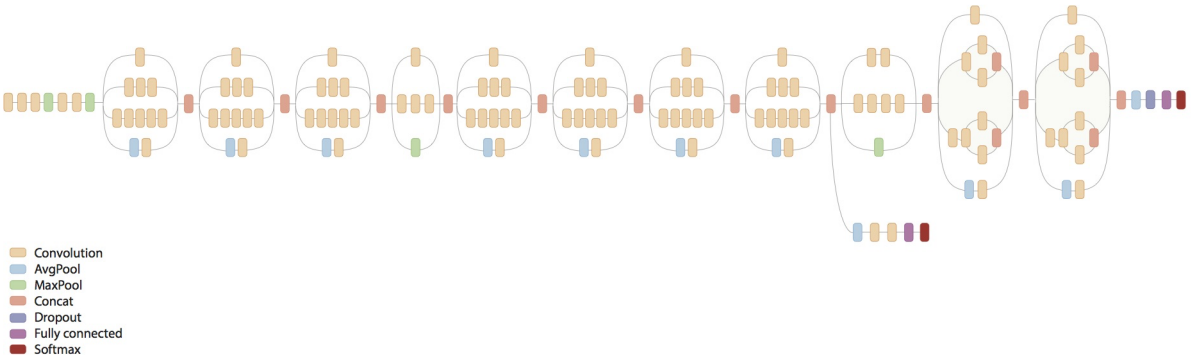

Fig. 4.1. Visualization of the InceptionV3 model architecture.

4.2.2. Prepare captions

The COCO dataset provides captions to each picture. However, each caption has to be pre-processed and tokenized, respectively. This process allows vectorizing a text corpus by turning each text into a sequence of integers (each integer is the index of a token in a dictionary). The mechanism looks as follows:

1. Choose the top k words from the vocabulary replace all other words with `<unk>`.
2. Filter out the given characters from each caption:

   ```
   !"#$%&()*+.,-/:;=?@[\]^_`{|}~
   ```

3. Extend each caption with adding `<start>` at the start and `<end>` at the end.
4. Create word-to-index and index-to-word mappings.
5. Create the tokenized vectors.
6. Pad each vector to the max length of the captions; the padding value is 0.

After padding, the length of each caption vector is the same as the max length. Prepared captions, tokenizer, and max length of the captions are stored in the repository to be used in further pipeline steps.

4.3. Deep learning model architecture

This project is based on the encoder-decoder approach, more deeply covered here [1]. Fig. 4.2 shows an illustrative diagram. The encoder definition is kind of tricky due to the fact that this solution is using the pre-trained InceptionV3 for the feature extraction. Basically, the InceptionV3 is part of an encoder, the second part is single fully connected layer. So, the encoder is divided in to parts: feature extractor and neural network layer for interpreting obtained features. The decoder is Gated Recurrent Unit [33] reinforced with soft attention presented in [19].

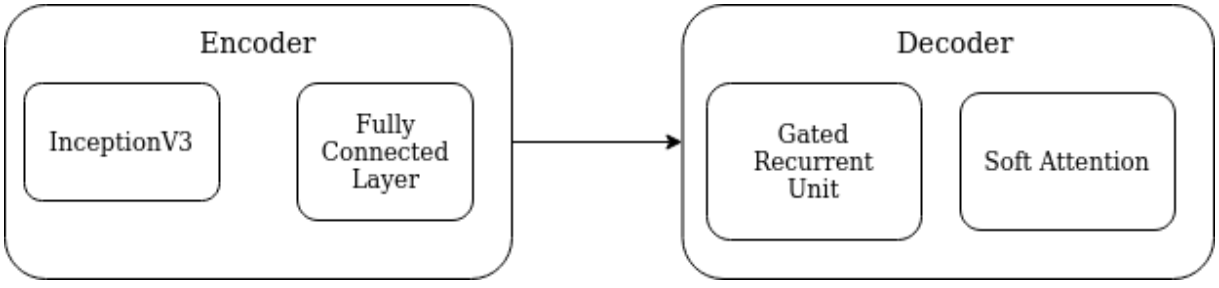

Fig. 4.2. Encoder-decoder model architecture.

4.3.1. Encoder architecture

As I mentioned earlier, the encoder work is divided into two parts. The first step was described in Section 4.2.1, which is feature extraction. This step is carried out before any training at the preprocessing phase. The second step relies on encoding the features extracted from the InceptionV3. For encoding, I decided to use a simple, single, fully-connected neural layer with the RELU activation function. The second step executed during the training phase, when the is a need to deliver the input data to the decoder.

4.3.2. Models loading and preparation

The solution allows weights loading from previous training sessions. This procedure is very common in machine learning solutions and allows:

– Saving checkpoints, after each five epochs the weights for each model are saved. It makes that the program is resistant to possible system errors and allows you to stop training at any time without too much loss.

– Reuse model in a real use case. The purpose of this solution is to help in everyday life, so it has to be stable and fast in reusing.

– Achieve the same results during validation.

– Share solutions among the other contributors.

To track models, save and restore them `tensorflow.train.Checkpoint()`[3] was utilized.

4.4. Deep learning model workflow

When the input data is obtained and preprocessed, and the model architecture is chosen, it is high time to start working on the deep learning model for the image captioning. My deep learning model workflow looks as follows:

1. Prepare model.
2. Training. Fit the model based on the well known input and output data.
3. Validate. Check how good is the prototype model on predicting an output data based on a not known input data.
4. Test. Evaluate the final model.

The following code shows how to utilize the API prepared by me to proceed steps written above:

[3] *https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint*

```
from image_captioning.model_manager import Model

model = Model(top_k, batch_size)
model.train(train_samples, epochs)
model.validate(val_samples)
model.test(test_samples)
```

4.4.1. Training

The training process is based on iterating through batches of the input data. After going through the whole training samples, the procedure is repeated as many times as many as are given epochs. The iteration for a single batch is simple. The extracted features were stored in the respective NumPy files. Those features go through the encoding fully connected layer. The results obtained from the fully connected layer are treated as an input to the decoder model. The decoder returns the predictions and the decoder hidden state. The decoder's hidden state is then passed back into the model, and the predictions are used to calculate the loss. The Recurrent Neural Network is trained by the teacher forcing method. Teacher forcing is a strategy for training recurrent neural networks that uses model output from a prior time step as an input. The final step is to calculate the gradients and apply it to the optimizer and backpropagate. The loss archived during training is shown on Fig. 4.3.

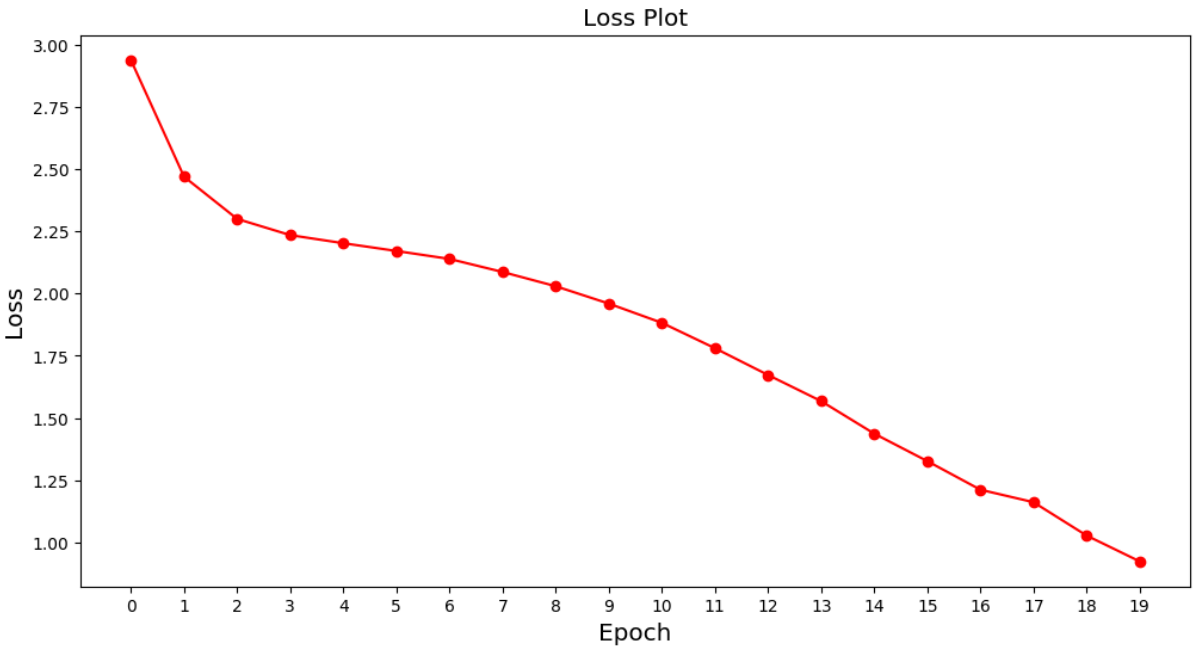

Fig. 4.3. Values from loss function gathered during training.

4.4.2. Evaluation and validation

The evaluation function is similar to the training loop, except there is no use of teacher forcing here. The input to the decoder at each time step is its previous predictions along with the hidden state and the encoder output. The prediction is stopped when the model predicts the end token. Fig. 4.4 shows a sample image, which will be treated as an input data for my trained image captioning model. This image will

Fig. 4.4. Sample input image, which will be passed through deep learning pipeline to receive textual description.

be passed through a deep learning pipeline to receive textual description. The deep learning model was prepared and trained on 30.000 input images within 20 epochs. The predicted caption and five reference captions given by the COCO dataset are presented below:

```
True captions:
A line of people in ski gear prepare to ski.
A group of children and adults in skis are standing on a snowy hill.
The group of people are standing while wearing skis and ski poles.
A group of people in gear standing in the snow.
a group of people lined up  in the snow standing on some skis

Predicted caption:
a group of people skating on a snow near the woods
```

Moreover, the decoder returns attention weights for each predicted word. This very cool feature enables to debug and explain image captioning model prediction. Fig. 4.5 presents the specific location of an input image, which is the attention of the decoder. The attention indicates the most important parts of the given image for the decoder to make a decision about the next word prediction.

The evaluation of one or several images is not objective, because we can get extreme results: very good or very bad description. Therefore, a validation set was created, which includes 10.000 input images. This will allow us to get a generalized prediction rating. Table 4.2 presents the results of such a validation for eight metric: CIDER, BLEU4, BLEU3, BLEU2, BLEU1, ROUGE, METEOR, SPICE. The scores were computed with the `coco-caption` code [24]. Each method evaluates a candidate sentence by measuring how well it matches a set of five reference sentences written by humans. Fig. 4.6

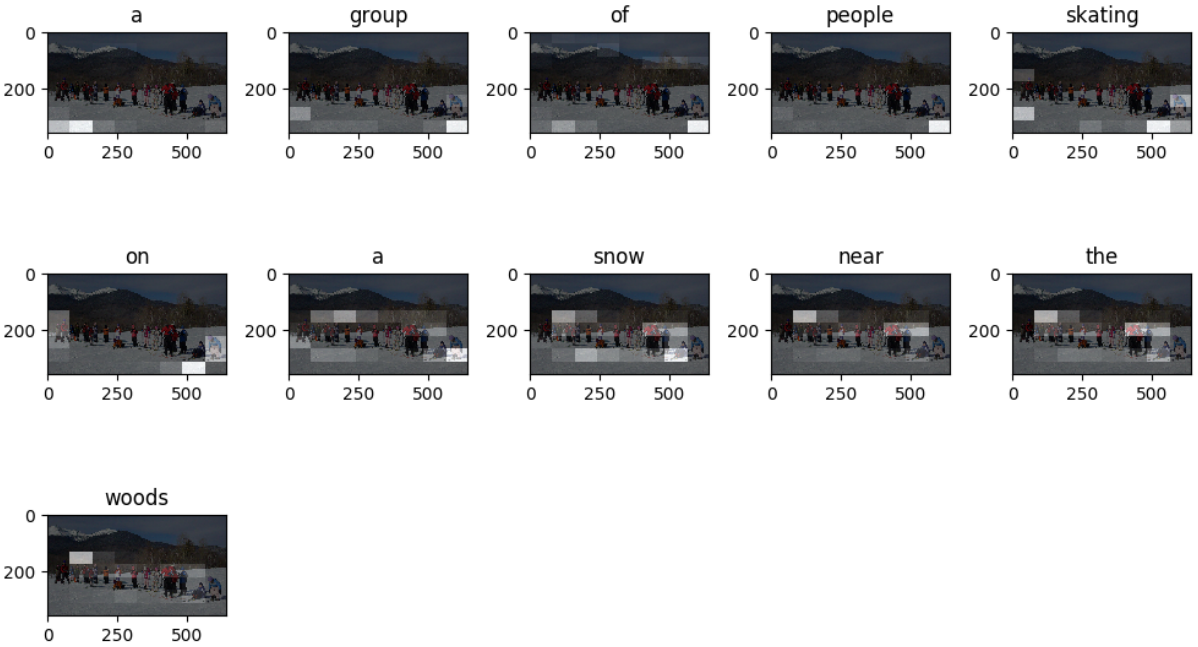

Fig. 4.5. An input image with corresponding attention weights for each predicted word.

shows histograms of CIDER, ROUGE, METEOR, BLEU-1, BLEU-2 and BLEU-4 scores for validation set captions.

Table 4.1. CIDER, BLEU4, BLEU3, BLEU2, BLEU1, ROUGE, METEOR, SPICE metrics for the validation data.

CIDER	BLEU4	BLEU3	BLEU2	BLEU1	ROUGE	METEOR	SPICE
0.225	0.061	0.118	0.226	0.411	0.315	0.134	0.077

4.4.3. Test

This is the final step of each serious deep learning project. Previous steps were repeated many times in order to receive the most accurate and reliable prediction. However, this is supposed to proceed only one time, so when the solution is ready to use and finished its is a time to give a final assessment of its quality. The test set includes 10.000 input images. Table 4.2 presents the results of the test set validation for eight metrics: CIDER, BLEU4, BLEU3, BLEU2, BLEU1, ROUGE, METEOR, SPICE. Fig. 4.7 shows histograms of CIDER, ROUGE, METEOR, BLEU-1, BLEU-2 and BLEU-4 scores for test set captions.

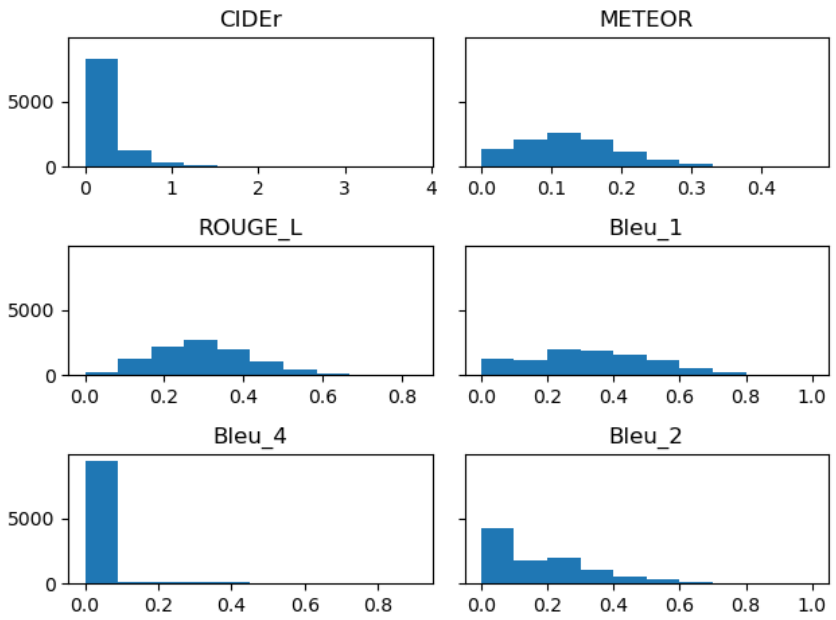

Fig. 4.6. The histograms showing CIDER, ROUGE, METEOR, BLEU-1, BLEU-2 and BLEU-4 scores for validation set captions.

Table 4.2. CIDER, BLEU4, BLEU3, BLEU2, BLEU1, ROUGE, METEOR, SPICE metrics for the test data.

CIDER	BLEU4	BLEU3	BLEU2	BLEU1	ROUGE	METEOR	SPICE
0.213	0.060	0.115	0.221	0.400	0.310	0.133	0.075

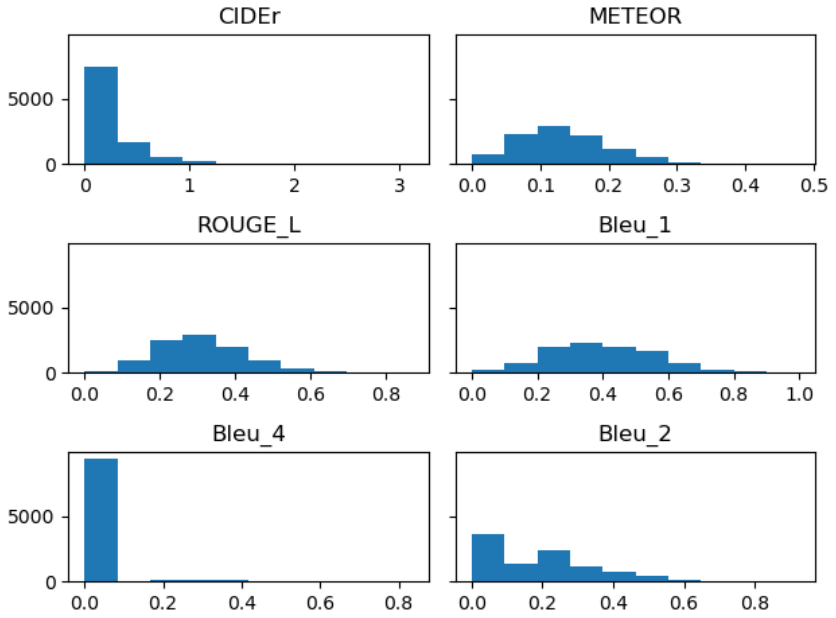

Fig. 4.7. The histograms showing CIDER, ROUGE, METEOR, BLEU-1, BLEU-2 and BLEU-4 scores for test set captions.

5. Critical reflection

Many of the challenges that I faced when preparing the image captioning model had to do with overfitting. Purely supervised approaches such as image captioning tasks require large amounts of data. The task of assigning a description is strictly harder than object classification, and data-driven approaches have only recently become dominant thanks to datasets as large as ImageNet. Overfitting is a real problem for image captioning. Besides overfitting, the image captioning is a very computationally demanding process. One of the solutions being the current state of the art "Show and Tell: A Neural Image Caption Generator" [34] trained their solution for several weeks on very advanced computational equipment:

'*In our experience on an NVIDIA Tesla K20m GPU, the initial training phase takes 1-2 weeks. The second training phase may take several additional weeks to achieve peak performance ...*'

Chris Shallue[1]

That's why it's clear that neither my goal was to compete with the current state of the are in the field of image captioning nor proposed ground-breaking techniques. I decided not to compare nor brag about the results with related work. I believe, that there exist many works that presented much better models than mine. Nevertheless, my honest goal was to study and learn as much as possible in the field of deep learning and software development. I wanted to provide a solution that is well designed and understood by the potential user; the code should be written in a proper object-oriented way so that it becomes easier for others to replicate. After reflecting on my work, I believe that I have fully achieved those goals, which was determined at the beginning of this project. I can say that I am truly satisfied with the results of my own work.

5.1. Discussion of open issues

I realize that my project is far from ideal, but I was very low on computational resources, which are crucial for a data-driven solution. However, I achieved the intended goal, because my solution is well

[1] *https://github.com/tensorflow/models/tree/master/research/im2txt*

maintained and ready to use[2]. Nevertheless, I think it is important to mention the modifications can be made to improve this solution:

- Larger dataset. I decided to randomly pick 50.000 input images with corresponding captions due to technological restrictions. However, the COCO dataset includes more than 200.000 labeled images [23].
- Hyperparameter tuning (learning rate, batch size, number of layers, number of units, batch normalization, etc.).
- Input image augmentation. It is generating more training data from existing training samples, by augmenting the samples via a number of random transformations that yield believable-looking images. The goal is that at training time, the model would never see the exact same picture twice. This helps the model get exposed to more aspects of the data and generalize better.
- Validation during training. A part of the training set becomes a validation set; the model is trained on the training set, however, the loss is computed for both validation and test sets. This helps to monitor that the model is not overfitting.

5.2. Lessons learn

I would like to draw my conclusions related to the technology stack picked by me. Python was a great choice as the main programming language for creating an image captioning project. It is unbelievable good at prototyping and developing deep learning projects. It has a diversity of libraries, which can handle most problems occurred during a workflow. Python supports data structures that enable efficient data access and modification. I am gladly picking this specific programming language. Now a few words about the Tensorflow library. In the beginning, it is worth saying that Tensorflow is a great deep learning library. It is very rich and extensive, which contains every utility one can imagine. That is why it makes sense that it is so popular and constantly developed. However, I have to admit that work and debugging with this specific library was a real nightmare for me. Moreover, Tensorflow's documentation was very obscure and not clear for me, in most cases I have to retrospect Tensorflow's objects on my own to gather the information I want. One of the most prominent examples is the Tensorflow's Tokenizer[3] object, which does not contain information about its attributes. I had to inspect the Tokenizer object on my own. Besides, I have a lot of trouble with fixing bugs connected in Tensorflow backend execution. When any error occurred connected with Tensorflow, I had a lot of troubles to know where the crash happened. To conclude, despite the problems described above, I am happy with my choices. I am well acquainted and proficient with technology widely used in the computer science industry!

[2] *https://github.com/tynski/ImageCaptioning*

[3] *https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer*

6. Summary and future work

I have presented an end-to-end neural network system that can automatically view an image and generate a description in plain English with very few hard-coded assumptions. My image captioning project is based on a convolution neural network that encodes an image into a compact representation, followed by a recurrent neural network that generates a corresponding sentence. The model was trained to maximize the likelihood of the sentence given the image and validated with corresponding evaluation metrics.

I have plenty of ideas how to develop and improve the proposed solution. Of course, the first thing that comes to mind is caption predicting model enhancement. The model should be trained more, with special precautions to overfitting. The current state of the art presents many techniques, which helps to improve overall accuracy and quality of description. Another interesting direction is the fact, that the produced descriptions are one of many possible image interpretations. One possible direction is to have a system which is capable of more targeted descriptions – either anchoring the descriptions to given image properties and locations or being a response to a user-specified question or task. However, this is not my particular object of interest. What really turns me on is a real image captioning application for a potential customer. I mean a web solution or a mobile application, where a typical user can upload his photo to obtain input image with a corresponding predicted caption. I think it will be a very good idea to bring this project to a more user-friendly interface.

I must admit that I am very happy choosing this project as the subject of my master's thesis. This project was a great experience, and I have not only acquired knowledge in deep learning but also a as computer scientist. Moreover, I am acquainted with novel techniques in the field of Natural Language Processing and Computer Vision. This project allows me to get to know an artificial intelligence better, how it works, and what its limitations are. I am very happy that I had the opportunity to work on such an educational project. Thanks to this project, my work culture has improved, I make better use of a version control system, and I get to know how to efficiently debug such a project. I put every part of code with the following results on my GitHub page[1], to be available for everyone to run my solution.

[1] *https://github.com/tynski/ImageCaptioning*

Bibliography

[1] Oriol Vinyals et al. “Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge”. In: *CoRR* abs/1609.06647 (2016). arXiv: *1609.06647*.

[2] Ali Farhadi et al. “Every picture tells a story: Generating sentences from images”. In: *European conference on computer vision*. Springer. 2010, pp. 15–29.

[3] Girish Kulkarni et al. “Babytalk: Understanding and generating simple image descriptions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), pp. 2891–2903.

[4] Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. “Multimodal neural language models”. In: *International conference on machine learning*. 2014, pp. 595–603.

[5] 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, pp. 3128–3137.

[6] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. “Densecap: Fully convolutional localization networks for dense captioning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4565–4574.

[7] Kelvin Xu et al. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. 2015, pp. 2048–2057.

[8] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. “Unifying visual-semantic embeddings with multimodal neural language models”. In: *arXiv preprint arXiv:1411.2539* (2014).

[9] Quanzeng You et al. “Image Captioning With Semantic Attention”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[10] Google Brain Team. *"Show and Tell: image captioning open sourced in TensorFlow". https://ai.googleblog.com/2016/09/show-and-tell-image-captioning-open.html*. [Online; accessed 28-February-2020]. 2016.

[11] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). arXiv: *1409.4842*.

[12] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: *1502.03167*.

[13] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567 (2015). arXiv: *1512.00567*.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *“Deep Learning”*. *http : // www . deeplearningbook.org*. MIT Press, 2016.

[16] Andrej Karpathy. *“The Unreasonable Effectiveness of Recurrent Neural Networks”*. *http : // karpathy.github.io/2015/05/21/rnn-effectiveness/*. [Online; accessed 09-March-2020]. 2015.

[17] Hava T Siegelmann. “Computation beyond the Turing limit”. In: *Science* 268.5210 (1995), pp. 545–548.

[18] Christopher M Bishop. *“Pattern recognition and machine learning”*. Springer, 2006.

[19] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[20] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Zhiheng Huang, Wei Xu, and Kai Yu. “Bidirectional LSTM-CRF Models for Sequence Tagging”. In: *CoRR* abs/1508.01991 (2015). arXiv: *1508.01991*.

[22] Thomas Elliott. *“The State of the Octoverse: machine learning”*. *https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/*. [Online; accessed 27-February-2020]. 2019.

[23] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: *1405.0312*.

[24] Xinlei Chen et al. “Microsoft COCO Captions: Data Collection and Evaluation Server”. In: *CoRR* abs/1504.00325 (2015). arXiv: *1504.00325*.

[25] Kishore Papineni et al. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[26] C.-Y. Lin. “Rouge: A package for automatic evaluation of summaries”. In: *ACL Workshop* (2004).

[27] Chris Callison-Burch et al., eds. *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland: Association for Computational Linguistics, 2011.

[28] Michael Denkowski and Alon Lavie. “Meteor Universal: Language Specific Translation Evaluation for Any Target Language”. In: *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*. 2014.

[29] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

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

[31] Stephen Robertson. "Understanding inverse document frequency: on theoretical arguments for IDF". In: *Journal of documentation* (2004).

[32] Google. *"Image captioning with visual attention". Creative Commons 4.0 Attribution License. https://www.tensorflow.org/tutorials/text/image_captioning*. [Online; accessed 02-March-2020].

[33] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[34] Chris Shallue. *"Show and Tell: A Neural Image Caption Generator". https://github.com/tensorflow/models/tree/master/research/im2txt*. 2020.