Automated image captioning

Adela Puscasiu
Department of Automation
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Adela.Puscasiu@aut.utcluj.ro

Alexandra Fanca
Department of Automation
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Alexandra.Fanca@aut.utcluj.ro

Dan-Ioan Gota
Department of Automation
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Dan.Gota@aut.utcluj.ro

Honoriu Valean
Department of Automation
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Honoriu.Valean@aut.utcluj.ro

***Abstract*— The research of comprehension of machine learning algorithms and procedures suitable for both image processing and natural language processing. The accommodation with existing packages used in implementing machine learning algorithms. The implementation of an algorithm that takes an image and describes it in comprehensive sentences. After the analyzed requirements, a bibliographic study was conducted for familiarization with the domain and possible models. Next, a study of the available technologies which would help to build the application was made, and then the design, implementation, and validation of the solution began. This paper presents a composite model, consisting of a deep convolutional neural network for feature extraction that makes use of transfer learning, and a recurrent neural network for building the descriptions. Keras with TensorFlow backend is used for implementation. A trained model that describes images using natural language was obtained.**

Keywords*— *image captioning, machine learning, model.

I. INTRODUCTION

Technology has become an integrated part of our daily lives over the past decades. From offering readily-answers with the use of search engines, ease of world-wide cheap communication through the messaging applications that run on the internet to solving complex engineering problems.

Once with the ever-growing information and data and new technologies, the filed that has been growing rapidly is artificial intelligence. More and more different companies seek to incorporate at some level data science and machine learning algorithms while developing their products while the rest of the companies seek to take advantage of the information that can be deduced using these methodologies [1].

Applications for artificial intelligence, also known as machine learning or deep learning, depending on the type of algorithm chose, are vast and lie over many fields, from getting insights of the financial transactions of a company, to detecting fraud; from aiding in developing better guidance systems (GPS applications and maps) to identifying elements from a satellite image; from language translation to text-to-speech applications, and the list could go on [2].

When it comes to specific fields involved in developing machine learning algorithms, the most notable are computer science, mathematics, and statistics. The difference between machine learning algorithms and deep learning algorithms is the approach. Deep learning makes use of neural networks, which have an incredible computation power but also require a lot of data and strong hardware to give good results. It is also just a subcategory of machine learning. It is based on building neural networks that can learn by themselves, while the machine learning algorithms are considered classic algorithms that take the data and try to learn from it and apply it further. If a deep learning algorithm can determine whether the prediction is good or not, is a machine learning algorithm gives a bad prediction, the developer has to go and adjust the parameters and retrain the model [3]. This step is called hyperparameter tuning.

For an accurate algorithm, it must be fed good data. The datasets that are available for toy projects are usually clean and need few modifications or not at all, but the real-life data is usually messy and needs a lot of cleaning and normalization. This step is called data preprocessing and it can take up to 80% of the time of a machine learning engineer. When it comes to unstructured data, such as images and text, even the datasets available online have to be modified to fit the inputs or specifications of the neural networks [4].

People can extract data from almost everything that surrounds them. For example, when hears a sound, it can describe it using natural language. When it comes to machines, they cannot do that readily. Humans can make use in that way of every sense they possess. Now imagine if one of those senses is missing. Machine learning applications can help, for instance, deaf people when it comes to written information by using a text-to-speech algorithm to “read” to them.

People can also extract information from images. If a person is presented with an image showing a flower, given that the person has seen a flower before, she or he will be able to identify

978-1-7281-7166-1/20/$31.00 ©2020 IEEE

that the object. But if that person is visually impaired, they cannot see that image at all. What if there was a way for them to gather some of the information without needing to see the image or have somebody else describe it to them? Machine learning can aid in that direction [5].

This paper aims to extract meaningful textual information from images, in the form of short descriptions. The results can be further run through a text-to-speech engine to offer full sustainability. This way, a full independent experience could be delivered to people with this disability. With the rise of social media interactions, these people can feel like they are being left out, especially when it comes to their families and close friends posting pictures online.

The rest of the paper is organized as follows: Section II presents a study in the field and some of the existing systems, the proposed idea is presented in Section III and in Section IV the conclusions are presented.

II. Related work

A combined bottom-up and top-down attention mechanism that enables attention to be calculated at the level of objects and other salient image regions which is the natural basis for attention to be considered is presented in [6].

Image caption generation is the problem of generating a descriptive sentence of an image. Also, automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. A brief survey of some technical aspects and methods for the description-generation of images is presented in [7]. Paper [8] presents a new challenge of image captioning which is how to effectively inject sentiments into the generated captions without altering the semantic matching between the visual content and the generated descriptions.

A comparative study of an attention-based deep learning model for image caption generation is presented in [9]. For image captioning, the encoder-decoder network based on attention mechanism is very useful to generate sentence descriptions [10]. Also, attention mechanism attends a more salient part of the image in the output of the encoder network and converts feature maps to feature vector for the input of the decoder network. The paper also suggests important steps for image caption generation and presents datasets that mostly used, and evaluation metrics to calculate the performance.

This paper focuses on the translation of images into text, or in other words, making textual sense of the contents of an image. The first step would be researching the available algorithms and the best combinations of them, referring to the model architecture. This can be considered a lengthy part, since there are a lot of options available, with plenty to choose from when it comes to most of the general issues.

After choosing the right set of algorithms and models, picking a suitable dataset is the next step. There are a lot of datasets out there that are based on images and used for supervised learning at a great scale, but most of them are specifically collected and labeled for different types of applications. Ideally, an image dataset that has the labels in the form of written sentences describing the contents of the image has to be selected. The dataset also must have a lot of entries, since deep learning algorithms require vast amounts of data to learn and to be used in making good predictions.

After selecting the architecture, models, and dataset, one has to preprocess the dataset in a way that it matches the required inputs of the algorithms. This usually means resizing the images, for the neural network that deals with images, and tokenizing and vectorizing the textual data, for the neural network that deals with the natural language part.

Once the data has been preprocessed, the models have to be implemented. They have to keep an object-oriented perspective and to be made clear and easy to use further, in the training part.

For the training part of the algorithm, the device that the program will run on has to be kept in mind. The deep learning algorithms are very powerful, but also need powerful devices to run on. With the complexity of the algorithm growing, the computational power of the device also has to grow. The algorithm can still be trained even on older devices, but if the dataset is too large, it might run into an "Out Of Memory" issue. It is recommended to be trained on GPU, but it also runs quite smoothly on newer CPUs. The issue is that if one makes the training process complex, it will take a long time for it to converge, from days to even weeks.

Also, in the process of training, various checkpoints of the results have to be saved. These checkpoints consist of the trained weights being saved, which can be later used for inferring the algorithm with new data. Also, if there was a problem while training, the last successful iteration would be saved. After training and saving the results of the algorithm, testing it is the next logical step. For this part, new data points are run through the saved results, evaluating them based on specific metrics. If the results are satisfying, one can consider the algorithm to be a success.

III. Proposed idea

One of the objectives is to research the vast field of machine learning to identify the best techniques to make use of in this specific context. Since there are a lot of algorithms and architectures available, it is easy to pick one that is suitable, but not the best technique possible, do a thorough research is implied whenever dealing with a problem that wants to be solved using deep learning or machine learning techniques. Luckily, there are a lot of papers and blog posts available, most of them referring to the theoretical aspects of the solutions proposed.

Although for many data scientists the programmatic part of the application is not of the highest concert. When developing an application that has to be run at the production level, the quality of the code becomes a priority. Another objective is to develop a program that is performant enough to be run almost in real-time; nobody wants to wait more than a few seconds for a result. This implies researching and testing different platforms and packages available for developing applications based on deep learning. With the rise of this field, many tools became available, all of them optimizing the compiling and computational part, so that developers can focus on the algorithms in making them better and faster.

At the same time, besides performance, integration must be kept in mind. Given that this application aims to be integrated further in various more complex projects, it should be developed in a way that integration is easy, giving it an object-oriented approach.

So, in the next subchapters are presented the dataset, the data preprocessing, the encoder-decoder architecture, the training phase, and the results.

A. The dataset

The dataset used for training and testing the model is MS-COCO, which stands for "Microsoft-Common Objects in Context". It was originally made available in 2014, its last revision being in 2015 [11].

This dataset is specifically created for usage in this type of problem. According to their website, it "is large-scale object detection, segmentation, and captioning dataset" [12]. It contains more than 80000 labeled images, making it one of the most popular datasets for image-related projects. What makes it a perfect fit for the problem at hand is that it has five different labels each containing one written description for each image.

It is also made use of the "separate" dataset that is represented by all the labels (captions) for training and testing the decoder.

For this paper, only 40000 captions and their corresponding images are used, being split into training and validation (testing) samples, using the 80:20 split ratio. This comes as a response to the performance issue in the training process.

B. The data preprocessing

Data preprocessing represents a very important step in every machine learning algorithm. Skipping this part results in the model raising an error because it does not receive the expected input.

This application requires two different types of data preparation: one for the deep convolutional neural network encoder and one for the deep recurrent neural network decoder.

Given that the deep convolutional neural network encoder is based on the Inception-v3 model, the images must be resized to the expected format, i.e. (299, 299) and the pixels must be brought in the [-1, 1] range. TensorFlow offers the "image" module for processing images, which allows for them to be read into memory, decoded as jpeg and resized. The application of the Inception-v3 of the Keras high-level API offers the "preprocess_input" method, which normalizes the pixels in the desired range, mentioned above.

Data preparation for the language generator decoder, i.e. the deep recurrent neural network requires the preprocessing of the textual data, i.e. the captions. For this part, the module "preprocessing" and its methods, offered by Keras, are used. The steps performed are:

- Caption tokenization, i.e. splitting the captions by white spaces, ending up only with the unique words;
- Vocabulary size limitation; the vocabulary size is limited to the top 5000 words, to save memory;
- Converting text into a sequence of numbers;
- Word-index mapping;
- Padding all the sequences to the length of the longest one.

The result is a vector of a sequence of integers, padded to be of the same length with the longest caption that was present in the dataset. This result is depicted in the image below:

```
In [58]:  1 caption_vector[0]

Out[58]: array([  2, 354, 672,   2, 275,   3,   2,  81, 340,   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,   0,   0,   0,   0])
```

Fig. 1 Array resulted after the preprocessing of the caption data

C. The Encoder-Decoder Architecture

The Encoder-Decoder architecture is widely used in applications such as machine translation, by using two Recurrent Neural Networks. The main idea behind this architecture is to resize the input vector required by the second network.

In this case, the first neural network is of convolutional type, while the second is a Recurrent Neural Network. The vector output of the first neural network is the input of the second one. A general schematic of this architecture is showcased below:

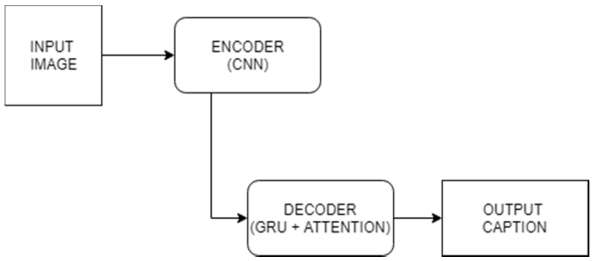


Fig. 2 Encoder-Decoder High-Level architecture

The attention mechanism, as described in the paper [13] acts as an interface between the encoder and the decoder. This is needed because without it, the data fed into the decoder would be just one vector representation. The attention mechanism offers information from every hidden state of the encoder, aiding the decoder in focusing on the useful parts [14].

1) The encoder

The output of this part is a vector of (8, 8, 2048) shape, but the application needs one of (64, 2048) shape to go into the recurrent neural network. For this, a Model class "CNN_Encoder" (Convolutional Neural Networks) is defined, which consists of one fully connected layer, followed by the ReLU (Rectified Linear Units) activation. The features extracted in the feature extraction part are read into memory and passed through this fully connected layer.

To simplify, the encoder is composed of the feature extraction using transfer learning with the Inception-v3 model, features that are further passed through a fully connected layer that also takes care of the resizing.

2) The decoder

The deep recurrent neural network decoder takes the encoded features from the encoder. The architecture of the recurred neural network is of Gated Recurrent Unit (GRU) type. The decoder attends over the image to predict the next word.

The decoder takes as inputs the shape of the vector outputted by the encoder, the number of neurons (units) per layer, and the vocabulary size, of the vocabulary, resulted in the data preparation part upon creation. When calling, one has to pass the extracted features and the reset state.

The features from the deep convolutional neural network encoder are fed into the attention layer, which returns the context vector and attention weight, being treated as a separate model. The returned context vector is concatenated with the model's embedding and passed through the layers of the gated recurrent unit network.

The recurrent neural network architecture consists of the gated recurrent unit layer and two fully connected layers. All of these layers can be added from the Keras API's module "layers". The gated recurrent unit layer is of type "GRU", taking as input upon creation the units (number of neurons) it should possess. The fully connected layers (or "dense" layers) are of type "Dense". The first fully connected layer takes as input the units, while the second fully connected layer takes as input the vocabulary size.

D. The training phase

The training phase is self-explanatory. This is how the algorithms are learning to map the function parameters. This is the most complex step, both computationally and programmatically. This consists of backpropagation the data through the algorithm several times and requires some parameters to be set up and some functions to be chosen. The logical steps that take place are:

- Feature extraction through the CNN encoder;
- Passing the encoder output, the hidden state initialized to 0 (zero) and the decoder input (the "start" token) to the decoder;
- In a loop, the hidden state of the decoder is passed back into the model and its predictions are used to calculate the loss;
- Calculating the gradient, applying it to the optimizer, and backpropagation it through the algorithm.

For training, a set of 40000 entries was used, being split by the 80:20 ratio; 80% for training, and 20% for testing or evaluating.

If the dataset is small enough it can be passed to the algorithm in one batch, in its entirety, but the dataset that is being used is quite large and it requires batching. Batching implies splitting the dataset into multiple equal parts and feeding them one at a time to the algorithm. Since the batches have to be equal, keeping proportionality between the number of instances in the dataset and the batches number is a must. The batch size used is 64.

The loss function chosen is cross-entropy with logits. It is initialized to zero before passing each batch to calculate the individual error values for each batch. The total loss is initialized to zero for each epoch. By plotting these values after the finalization of the training process we can see if the model converged correctly.

Deep neural networks "learn" mapping a function from the data they receive (inputs) to their outputs. This is calculated by running the dataset several times through the models, trying to reduce the error (loss), by setting the number of epochs. The larger the number of epochs, the more time it takes for the algorithm to train, but it also raises the chances of resulting in a better map function. The algorithm was trained on 2 and 20 epochs and the evolution of the loss function in both cases is depicted below:

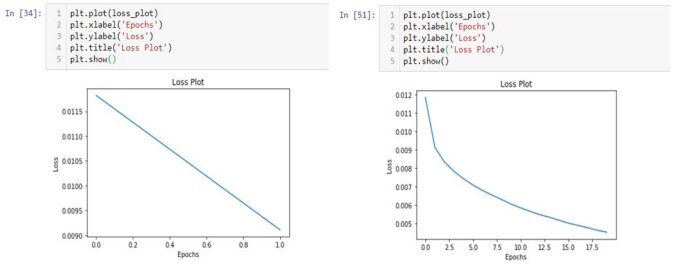


Fig. 3 Loss evolution on 2 epochs and Loss evolution on 20 epochs

One can notice the significant improvement, upon viewing the plots, given by the increase in the number of times the data has been fed through the algorithm.

During the training process, one has to save the weights of the model for further use. The training process is lengthy and all one is interested in are the final weights or the weights from the most efficient computation. These are saved after every 100 batches to ensure a proper back-up. The model can be saved in various formats, the most popular being a "pickle". In this project, the checkpoints are saved using the "checkpoint" attribute, of type index.

After analyzing the loss plots, one can conclude that the model should give good results, since it evolved smoothly, at a logarithmic pace, but this is not always the case. The way to properly test the results is by presenting the algorithm with new, unseen data points; in this case, images.

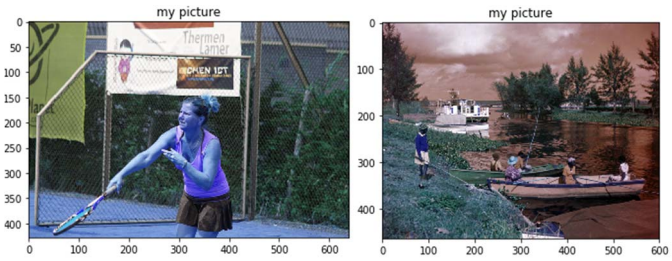
The duration of training the models on 32000 examples, for 20 epochs was about 26 hours, on a Hexa-core CPU.

E. Results

Unfortunately, the device the models are trained on does not have a lot of computing power and the training for a relatively low number of epochs took a lot of time. To be able to do this, the input dataset size has to be decreased. In this case, the input number of images is decreased to 80, also split into training and testing datasets by the 80:20 percent ratio. This leads to 64 images used for training. By choosing this proportionality, one can keep the other parameters, as the batch size. In other words, this is the minimum amount of data one

can use without decreasing the other parameters. One has to take it "one possibility at a time" and exclude each parameter as being faulty.

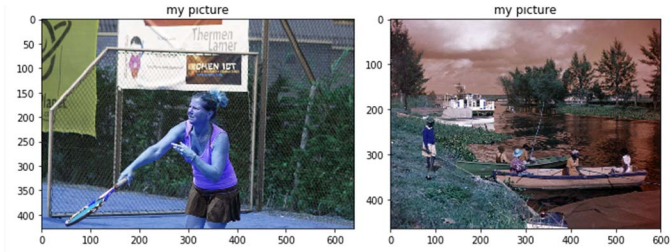
After reducing the number of training data, the number of remains 20, printing the actual and predicted captions. This is just a way to check that the algorithm still works, without waiting for a long period to notice it. To keep in a sanity check, two images are printed to see how the captions evolve for different types of images, in time. One of the images depicts a woman playing tennis, while the other depicts people in boats near a river. The first one should be considered easy to interpret, while the second is a bit more difficult because of the number of elements it contains. The results are depicted below:




```
Image 1: C:\Users\test_ml\imgs\COCO_train2014_000000001375.jpg
Actual caption: a tennis player swings her tennis racquet on the court
Predicted caption:  a <end>
Image 2: C:\Users\test_ml\imgs\COCO_train2014_000000001453.jpg
Actual caption: a boat of women are pushing off of the shore
Predicted caption:  a <end>
```

Fig. 4 Results of image number 1 and 2 after 20 epochs

The results are terrible, but the algorithm work. Next, the number of epochs is increased to 50. The results, presented below, should improve, but not by much.



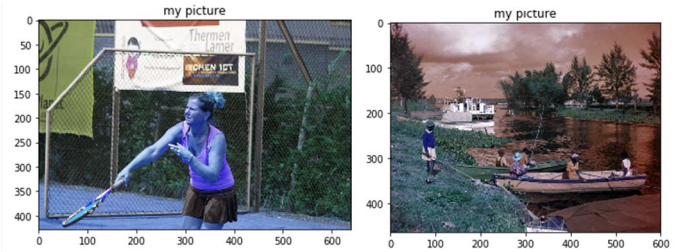

```
Image 1: C:\Users\test_ml\imgs\COCO_train2014_000000001375.jpg
Actual caption: a tennis player swings her tennis racquet on the court
Predicted caption:  a ball on a ball on a ball on a ball on a ball on
                    a ball on a ball on a ball on a ball on a ball on
                    a ball on a ball on a ball on a

Image 2: C:\Users\test_ml\imgs\COCO_train2014_000000001453.jpg
Actual caption: a boat of women are pushing off of the shore
Predicted caption:  a tv in a tv in a tv in a tv in a tv in a
                    tv in a tv in a tv in a tv in a tv in a tv
                    in a tv in a  tv in a
```

Fig. 5 Results of image number 1 and 2 after 50 epochs

The prediction of the first image improved, identifying the ball, while the second image's prediction is off. This makes it obvious that the models were not trained over enough iterations.

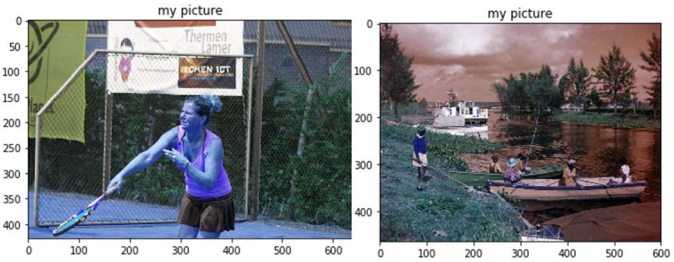
The next step is to see, even for a reduced number of input images, whether the models will ever converge. For this, the number of epochs is increased to 100 and the results are presented below:




```
Image 1: C:\Users\test_ml\imgs\COCO_train2014_000000001375.jpg
Actual caption: a tennis player swings her tennis racquet on the court
Predicted caption:  a person hitting a ball on a tennis court <end>
Image 2: C:\Users\test_ml\imgs\COCO_train2014_000000001453.jpg
Actual caption: a boat of women are pushing off of the shore
Predicted caption:  a big white in a single in the middle of a bottle <end>
```

Fig. 6 Results of image number 1 and 2 after 100 epochs

After 100 epochs, the prediction for image 1 is better than expected and can be considered a good prediction. If one was only to judge by that image, the conclusion would be that the models converged and the number of epochs is enough, but if one looks at the second image, it is noticeable that the prediction is still not satisfying, although it did identify the white boat ("a big white" – figure 6). Since this is a dummy dataset, i.e. it has nowhere near enough images to make a generalized enough model, one can increase the number of epochs to 200 to see how the model acts on complex images. The result expected is for the first image's predicted caption not to change, but the second one to get more accurate. The results are depicted below:




```
Image 1: C:\Users\test_ml\imgs\COCO_train2014_000000001375.jpg
Actual caption: a tennis player swings her tennis racquet on the court
Predicted caption:  a person hitting a ball on a tennis court <end>
Image 2: C:\Users\test_ml\imgs\COCO_train2014_000000001453.jpg
Actual caption: a boat of women are pushing off of the shore
Predicted caption:  a red many one in the middle of a surfboard <end>
```

Fig. 7 Results of image number 1 and 2 after 200 epochs

As expected, the predicted caption of the first image did not change, but the result for the second image shifted its focus from the white boat to the tiny boats. In a way, it improved since the paddleboards in this picture resemble a surfboard, and it does have something in the middle of it (people). Although the caption got better, it is still not close enough of being ideal. One can conclude that this is an outlier, given that no algorithm will ever correctly identify everything. Here, the poor result can be blamed on the quality of the input image.

After tuning the parameters, one can conclude that the number of iterations for the initial training was too low, so the algorithm never reached convergence. Unfortunately, increasing the number of epochs is a very expensive computational decision, but in this case, it is necessary. After running the model over various numbers of epochs, one can decide upon the value of the epochs. In our case, it should be somewhere between 50 and 100 or a satisfactory result.

IV. Conclusion

The paper aimed to develop an application for automated description of images using deep learning, while also doing thorough research of the field. This prototype has to be further integrated with a text-to-speech engine to reach its goal of helping visually impaired people make some sense of pictures.

The research part of the paper was successful, allowing one to understand the various aspects of both machine learning and deep learning algorithms. The fundamental elements of artificial neural networks were studied, such as feed-forward, backpropagation of the error through the perceptron of a neural network, possible loss function, and activations functions. At the same time, the convolutional neural network architecture, which makes use of the matrix computations and dimensionality reduction to make sense of elements in an image. The recurrent neural network, which is a time-series type of artificial neural network, allowing it to make decisions at each step, concerning the value present in the last step, i.e. it can "memorize" values during computation.

After researching the possible models and architecture, a composite model was selected, consisting of a convolutional neural network, as the encoder. That makes use of transfer learning, so reducing the training time and the computational complexity, and a recurrent neural network, as the decoder, which is the state of the art when it comes to dealing with textual data, especially predicting the next word in a sentence (sequential prediction). Since this classical architecture is not able to keep in memory long sentences, the attention mechanism was introduced as an interface between the two models, its output representing the features extracted from the encoder, concatenated with the attention weights.

Initially trained on 32000 images for 20 epochs, despite the good evolution of the loss, the overall model was not behaving as expected, suggesting that one has to adjust the parameters used for tuning. Given that the hardware at hand would not allow for much more complex and time-consuming operations, the parameter tuning was done on a training set of 64 images, to identify the issue. The problem was that the algorithm was not trained over enough epochs and did not converge yet. After training it for 100 epochs, the model converged, despite the minuscule dataset used. The conclusion is that it behaves normally, doing well at predicting the captions of some images, and not so well with others.

A good conclusion when it comes to overall development and training is that it can be done even if the hardware is not performant, at least for prototyping, as depicted in the "Parameter tuning".

Being a prototype, this application can be improved in several ways. The first step that should be taken is to obtain a more powerful device to train the models on. This would affect the time needed for the model to train, thus making it faster and allowing the developer to take further steps in improving the model's performance (in terms of predictions).

To take it one step further, instead of captioning images, the model can be changed to caption videos, which are multiple frames per second. This would be useful for creating more unstructured textual data that can be used to summarize videos, such as the description of movies.

References

[1] J. Anderson, L. Rainie, "Artificial Intelligence and the Future of Humans". Pew Research Center, 2018. Available online: https://www.pewresearch.org/internet/2018/12/10/artificial-intelligence-and-the-future-of-humans/.

[2] O. Apoorva, Y.M. Sainath, G.M. Rao, "Deep Learning for Intelligent Exploration of Image Details". International Journal of Computer Applications Technology and Research Volume 6–Issue 7, 333-337, 2017, ISSN:-2319–8656.

[3] F. Chollet, "The limitations of deep learning", Deep Learning with Python, Section 2, Chapter 9, Essays, 2017.

[4] K. Adnan, R. Akbar, "An analytical study of information extraction from unstructured and multidimensional big data". Journal of Big Data 6, No.91, 2019. Doi:10.1186/s40537-019-0254-8.

[5] O. Vinyals, A. Toshev, S. Bengio et al, "Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, Issue 4, 2017. DOI: 10.1109/TPAMI.2016.2587640.

[6] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, L. Zhang, "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6077-6086, 2018.

[7] S. Shabir, S.Y. Arafat, et al, "An image conveys a message: A brief survey on image description generation". 1st International Conference on Power, Energy, and Smart Grid (ICPESG), 2018. DOI: 10.1109/ICPESG.2018.8384519.

[8] Q. You, H. Jin, J. Luo, "Image Captioning at Will: A Versatile Scheme for Effectively Injecting Sentiments into Image Descriptions". Computer Science, 2018. Available online: https://arxiv.org/abs/1801.10121.

[9] P.P. Khaing, M.T. Yu, "Attention-Based Deep Learning Model for Image Captioning: A Comparative Study". International Journal of Image, Graphics and Signal Processing, Vol. 6, pp. 1-8, 2019.

[10] J. Guan, E. Wang, "Repeated review based image captioning for image evidence review". Signal Processing: Image Communication, Vol. 63, pp. 141-148, 2018.

[11] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C.L. Zitnick, P. Dollár, "Microsoft COCO: Common Objects in Context". Computer Science, 2015. Available online: arXiv:1405.0312.

[12] The COCO Dataset Website, Available online: http://cocodataset.org/#home.

[13] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate". Computer Science, 2015. Available online: https://arxiv.org/abs/1409.0473.

[14] Y. Wu et al, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". Computer Science, 2016. Available online: https://arxiv.org/abs/1609.08144.