DreamEyes - A smart cap to improve visually-impaired individuals' experience in their surrounding environment.

Saturday 18th November, 2023 - 14:17

Bacem Etteib
University of Luxembourg
Email: bacem.etteib.001@student.uni.lu

This report has been produced under the supervision of:
Giacomo DI TOLLO
University of Luxembourg
Email: giacomo.ditollo@ext.uni.lu

1. Introduction

How could deep learning generate a narrative description of the surrounding environment?

As of 2021, the World Health Organization (WHO) estimates that there are approximately 36 million visually impaired individuals worldwide[1]. This disability can significantly impact their ability to perform daily tasks and engage in social and professional activities. However, visually impaired individuals have the same potential as those with normal vision and can be just as productive. Therefore, it is crucial to take advantage of technology and advancements in assisting devices to create tools to support them.

Recent developments in the AI industry, particularly in deep learning, have spurred researchers to explore the creation of such assisting devices. **DreamEyes** is an AI-based project that aims to translate real-world data into rich and understandable paragraphs that will be converted into audio format using deep learning models. This will greatly contribute to improving the interaction of visually impaired individuals with their surrounding environment. The prototype consists of a wearable "smart" cap equipped with Internet of Things (IoT) components such as microprocessors that analyze real-world data and inform the user of what is happening around them. The algorithms used in this project are inspired by the learning mechanism of the human brain and are based on deep learning, a subset of machine learning where deep neural networks learn and make decisions from complex and large data sets.

DreamEyes is designed to replace traditional solutions that are limited in their technology and features. This device will allow individuals with partial to full vision loss to communicate with the external world and have an enhanced experience of their surroundings. In this report, we will focus on describing the implemented methods and approaches followed during the design as well the development of this contribution. The scientific part will present a descriptive top-down approach of deep learning and the sequence models architecture whereas the technical part will cover the development and deployment stages of the program.

The document is structured as follows:
A scientific design section depicting the included references and the techniques derived from them to conceptually design the assistive tool. In addition, we include a production section where we depict the main concepts behind the creation of DreamEyes. Examples of the concepts to be covered are video captioning and encoder-decoder structure. Finally, we add an assessment section to evaluate the quality of the developed scineitic approaches and to provide a feedback of the first part of this report.

In the second part of the report, we follow the main structure to shed the lights on the technical aspect of the developed software. Therefore, we provide comprehensive answers to the following questions: How was the software designed? Which approaches have been followed in the production phase? Finally, we conclude this part with an evaluation section where we explore the summary of the final results and assess the completion of the project's goals.

2. Project Description

For a human being, the process of understanding what is occurring in a real life scene is a straightforward task. Nevertheless, a computer is unable to figure out the visual elements appearing in an image, or in an ordered sequence of images. Through the last years, efforts of AI researchers

and high performing hardware companies (e.g., Nvidia) have been combined to create state-of-the-art solutions to Computer Vision (CV) problems. This huge advancement in the field of CV has allowed for more state-of-the-art solutions to real life challenging problems. One category that could extremely benefit from this development and flourishing of AI is people with visual-impairment. As computers could be taught to recognize objects more accurately, life-changing assistive systems could be designed to help this category of people to experience the world in a better way.

In fact, a machine would only be able to recognize elements in an image if it has been previously trained to do so. Therefore, we investigate modem methods and techniques used to introduce computers a way of 'thinking' into an image. One interesting and very common approach is to use a type of Neural Networks called Convolutional Neural Networks[3]. In his breakthrough research paper, Yann LeCun, chief AI scientist at Facebook, introduces the structure of CNNs and their extreme efficiency in achieving accurate outcomes in Vision Applications. At this point, the number of projects built on top of CNNs has drastically increased and more changes to the base architecture of the model were made. Some specific model architectures used in global competitions hosted by big high-tech companies such as Google and Kaggle have been made public due to their high performance and outstanding results. For example, Alex Krizhevsky et al. introduced in their research paper "ImageNet Classification with Deep Con-volutional Neural Networks"[4] the AlexNet model, which is a pre-trained deep convolutional neural network that achieved state-of-the-art performance on the ImageNet image classification task.

Pre-trained models are a popular technique in computer vision for achieving state-of-the-art performance on various tasks, such as object detection, image classification, and semantic segmentation. These models are pre-trained on a large dataset, typically millions of images, to learn general features that can be applied to new, unseen images. The use of pre-trained models has become prevalent due to the significant improvements in performance they offer, as well as their ease of use and adaptability to various tasks. One of the most widely used pre-trained models in computer vision is the ImageNet dataset, which consists of over 1.2 million images belonging to 1,000 different classes. The ImageNet dataset was used to train the famous VGGNet, ResNet, and Inception models, which have been shown to outperform previous state-of-the-art models in various tasks. In recent years, there has been a surge of research aimed at improving pre-trained models, including the development of larger and more complex models such as EfficientNet, which achieved state-of-the-art results on ImageNet with significantly fewer parameters than previous models. Other research has focused on training models on larger and more diverse datasets, such as the JFT-300M dataset, which consists of 300 million images and has been used to train the T5-XXL model. Pre-trained models have become an essential tool in computer vision, enabling researchers and practitioners to achieve state-of-the-art results with minimal effort. The use of pre-trained models is likely to continue to grow, as new and more diverse datasets become available and the models themselves continue to improve.

2.1. Domains

We, hereby, explore the two main domains of our work. In the first part, we talk about the scientific study of the project and the related methodology. In the second part, we provide an abstract depiction of the technical domains.

2.1.1. Scientific

. The scientific domains involved in this work are mainly the creation of a well-structured scientific report that involves multiple references to similar researches done by qualified individuals or intuitions. That being said, DreamEyes was built on top of an immense study and research of trending computer science topics and fields that are mainly Artificial Intelligence and the usage of encoder-decoder architecture (section 4.2). Therefore, we explore in details the definitions of those concepts, their applications and more importantly their relevance to our project.

2.1.2. Technical

. The technical domains are those involved directly during the production of the desired prototype. As explained above, multiple researches and studies of available resources related to the scientific domains were done in order to break down some the sophisticated concepts such as video captioning. The understanding of this term, requires a recursive process of defining previous and more general terms such as Computer Vision and Artificial Intelligence. However, concerning the technical domains, we restrict them to the usage of the Python programming language, a bunch of libraries that come with (section 5.2.1) and finally deep learning algorithms deployment (importing, training and testing).

2.2. Targeted Deliverables

At this level, we dive into the composition of the two types of deliverables necessary for the development of this work. We first look into what is meant by a scientific work and the criteria used during this report to ensure a high quality and well-defined (according to scientific principles) work. Later, we cover, using a synthetic description, the technical deliverable and its different aspects manifested, at a final stage, in the prototype.

2.2.1. Scientific deliverables

. The scientific study of the project DreamEyes involves different stages of topic research and methodologies investigation. This part explains the principal concepts involved in the

development of our proposed solution. During our research, we follow a top-down approach, from explaining global and broad concepts such as Artificial Intelligence and Computer Vision to more detailed terms like models architectures and training process. The presented structure of our project DreamEyes is supported by high-level research papers covering similar proposed systems and having precise, sometimes state-of-the-art definitions of the deployed techniques. Through the scientific section of the paper, we break the sophisticated scientific concepts into connected subsections for better understanding

2.2.2. Technical deliverables

. At this phase, we focus on delivering a well-structured and technically accurate description of the approaches followed during the coding and development of DreamEyes. Sections Design and Production cover the necessary information and methodologies required to understand how we converted from the scientific depiction to the deployment phase. Moreover, concepts like Convolutional Neural Networks and transformers, covered in the first deliverable of the project, are now approached differently, in terms of technical implementation. We provide a depiction of the different techniques used from data loading/preparation to models training and testing. As we are working with Python, we do also include snippets of code related to the main functionalities in the programming part.

3. Pre-requisites

The document is split mainly into two parts: Scientific and Technical study. Before diving into the details of each section, we require a basic understanding of some concepts so that it would be easier to easily grasp and deploy the content of this report.

3.1. Scientific pre-requisites

Concerning this part of the work, the reader needs to be able to locate and analyze the content of the provided references easily. Besides, a basic understanding of computing fundamentals (e.g, algorithms) , basic linear algebra and an overview about Artificial Intelligence is required.

3.2. Technical pre-requisites

For the technical part to be easily understood, the reader needs to be familiar with the python programming language and its applications. The python language is a high-level programming language providing understandable code. In addition, competency in Unix operating systems, terminal usage, libraries importing and installing are required.

4. How could we use deep learning to generate captions for real life scenes?

4.1. Requirements

In order to correctly understand the scientific question, there are some key concepts we should talk about:

- Functional requirements

After conducting a research on AI and specifically deep learning, using the references included at the end of the document, we conclude that the answer to the scientific question is related to the understanding of video captioning and the concept of encoder-decoder architecture (section 4.2). Thus, we ensure to present a scientifc work, accessible by researchers and individuals interested in this domain, that would be well-defined and easy to follow. It is, therefore, primitive to provide scientific defintions and explanation for the major terms involved and the followed methodologies.

- Non-functional requirements

The answer to the scientific question: How could we use deep learning for captions generation of real life scenes?, requires multiple researches and studies done on the main concept of the question: deep learning. Thus, we ensure to provide a well-structured work, according to scientific principles, with defined concepts and relevant citations.

4.2. Design

In our work, we have investigated the possibilities of training a model from scratch to recognize elements in a given image and using a pre-trained model as an alternative approach. After studying the two options and evaluating the available resources and project's complexity, we arrived at a conclusion that the problem would be better tackled using a pre-trained mode. Thus, we build on the foundation of pre-trained models to develop a novel system that aims to assist blind individuals in comprehending visual information. By utilizing pre-trained models for object detection and scene understanding, our system generates natural language descriptions of the environment in real-time, providing users with a comprehensive understanding of their surroundings. The use of pre-trained models has allowed us to leverage the vast amounts of data and computational resources necessary to train these models, reducing the time and effort required to build a robust system. Furthermore, the adaptability of pre-trained models to different tasks and datasets has enabled us to fine-tune these models for our specific application, resulting in more accurate and relevant image descriptions for the visually impaired. Thus, we make use of ResNet50[5] for extracting features from images. The scope of functioning of this specific architecture is covered in the production section.

4.3. Production

4.3.1. Neural Networks, architecture and training process.

4.3.1.1. Artificial Neural Networks

. Neural Networks, also referred to as Artificial Neural Networks(ANN), are a type of machine learning algorithm that is designed in a way to simulate the human brain functioning mechanism. Precisely, ANNs are a set of

interconnected layers, each with a determined number of nodes, that map an input X into an output Y. The process of 'teaching' the networks how to make predictions based on data is called training. The training process involves introducing large datasets to the model which would break them into patterns and relationships. The evaluation process consists of an assessment of the model's performance based on specific metrics such as accuracy and loss. At this stage, the behavior of the model on the validation data set is studied and conclusions such as overfitting and underfitting could be made. In simple terms, overfitting corresponds to the state where the model tends to memorize the data rather than learning the patterns. On the other hand, underfitting happens when the model architecture is too simple to handle the complex relationships in the data.

At a technical level, the first block of a neural network is called the input layer. As the name suggests, this layer is responsible for reading the input data. The next block of layers is called the hidden layers. At this level, data gets transformed into a more complex representation using matrix multiplication and basic mathematical operations. This data transformation is mainly based on a multiplication by random weights at each node, and an addition of a constant value called 'bias'.

Weights and bias are called parameters of the network and they are learnt along the training process. Particularly, weights represent the strength of the connections between the neurons while bias is a constant value that skews the result of the algorithm. The weighted sum is then introduced to an activation function that is responsible for introducing non-linearity and thus allowing the model to capture more complex data patterns. As randomness is introduced to the model, it needs to adjust the values to make accurate predictions.

After performing the forward propagation, the model now calculates the error function (e.g absolute value of the difference between predicted output and actual output) and applies mathematical approaches such as Gradient Descent to minimize it and hence to get more accurate results. The outcome of this process is, afterwards, transferred to the output layer and interpreted in the right format. For example, although the task of recognizing a digit might be trivial for humans, computers will need to use neural networks (mainly CNNs) to understand which digit is introduced.

4.3.1.2. Convolutional Neural Networks

. Artificial Intelligence is flourishing by leaps and bounds, this huge progress have inspired researchers to introduce more complexity to the ANNs architecture. In the late 1980s and early 1990s, Yann Lecunn introduced in his paper 'Gradient-based learning applied to document recognition', Convolutional Neural Networks[2] and demonstrated their ability and high accuracy in predicting handwritten digits.

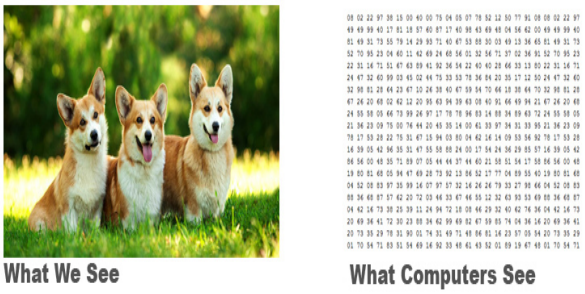

Fig. 1. Image for a machine (videonet9.com)

To better understand the architecture of a CNN, we look into how a computer interprets an image. Obviously, unlike a human brain, the machine needs to break an image into digital information. For instance, a computer would interpret an image as a two-dimensional array of pixels colors' values. A pixel refers to a small region in the image and in case of a colored image, the color value is represented using the RGB color model. This model consists of three main channels: Red, Green and Blue. Most colors could be created based on those primary channels. The value range is between 0 and 255 for each pixel in the RGB model. An example would be (0,0,255) representing blue color.

In the perspective of a CNN, we will focus on this array to perform computer vision operations. Hereby, we thoroughly investigate the role of each layer:

- **Convolutional Layer**: This layer is considered the base layer for the CNN architecture. At this part, most operations on the input vector are performed. If we take, for the sake of breaking down the mechanism of the CNN layer, an RGB image, the input which will consist of a three-dimensional matrix will undergo the application of a filter, also known as kernel that moves among the matrix values and detects the presence of features using what's known in mathematics as the convolution operations.

  The kernel is usually an n*n matrix of dimensions varying from 3*3 to 5*5. Once the filter moves across all the pixel cases and the mathematical operations take place, we get in a result a new output matrix referred to as the feature map. The volume size of this feature map depends on the number of filters used, the number of pixels operated by the kernel and the padding.

- **Pooling Layer**: Down sampling or dimensionality reduction is introduced in this layer. As a result, there will be fewer parameters to learn, which will also reduce computational complexity. Different methods of matrix reduction demonstration could be used, depending on the kind of pooling layer. For instance, the max pooling

operation will select the area of the feature map with the highest pixel value.

- **Fully Connected Layer**: One or more convolutional layers and pooling layers are followed by the fully connected layer. The output of the preceding levels is vectorized and fed into the FC. The fully linked layer, which stands in for the output layer, transforms the abstract features into labels and classes. It consists of interconnected nodes that execute categorization based on previously acquired features, with each node depending on the activation of the previous one. The FC layer often employs a softmax function, which converts an input value to 0 or 1, in order to appropriately identify the output.

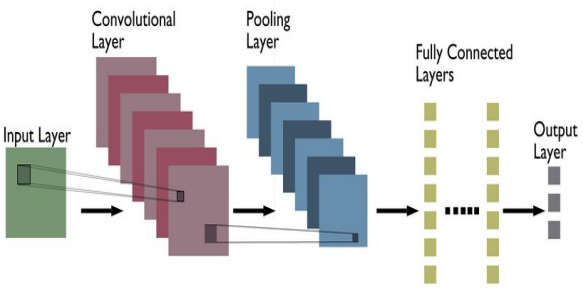

Fig. 2. CNN architecture (researchgate.com)

Overall, Convolutional Neural Networks have proven their efficiency in solving computer vision problems and they could even be combined with sequence models to approach more complex problems such as video captioning.

4.3.1.3. Sequence models: LSTMs and RNNs

. Sequence models are a type of machine learning model that is designed to handle sequences of data, such as text, audio, or video. One of the key advantages of sequence models is their ability to capture temporal dependencies between the elements in a sequence. In other words, sequence models can learn from the order of the elements in the sequence and use this information to make predictions or generate new sequences.

Recurrent Neural Networks (RNNs) are a type of sequence model that is designed to handle sequences of arbitrary length. RNNs have a hidden state that can capture information from previous time steps in the sequence, allowing them to model long-term dependencies. The hidden state is updated at each time step using the current input and the previous hidden state. This allows the RNN to capture the temporal dependencies between the elements in the sequence. However, RNNs can suffer from the vanishing gradient problem, where the gradients that flow through the network become exponentially small, making it difficult to learn long-term dependencies.

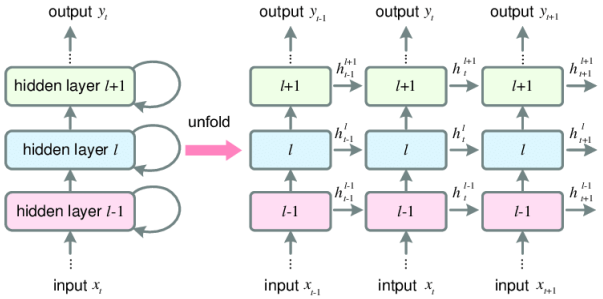

Fig. 3. RNN architecture (researchgate)

Long Short-Term Memory (LSTM) networks were developed to solve this issue. An RNN type called LSTMs has a more intricate construction than a typical RNN. Because LSTMs contain a cell state that persists throughout the entire sequence, the network can choose what information it wants to remember or forget from a previous time step. The input gate, forget gate, and output gate are the three gates that make up an LSTM cell. The amount of data input into the cell state is controlled by the input gate, the amount of data removed from the cell state is controlled by the forget gate, and the amount of data output from the cell state to the following time step is controlled by the output gate. This gating mechanism allows LSTMs to selectively store or discard information from previous time steps, making them well-suited for modeling long-term dependencies in sequences.

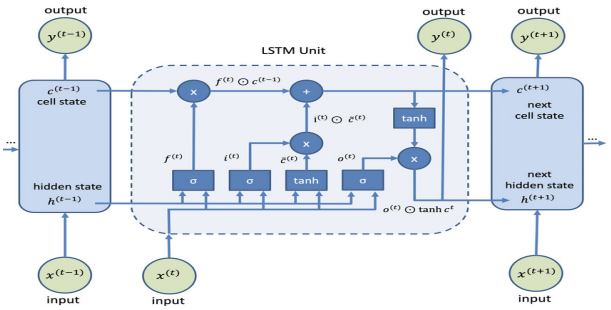

Fig. 4. LSTM architecture (researchgate)

An encoder and a decoder are the two parts of an encoder-decoder architecture, a type of sequence model. The encoder processes the input sequence to produce a fixed-length representation of the input, which is then provided to the decoder. The decoder then uses this representation to create the output sequence. In an LSTM-based encoder-decoder architecture, the input sequence (in this case, the video frames) would be fed into the LSTM-based encoder one frame at a time. The LSTM would analyze each frame before producing a fixed-length representation of the scene. After receiving this representation, the decoder would provide a caption for the scene. The decoder would then receive this representation and provide a caption for the scene. The

decoder could be another LSTM-based network that generates the caption one word at a time. At each time step, the decoder would receive the fixed-length representation of the scene and the previously generated word, and would output the next word in the caption.

An LSTM-based encoder-decoder architecture that can identify the temporal links between video frames can be used to provide a scene description overall. By converting the caption into audio, a person suffering from partiel/total vision-loss could hear a real-time description of the scene. The encoder-decoder architecture with an LSTM-based encoder generates accurate and coherent predictions when processing sequential input.

4.3.2. Video Captioning

. Automatically creating a narrative description of a video's visual content is known as video captioning. It is a crucial work in computer vision and natural language processing with the goal of improving video search and indexing, primarily for content providers, as well as making video content more accessible and understandable to those with hearing or visual impairments.

The two main steps involved in the video captioning process are text generation and video analysis. At a first stage, a high-level representation of the video's visual content that captures its semantic and syntactic structure is examined and encoded. In order to extract pertinent visual information from each frame of the video, convolutional neural networks (CNNs) are frequently used in this stage. These capabilities may include motion analysis, scene identification, and object detection. For instance, the computer vision technique of object detection seeks to separate an image into various named things. Nevertheless, since not every object depicted in a video frame is always pertinent to the context and the scenario, this procedure might differ slightly from picture captioning in the case of video captioning. Additionally, the focus on detecting every single object will result in an increased computational complexity and exhausted resources.

The second stage involves feeding a new class of neural networks called Recurrent Neural Network (RNN) with the features that were retrieved from the video frames. The RNN's job is to record the temporal dynamics of the video and produce a series of feature vectors that correspond to the content of the video. As a result, RNN will be employed as a language model and will accept the produced sequence of feature vectors as an input. A narrative caption for the video is produced by this RNN using the encoded visual features. The generated description is usually made up of a collection of words or phrases that meaningfully describe what is happening in the video frames. The language model can be trained on a large dataset containing information regarding video frames, their characteristics and the corresponding captions in order to learn the statistical patterns and structures of natural language.

Overall, computer vision and natural language processing techniques could be, together, used in order to successfully break down the challenging task of video captioning. This technique has numerous potential uses in fields like assistance and education. Additionally, recent developments in deep learning have significantly increased the reliability and accuracy of video captioning systems, enhancing their use and dependability in real-world situations.

As we have mentioned, the two major approaches in video captioning are template-based language models which create templates for sentence generation while highlighting grammar rules and syntactical structure, and sequence learning methods (e.g RNNs) that use sequence learning algorithms to convert the video content into a phrase. Thus, we hereby introduce the encoder-decoder architecture, a well-known term in the world of deep learning where we map sequences-to-sequences with different lengths (e.g translation tasks).
One important aspect about this architecture is that the final state of the encoder is used as the initial state of the encoder.

- **Encoder**: encoding refers to the conversion of data from one format to another required one. In our case, the conversion happens on the input video sequence into a fixed-dimension feature vector. We often make use of LSTMs at this stage.
- **Decoder**: It will generate the output sequence or more precisely captions based on the input of the encoder. Since captions are also a set of words, we will make use of LSTMs at this stage as well.

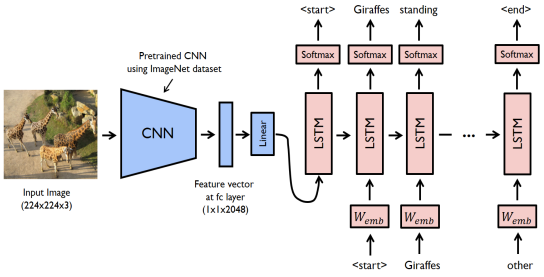

Fig. 5. Encoder Decoder (Analytics Vidhya)

4.3.3. Answer to the scientific question

. Since the main usage of technologies, nowadays, is to address major issues and help people overcome obstacles preventing them from experiencing a relatively comfortable life and achieving their daily tasks in an easy way, we conclude the importance to deploy one manifestation of technology for the sake of individuals suffering from partial or complete vision-loss. Thus, we conduct a scientific research and we end up

by choosing a subfield of AI called Computer Vision as our main field of study. The intensive researches done on this topic have lead to creating a recursive set of definitions for related concepts such as Neural Networks and other deep learning architectures. Afterwards, we put the gathered knowledge into usage and we try to develop, using a well-defined technical approach, a software simulating the proposed solution.

4.4. Assessment

The project's main goal is to provide a rich narrative description of the surrounding environment for the visually-impaired people. During our followed approach in the scientific part, we tried to cover the necessary terms and concepts required to break this task into smaller parts. By using high-quality references, including state-of-the-art research papers and articles, we come up with a detailed description of the most relevant topics: Video Captioning and the CNN architecture. In terms of accuracy, the provided approach should be working well with simple to moderately-complicated images. That means, images that include complex relations between items and invisible elements would be hard to describe accurately, sometimes leading to false generated descriptions. By looking at the most recent work done on this field, we notice the existence of a modern approach, based on video captioning but with an addition of what's called Sparse Attention. Unlike the most common solution that focuses on temporal features, the recent work gives more credit to the attention over space and time[8]

5. Technical description

The technical deliverable of this project will consist of a semi-automatic, interactive system built on top of an encoder-decoder architecture with the goal of generating descriptive text of a given image. The generated captions could be aggregated to approach the problem of a video captioning. In a typical use case, a user equipped with a wearable cap would be able to interact with the system to unconditionally request narrative captions of a real-life scene. As this project is primarily designed for individuals with vision difficulties, we note the absence of a user dedicated interface and we use rather vocal commands to control the input/output of the system. The methodologies behind the development of this project involve an intense deployment of deep learning techniques and the usage of pre-trained models, as explained in the scientific part. Concerning the hardware part, we make use of a wearable cap equipped with a micro-controller, typically a RaspberryPi, and headphones to receive commands and communicate the captions to the user. In addition, the main external communication component is a camera, responsible for the capturing of real-life scenes.

Due to its high processing power, adaptability, and affordability, the Raspberry Pi micro-controller[6] has grown to be a popular choice for implementing deep learning models in embedded systems. The Raspberry Pi is an excellent option for an image captioning system for those who are visually impaired since it can analyse image inputs from a camera in real-time and run sophisticated deep learning algorithms. The Raspberry Pi's deep learning model receives an image from the camera and processes it to produce a written description that is then played back to the user. As a result, we could divide the system's functionalities into three primary categories: input capturing, image processing, and creation of captions, and lastly output communication. Each of the functions is dependent on the outcome of the previous one.

5.1. Requirements

The software is dependent on the user input. In fact, this input is gathered automatically using the installed camera on top of the device. That being said, as we design this prototype for individuals suffering from visual-loss, we expect no major manual interaction with the user. However, we may introduce voice commands for easier communication between the two parts. For the software to be run accurately, we require the successful execution of a set of functions representing the core of our prototype. The first function consists, in short terms, of the conversion of captured the real life scenes into a stack of a number n of images. The format as well as the length of the sequence of those images are pre-defined (due to software limitations). Afterwards, we involve steps of input pre-processing to ensure that the input meets the right requirements of the model (size, color..). Finally, we call the model instance, feed it with the obtained set of images and transform the resulting captions into speech using the neccessary python libraries.

- Functional requirements

They could be summarized in the following points: The ability of the tool to capture and analyze real-life scenes without complications; The generation of the corresponding captions with respect to the presented input and finally the text-to-speech function that converts the obtained text to sound.

- Non-Functional requirements

The main concern at this point is reliability of the tool, usability and accessibility. Hence, the deliverable is set to generate relatively accurate results (model predictions always prone to errors). Besides, the time delay between the moment of scenes capturing and the generation of the corresponding caption is dependant on the resources computational power (limited memory).

5.2. Design

The conception of the system was done initially using a black box approach. We first denote the need of detecting elements in an image, analyze their relationships and extract the most important features among them. Afterwards, given the features vector, we generate appropriate captions that will represent the narrative description of the visuals. At a higher level, we conclude that this first task requires the

usage of a deep learning model to approach the mentioned steps. We start designing a function, name input processing, where we suppose the existence of an already processed input image/video and we code the processing and modelling phases. At this stage, after performing several researches summed up in the scientific part, we decide on using the encoder-decoder architecture. Hence, we experiment the existing methods to deploy this model architecture in python and we end up using a bunch of libraries designed mainly for the task of image captioning. Hereby, we explain, in detail, their implementation in the project.

5.2.1. Main Libraries:.

5.2.1.1. PyTorch

. Machine learning models can be trained, assessed, and deployed using a variety of tools and capabilities provided by PyTorch. With PyTorch, programmers can quickly create and implement unique deep learning models or use pre-trained models that have already been trained on huge datasets. Furthermore, PyTorch offers a set of tools for refining already-trained models, allowing developers to alter the models to better suit their unique use cases. PyTorch makes it simple to adjust hyperparameters[7] during the training process, including learning rates, optimizer settings, and loss functions, allowing for quick experimentation and model improvement.

5.2.1.2. Numpy

. Numpy can be used to process word embeddings or image features in the context of creating captions. These are crucial inputs for training deep learning models. The training process is substantially sped up by its efficient multidimensional array storage and computation capabilities, which enable batch processing and vectorized operations. Numpy also makes it easier to handle data transformations such producing training/validation batches, padding sequences, and one-hot encoding. Numpy is a crucial tool for manipulating and processing data in caption generation models, improving their training efficiency and efficacy, thanks to its vast capability and speed optimizations.

5.2.1.3. BLIP decoder

. The Blip model, a cutting-edge architecture created especially for natural language synthesis applications like image captioning, is implemented via the blip decoder package. Recurrent neural networks (RNNs) and attention processes are some of the deep learning strategies that this model uses to efficiently produce captions that accurately describe images.

We were able to easily include the Blip model into our picture captioning system by utilizing the blip decoder package. The pre-trained weights and effective inference techniques provided by the library enabled us to produce captions with little processing overhead and high-quality results.

The blip decoder library also offered adaptable options for perfecting the model on our particular dataset. Due to this adaptability, we were able to customize the caption generating process to the particulars of our image data.

5.2.2. Data Collection

. Data collection is among the most important steps during the development of a deep learning based project. It plays a crucial role in the final results and is most considered one of the hardest parts in the process of model development due to factors such as missing data, absence of annotation or total lack of relevant instances for the desired topic. In our work, we heavily rely on two data resources to train the model: COCO (Common Objects in Context) and NoCaps.

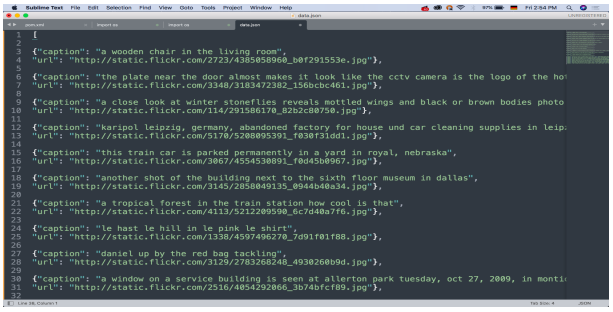

Fig. 6. data in JSON format

The COCO dataset[8] is public, widely used and well-known in the field of image/video captioning. As it contains over 1.5 million human-generated captions for each of the training and validation images, this dataset allows the pre-trained model to maximize its ability in tasks involving object, scenes recognition and contextual relationships identification. On the other hand, NoCaps dataset relies on unpaired images and captions which leverages the model's ability to learn from a wider range of visuals and also generate diversified and creative captions.

5.2.3. Algorithm Development

. The model development phase involves different steps that are crucial in creating a functioning deep-learning based model for this task. The first phase includes providing the base model (provided by huggingface) with the collected data points. Afterwards, we initialize the fine-tuning and training steps. This includes mainly looping over image-caption pairs, optimizing hyper parameters using strategies like back-propagation[9], and the model weights are then updated accordingly. After several epochs of looping over the same procedure, the model will be ready and evaluated using metrics such as accuracy and will be ready for deployment. After training, the model is prepared to generate captions. The trained model is incorporated into the captiongeneration.py script, which also offers the ability to create captions for fresh photos. It accepts an image as input, analyses it, and uses the trained model to provide captions.

5.3. Production

The production of the technical deliverable has undergone several steps and phases. At this level, we provide a comprehensive description of the technical aspect of the tool: the code generated, main functionalities and concrete examples to illustrate the work achieved.

5.3.1. Code structure

. For the sake of accessibility and easier control and modification of the code at later stages, we split the main blocks of the program into separate files. Each file contains relevant code for the associated task.

- **datasets.py:** this file contains the code for locating, downloading, initial processing and storing the COCO and NOcaps data sets into the local machine.
- **data-preprocessing.py:** code for main data preprocessing steps on the two provided data sets: COCO and NoCaps. The detailed explanation of the different phases for this process are explained below.
- **training.py:** this part defines the training pipeline followed for the model creation. It includes code for model initialization, hyper-parameters and the loss function.
- **main.py:** main code for the program that includes model loading, data reading and transforming and the call for external modules for prediction generation.

5.3.2. Code snippets

.

5.3.2.1. Data collection

. To manage the loading and processing of data from the COCO and NoCaps datasets, two classes, COCODataset and NoCapsDataset, are defined in this section of the code. The COCODataset class creates picture paths, opens images using PIL, and adds images and captions to a data list after initializing the image directory and annotation file paths. It also reads the annotation file to get annotations.

```
class COCODataset:
    def __init__(self, image_dir, annotation_file):
        self.image_dir = image_dir
        self.annotation_file = annotation_file

    def load_data(self):
        with open(self.annotation_file, "r") as file:
            annotations = json.load(file)["annotations"]

        data = []
        for annotation in annotations:
            image_id = annotation["image_id"]
            caption = annotation["caption"]

            image_path = os.path.join(self.image_dir, f"COCO_train2014_{str(image_id).zfill(12)}.jpg")
            image = Image.open(image_path).convert("RGB")

            data.append((image, caption))

        return data
```

Fig. 7. classes for data collection

The NoCapsDataset class creates picture URLs, downloads images using requests and converts them to RGB format using PIL. Finally, images and captions are added to a data list. It also initializes the annotation file path and reads the annotation file to extract annotations. The load data method of both classes returns the data that has been gathered.

5.3.2.2. Data processing

. The main functions that this snippet of code carries out are as follows: it loads and processes data from the COCO and NoCaps datasets by extracting image paths and captions, preprocesses the captions by tokenizing them using NLTK's "wordtokenize()" function, and saves the preprocessed data as a JSON file.

```
# Preprocess captions
processed_data = []

for item in dataset:
    image_path = item['image_path']
    caption = item['caption']

    # Resize images (if necessary) using libraries like PIL
    image = Image.open(image_path)
    # Perform image resizing operations as needed

    # Tokenize captions using NLTK or Hugging Face Tokenizers
    tokenized_caption = word_tokenize(caption)

    # Add image path and tokenized caption to processed_data
    processed_data.append({
        'image_path': image_path,
        'caption': tokenized_caption
    })

# Save preprocessed data
preprocessed_data_path = os.path.join(data_dir, 'preprocessed_data.json')

with open(preprocessed_data_path, 'w') as file:
    json.dump(processed_data, file)
```

Fig. 8. data processing

For file and data processing, it makes use of the os, json, nltk, and PIL libraries. For operations involving images, it makes use of the Image module from PIL. The program divides the data into the 'coco data' and 'nocaps data' lists, loads the appropriate datasets, combines them into a single dataset list, and then deals with the dataset list's captions and image paths. The preprocessed data is then saved as a JSON file.

5.3.2.3. model training

. The primary functions of this code piece include: It creates the setups and routes required to train an image captioning model. The preprocessed data is loaded from a JSON file, a new dataset is defined using the ImageCaptionDataset class, and a data loader is created for quick data processing.

```
# Define the model
model = BlipForConditionalGeneration(config)
model.to(device)

# Define optimizer and loss function
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss(ignore_index=tokenizer.pad_token_id)

# Training loop
for epoch in range(num_epochs):
    total_loss = 0.0

    for images, captions in dataloader:
        images = images.to(device)
        captions = tokenizer(captions, padding=True, truncation=True, return_tensors="pt").input_ids
        captions = captions.to(device)

        outputs = model(input_ids=images, decoder_input_ids=captions[:, :-1])
        logits = outputs.logits

        loss = criterion(logits.view(-1, logits.size(-1)), captions[:, 1:].contiguous().view(-1))
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    average_loss = total_loss / len(dataloader)
    print(f"Epoch {epoch+1}/{num_epochs} | Loss: {average_loss}")
```

Fig. 9. model training

Additionally, the code initializes the model and optimizer, defines the loss function, and loads a tokenizer and model configuration particular to the Blip image captioning model. The primary training loop performs forward and backward sweeps across the dataset while updating the model parameters

based on computed gradients. The average loss is computed and printed after each epoch. The trained model is then saved to a predetermined output path. The program makes use of PyTorch, including its neural network modules (torch.nn), data loading (torch.utils.data.DataLoader), and optimization (torch.optim), as well as the transforms module from torchvision for image preprocessing.

5.3.2.4. main file

. It imports the essential libraries for text-to-speech conversion, HTTP requests, picture processing, image captioning, and audio playing. A tokenizer and a model for automatic picture captioning are loaded by the code. It defines the URL of the image that needs a caption, opens the image, and changes it to RGB format.

Fig. 10. predicition and speech generation

After that, the image is tokenized and made ready for caption creation. The model creates captions for the picture, which are then decoded and kept in the captions and "caption text" files. It prints the caption text. The code renders the caption text as voice and stores it as an audio file using the gTTS library. Using pygame.mixer, it initializes, loads, and plays the audio file.

5.4. Assessment

During our assessment phase of the generated technical part related to our project, we manually gathered 50 art images related mainly to the musical field. The samples visual complexity varies from easy to understand, to moderately sophisticated representations of musical instruments and figures.

Fig. 11. Some of the images used for assessment

We first start by evaluating the captions' variation. The analysis was conducted on the 50 gathered samples where we create a json file containing an association of the image names and their corresponding caption. Afterwards, thanks to an approach developed by a fellow student for pair-wise similarities detection, we apply a form of cosine similarity to evaluate the caption's similarity. The output of the algorithm is a range of scores varying from -1 (no correlation) to 1 (same caption). Overall, the model was often using the same words for musical instruments and gender identification in the images. Besides, as we use a limited number of vocabulary during the training process, we notice some similarity between the generated captions.

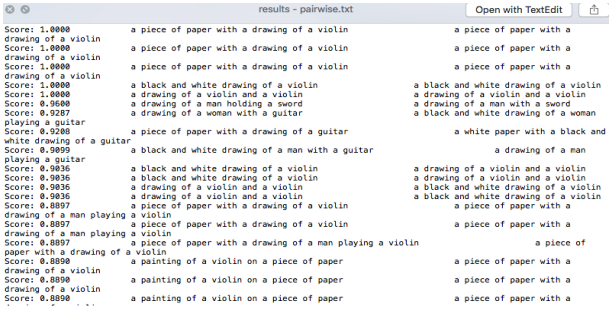

Fig. 12. Pairwise captions similarities scores

At a second stage, we compare our approach to the fellow student's(Antonia) approach for image captioning. The two approaches, being different in the deployed algorithms, generate slightly correlated captions. We note that the technical algorithm used by Antonia relies on sentence transformers to evaluate Semantic Textual Similarity.

5.5. Conclusion

Driven by the need of finding possible implementations of modern technologies to help assist people with disabilities and offer them the chance to experience the world in a more comfortable way, we conducted intensive scientific research about the existing solutions designed for people suffering from partial or total vision-loss. Therefore, we concluded the need of deploying a deep learning based model to help create a tool that will transform real life scenes into narrative descriptions and generate speech from them. In the first part of the report, we explored the main scientific methodologies behind the design, development and production of this tool. In the second part, we looked into ways of generating associated code for the conducted researches and therefore we relied heavily on python and its deep learning packages. Overall, this work was a demonstration of the flourishing of the AI field and its ability to tackle challenging problems in a very smooth and relatively accurate way.

6. Acknowledgment

The authors would like to thank the BiCS management and the education team for the amazing work done.

7. Plagiarism statement

This 350 words section without this first paragraph must be included in the submitted report and placed after the conclusion. This section is not counting in the total words quantity.

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a precheck by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as ones own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:

    1) Not putting quotation marks around a quote from another persons work
    2) Pretending to paraphrase while in fact quoting
    3) Citing incorrectly or incompletely
    4) Failing to cite the source of a quoted or paraphrased work
    5) Copying/reproducing sections of another persons work without acknowledging the source
    6) Paraphrasing another persons work without acknowledging the source
    7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in ones own name(ghost-writing)
    8) Using another persons unpublished work without attribution and permission (stealing)
    9) Presenting a piece of work as ones own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced.

    Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] ”Blindness and Vision impairment”. Fact sheet by the World Health Organization (WHO) , October 2022.

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, ”Gradient-based learning applied to document recognition,” in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998..

[3] Y. LeCun, K. Kavukcuoglu and C. Farabet, ”Convolutional networks and applications in vision,” Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Paris, France, 2010, pp. 253-256, doi: 10.1109/ISCAS.2010.5537907..

[4] Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey. (2012). ”ImageNet Classification with Deep Convolutional Neural Networks”. Neural Information Processing Systems. 25. 10.1145/3065386..

[5] He, Kaiming et al. “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 770-778..

[6] Ghael, Hirak. (2020). A Review Paper on Raspberry Pi and its Applications. 10.35629/5252-0212225227..

[7] Koutsoukas, A., Monaghan, K.J., Li, X. et al. Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data. J Cheminform 9, 42 (2017)..

[8] Jason Li, Helen Qiu. Comparing Attention-based Neural Architectures for Video Captioning..

[9] Sekhar, Ch, Meghana, P. (2020). A Study on Backpropagation in Artificial Neural Networks. Asia-Pacific Journal of Neural Networks and Its Applications. 4. 21-28. 10.21742/AJNNIA.2020.4.1.03.


```
# Import necessary libraries
import requests # For making HTTP requests
from PIL import Image # For image processing
from transformers import AutoTokenizer,
    AutoModelForImageCaptioning # For image
    captioning
from gtts import gTTS # For text-to-speech
    conversion
import pygame # For audio playback

# Load tokenizer and model for image
    captioning
tokenizer =
    AutoTokenizer.from_pretrained("Salesforce/blip-image-captioning-base")
model =
    AutoModelForImageCaptioning.from_pretrained("Salesforce/blip-image-captioning-base")

# Specify the URL of the image to be captioned
img_url = '#Enter URL here'

# Open and convert the image to RGB format
image = Image.open(requests.get(img_url,
    stream=True).raw).convert('RGB')

# Tokenize and prepare the image for caption
    generation
inputs = tokenizer(image, return_tensors="pt")

# Generate captions for the image
outputs = model.generate(**inputs)

# Decode the generated captions
captions = tokenizer.batch_decode(outputs,
    skip_special_tokens=True)
caption_text = captions[0]

# Print the caption text
print(caption_text)

# Convert the caption to speech
tts = gTTS(text=caption_text, lang='en')
tts.save('caption.mp3')

# Initialize and load the audio file
pygame.mixer.init()
pygame.mixer.music.load('caption.mp3')

# Play the audio
pygame.mixer.music.play()

# Wait for the audio playback to finish
while pygame.mixer.music.get_busy():
    continue
```