Prototyping with Deep Learning - Final Report
Voice synthesis from image data a.k.a image captioning with voice

Author: Rakotondrasoa Rafidison Santatra - 0190252530

Abstract

One of the primary goals of computer vision challenges is to determine which objects are present in an image. One word is generated to describe each object. A more complex problem of that can be to capture and express their relationships in a sentence to describe the whole image, commonly called “image captioning”. For this reason, caption generation of image has long been viewed as a difficult problem. Image captioning with voice is proposed for our project in Prototyping with Deep Learning course. We want to understand and apply the concept of Deep Learning and Neural Networks seen in class such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) like Long Short-Term Memory (LSTMs).

1) Introduction

Automatically describing the content of an image is a fundamental problem in artificial intelligence and one of the most important topics in computer vision. Many applications can be solved using an image-to-speech model to get a natural language speech description of images such as for helping visually impaired people to better perceive the information around them, for cost-saving labeling of many images uploaded to the web in a day, for beneficial for virtual assistants like Alexa and Siri, for indexing of images, etc.

In the beginning, we wanted to build a model in order to return the voice description directly from image inputs. But this task requires a dataset that consists of images and their corresponding description audios that we do not have. One approach is to divide our problem, one model will get as input images and returns the corresponding text description, and another model will take the resulting outputs and returns the corresponding audio. On the first hand, the image-to-text task takes the image and returns the English text description of the image, and on the other hand, the text-to-speech model takes the resulting text description of the image and transforms it into speech.

Such problem entail the following research hypothesis: *By combining a trained image-to-text model and a pre-trained text-to-speech model, can we get the audio description of images?*

To tackle the problem cited before, our main goal is to implement a model that can understand the visual semantics of images, more precisely, captioning the image to text and converting it into voice. Firstly, we want to train an already existing image-to-text model on a chosen dataset, and secondly, we want to transform the resulting text descriptions with a text-to-speech model.

2) Related work

There has been much prior work on image captioning with neural networks so a large number of architecture are available in the market for image captioning generator. For an

attention-based model, one of the most widely-used architectures using attention mechanism was presented by Kelvin Xu et al. [0]. They introduced two attention-based methods, a “soft” deterministic attention mechanism a “hard” stochastic attention mechanism. They showed by visualization “where” and “what” the attention focused on and measured the usefulness of attention mechanism.

“Attention” allows people to focus on information, and it can be in some form replicated. When we look at an image, our attention can be more focused on some part than the others. Muhammad Abdelhadie Al-Malla et al. used a method called “Areas of Attention”, an attention-based model for image captioning that allows a direct association between caption words and image regions [5].

Even if there exists wide range of attention models and captioning architectures, many other architectures can be found in the literature:

Many researchers like Jeff Donahue et al. used a Long-term Recurrent Convolutional Networks (LRCNs) model combining a CNN for feature extraction with LSTM that predicts sequential data [3], in which many applications can be performed like activity recognition, image captioning generator, and video description. Vinayak D. Shinde at al. used the same concepts of a LSTM and CNN model to build an image captioning generator [6]. The CNN extracts the image vectors and LSTM extracts the word vectors.

Oriol Vinyals et al. used a generative model based on a deep recurrent architecture that combines recent advances in computer vision and machine translation and that can be used to generate natural sentences describing an image [1].

Jasmita Khatal et al. implemented a real-time image-to-text-to-speech application that learn to focus on important frames [2]. The system takes the frame from a video and generates captions and text based speech at output. Their goal was to provide real-time outputs to the users, to get a very good accuracy compared to others methods.

Andrej Karpathy and Li Fei-Fei proposed a Multimodal Recurrent Neural Network model that consists of detecting the objects and does the module localization and gets the image description’s accordingly [4]. The model generates natural language descriptions of images and their regions and aligns text language data and visual data through a multimodal embedding based on a combination of Convolutional Neural Networks over image regions and bidirectional Recurrent Neural Networks over sentences.

3) Materials and methods

3.1) Data:

Our data are used for image captioning. There are multiple dataset that are available but for our preliminary use, we decided to use the Common Objects in Context (COCO) [7] dataset from Microsoft, which is among the most widely-used for our task. It consists of images and a sentences in English describing each image. Each image has a single caption. The data is divided into two folders:

annotations folder : contains the captions of every images with a total size of 844MB. It can be downloaded from this link [8].

train2014 folder : contains the image files with a total size of 13.5GB, and can be downloaded here [9].

The dataset is splitted into training (80%), validation (10%) and test sets (10%), respectively 327437 train samples, 40930 validation samples and 40930 test samples.

3.2) Method and architecture: Describe the model you will train or use: inputs and outputs, model architecture (feel free to add a figure), optimizer and learning rate, batch size, number of training epochs, regularization strategies (e.g. dropout, early stopping), evaluation metrics and any other relevant information that would help a researcher to replicate what you did. Check your related papers to see how they describe the models they use, for inspiration.

Our model uses a CNN-RNN Architecture with Attention Mechanism. It has three main parts: a convolutional neural network (CNN) that extracts features from images, an attention mechanism that weights the image features, and an RNN that generates captions to describe the weighted image features. Our model is and encoder/decoder model which is described in the following section.

3.2.1) Encoder - using a CNN

Here, we convert the image data into a required format. We have chosen to use a pre-existing CNN model, called “InceptionV3”, the third edition of Google's Inception Convolutional Neural Network. We use the pretrained version of the network trained on the images from the ImageNet database. It model takes images as input with dimension 299x299x3 and represents them in a feature vectors with dimension 8x8x2048.

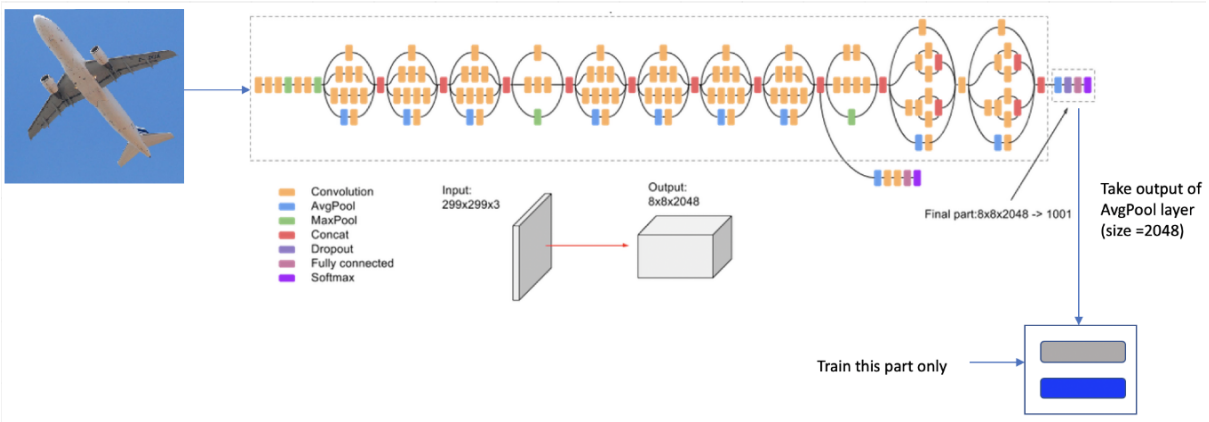


Figure1: InceptionV3 model architecture [11]

After generating the feature map for images using InceptionV3, we determine the dimension of the encoding size of images which is 512 here.

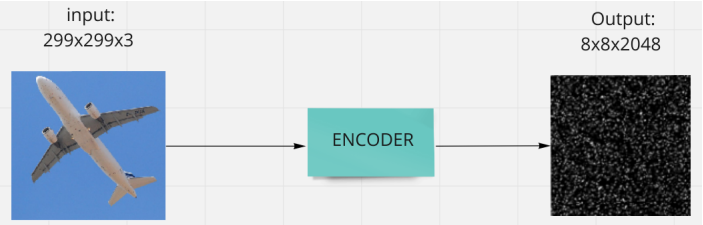


Figure2: Encoder using inceptionV3 model

The following table shows the architecture of our Encoder model:

Layer (type)	Output Shape	Param #
preprocessing_layer (Lambda)	(None, 299, 299, 3)	0
inception_v3 (Functional)	(None, 8, 8, 2048)	21802784
encoding_layer (Dense)	(None, 8, 8, 512)	1049088
reshape_layer (Reshape)	(None, 64, 512)	0

Total params: 22,851,872
Trainable params: 1,049,088
Non-trainable params: 21,802,784

3.2.2) Decoder - using LSTM with attention mechanism

The Decoder's job is to look at the encoded image and generate a caption word by word. Since it's generating a sequence, it would need to be a Recurrent Neural Network (RNN), in our case we use an LSTM. Each predicted word is used to generate the next word as shown in Figure 4.

Attention mechanism: When we describe an image, we don't analyse it as a whole but we inspect the relevant parts of the image. In the same way, we want a machine learning model to use that attention mechanism to help the model to focus on the most relevant portion of the image. Rather than compress an entire image into a static representation, attention mechanism allows salient features to dynamically come to the forefront as needed. This is especially important when there is a lot of clutter in an image. Here, we are using Local and Soft Attention known as Bahdanau Attention introduced by Bahdanau et al. (2014) [10]. Soft attention is when we calculate the context vector as a weighted sum of the encoder hidden states, whereas the hard attention is when, instead of weighted average of all hidden states, we use attention scores to select a single hidden state. The attention mechanism will generates each word of the caption.

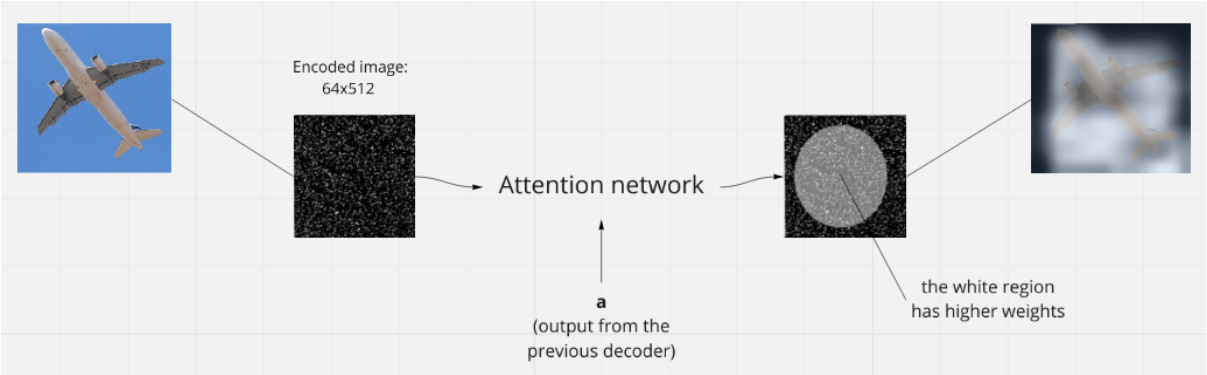


Figure3: Attention network

Local attention chooses to focus only on a small subset of dimension 8x8 of the hidden states of the Encoder represented by h_j of dimension 8x8x512, and per target word **a** represented by s_{t-1} a previous state of the Decoder. The function f_{ATT} is defined to represent the attention network. The function is a simple Feed Forward Neural Network which is a linear transformation of input $U_{attn} * h_j + W_{attn} * s_t$:

$$f_{ATT}(s_{t-1}, h_j) = V^T_{attn} * tanh(U_{attn} * h_j + W_{attn} * s_t)$$

where

$$V^T_{attn} \in R^d,\ U_{attn} \in R^{d \times d},\ W_{attn} \in R^{d \times d},\ s_{t-1} \in R^d \text{ and } h_j \in R^d.$$

The encoded image is weighted by the probability distribution α_{jt} to indicate where the decoder should pay attention. In other terms, we select the subset of the feature vector to apply more attention (more weight). We extract features from a lower convolutional layer instead of the fully connected layer. This allows the decoder to selectively focus on certain parts of an image by selecting a subset of all the feature vectors. It is defined as follows:

$$\alpha_{jt} = Softmax(f_{ATT}(s_{t-1}, h_j))$$

We can compute C_t, the context vector which is the weighted sum of the Decoder input h_j:

$$C_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$

Such that

$$\sum_{j=1}^{T} \alpha_{jt} = 1 \text{ and } \alpha_{ij} \geq 0$$

We can now compute the current state s_t of the Decoder:

$$s_t = LSTM(s_{t-1}, [e(\hat{y}_{t-1}), C_t])$$

Where

$e(\hat{y}_{t-1})$ is the predicted word.

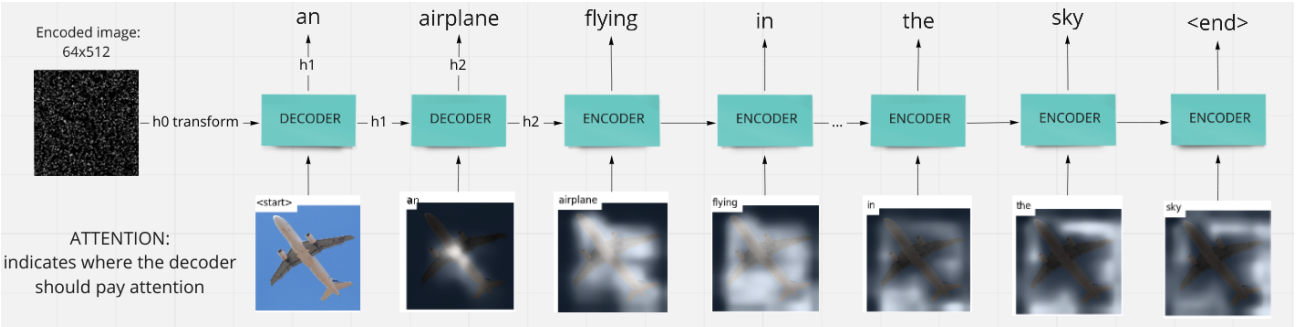


Figure4: Decoder using LSTM with attention mechanism

The following table shows the architecture of the Decoder model:

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 512)]	0	
input_2 (InputLayer)	[(None, 64, 512)]	0	
repeat_vector (RepeatVector)	(None, 64, 512)	0	input_3[0][0]
W1 (Dense)	(None, 64, 512)	262656	input_2[0][0]
W2 (Dense)	(None, 64, 512)	262656	repeat_vector[0][0]
tf.__operators__.add(TFOpLambd	(None, 64, 512)	0	W1[0][0], W2[0][0]
tf.math.tanh (TFOpLambda)	(None, 64, 512)	0	tf.__operators__.add[0][0]
V (Dense)	(None, 64, 1)	513	tf.math.tanh[0][0]
tf.nn.softmax (TFOpLambda)	(None, 64, 1)	0	V[0][0]
input_5 (InputLayer)	[(None, 1)]	0	
dot (Dot)	(None, 1, 512)	0	tf.nn.softmax[0][0], input_2[0][0]
embedding (Embedding)	(None, 1, 256)	2560256	input_5[0][0]
concatenate_3 (Concatenate)	(None, 1, 768)	0	dot[0][0] , embedding[0][0]
input_4 (InputLayer)	[(None, 512)]	0	

lstm (LSTM)	[(None, 512), (None,	2623488	concatenate_3[0][0] input_3[0][0] input_4[0][0]
dense (Dense)	(None, 10001)	5130513	lstm[0][0]

Total params: 10,840,082
Trainable params: 10,840,082
Non-trainable params: 0

3.2.3) Training

The code is inspired by Tensorflow code [12]. The train loop is as follows:

- The image feature vectors are passed through the encoder.
- The encoder output, the hidden state which is initialized to 0 and the decoder input which is the start token <start> is passed to the decoder and it returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss (we used a customed loss function based on the categorical crossentropy).
- Use teacher forcing to decide the next input to the decoder. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
- The final step is to calculate the gradients and apply it to the optimizer and backpropagate (we used the optimizer that implements the Adam algorithm)

Our batch size is 256 and the number of epochs is 15.

3.2.4) Text-to-Speech

We use gTTS (Google Text-to-Speech) module to transform the generated image caption into voice or speech.

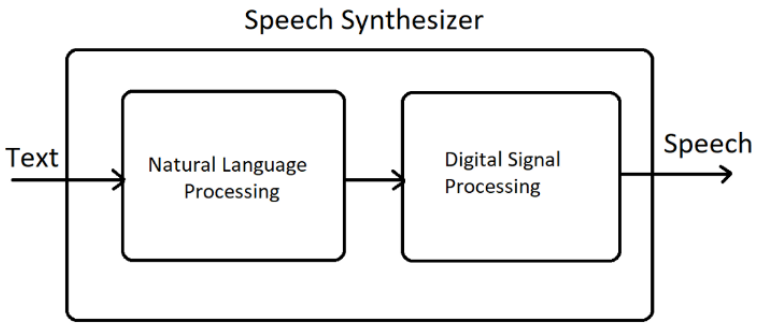


Figure5: Text-to-Speech [13]

3.2.5) Ressources

We used the University of Luxembourg High Performance Computing (ULHPC). Student enrolled in UL are allowed to access it. Here, we used Iris cluster, its features and nodes are presented in the following table:

Node	Model	#nodes	#GPUs	CUDA Cores	Tensor Cores	RPeak DP	RPeak Deep Learning (FP16)
iris-[169-186]	NVIDIA Tesla V100 SXM2 16G	18	4/node	5120/GPU	640/GPU	**561.6TF**	**9000 TFlops**
iris-[191-196]	NVIDIA Tesla V100 SXM2 32G	6	4/node	5120/GPU	640/GPU	**187.2TF**	**3000 TFlops**

Figure6: Iris cluster features [14]

We used 2 GPUs from NVIDIA Tesla V100 SXM2 32G throughout this project to train and run our code.

4) Results and Discussion

Results

During the training of our model, we computed the loss for each epoch:

Epoch 0 Loss 6.2603124
Epoch 1 Loss 1.9180943 Model improved. Saving..
Epoch 2 Loss 1.7568517 Model improved. Saving..
Epoch 3 Loss 1.7009709 Model improved. Saving..
Epoch 4 Loss 1.6762156 Model improved. Saving..
Epoch 5 Loss 1.6666616 Model improved. Saving..
Epoch 6 Loss 1.6679842 Model didn't improve.
Epoch 7 Loss 1.6755275 Model didn't imporve.
Epoch 8 Loss 1.6872364 Model didn't imporve.
Epoch 9 Loss 1.7031935 Model didn't imporve.
Epoch 10 Loss 1.7199607 Model didn't imporve.
Epoch 11 Loss 1.7397624 Model didn't imporve.
Epoch 12 Loss 1.7566384 Model didn't imporve.
Epoch 13 Loss 1.7765595 Model didn't imporve.
Epoch 14 Loss 1.7947617 Model didn't imporve.
Epoch 15 Loss 1.8145951 Model didn't imporve.

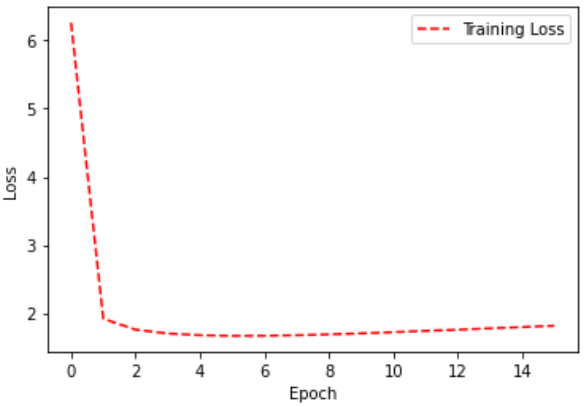


Figure7: Training loss per epoch

The running time to train our model on 15 epochs is about **8 hours**.

We can see in the following the result of the prediction on some images from the test set:

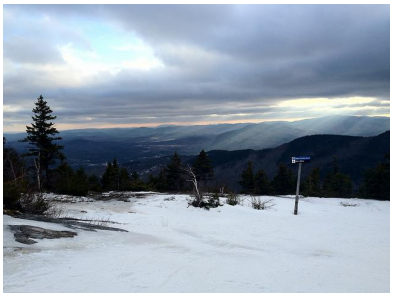
<start> man and woman standing next to each other <end>

<start> pizza with cheese and cheese on it <end>

<start> man train surfboard on sidewalk <end>

<start> dog is sitting on grass with its direction out <end>

<start> two people on bowl on broccoli middle <end>



<start> person is four down broccoli slope <end>

Discussion

The result is not very good, not very bad. We observed from the values of the training loss that at epoch 5, we have the least value of the loss where our model has its best performance. One way to overcome this is maybe to optimize the parameters of our model like:

- Vocabulary size: number of unique words in our model
- Encoding size: dimension of the encoding vector (output of the Encoder model)
- Batch size
- Optimizer
- Loss function

However, performing a Grid Search or Random Search on our model will be very time consuming.

Conclusion

In conclusion, we can considere that our project is a success, the results of the project proved that our research hypothesis can be accepted.

References

[0] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, Yoshua Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”.

[1] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, "Show and Tell: A Neural Image Caption Generator".

[2] Jasmita Khatal, Prajkta Jadhav, Shraddha Parab, Prof. Rasika Shintre, “Real Time Image Captioning and Voice Synthesis using Neural Network”.

[3] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell, “Long-term Recurrent Convolutional Networks for Visual Recognition and Description”.

[4] Andrej Karpathy, Li Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions”.

[5] Muhammad Abdelhadie Al-Malla, Assef Jafar and Nada Ghneim, “Image captioning model using attention and object features to mimic human image understanding”.

[6] Dr. Vinayak D. Shinde, Mahiman P. Dave, Anuj M. Singh, Amit C. Dubey, “Image Caption Generator using Big Data and Machine Learning”.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, “Microsoft COCO: Common Objects in Context”.

[8] url: http://images.cocodataset.org/annotations/annotations_trainval2014.zip

[9] url: http://images.cocodataset.org/zips/train2014.zip

[10] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. “Neural machine translation by jointly learning to align and translate”.

[11]url: https://alquarizm.wordpress.com/2019/03/11/transfer-learning-using-inception-v3-for-developing-image-classifier-part-2/

[12] url: https://www.tensorflow.org/tutorials/text/image_captioning#model

[13] Meelis Mihkla, “Modelling the temporal structure of speech for the Estonian text-to-speech synthesis”.

[14] url: https://hpc.uni.lu/old/systems/iris/