32933: Research Project

Short Technical Report

Project Title: Video Caption Generation

Team Members:

Student Name	Student Id
Vraj Mehta	13488642
Nivetha Anand	13663024
Prem Rijal	12957167

Supervisor: Dr Nabin Sharma

Table of Contents

1. Introduction:

Describing an image or scene in the video is a highly challenging task for humans. Generating captions for both videos and images are the demanding part of imaging science. A connection between understanding human language and the analysis of visual information extracted from image or video frames is used to solve this problem (Sehgal, 2020). Now, on specifying it in technical terms, a combination of ***Natural Language Processing (NLP) with Convolution Neural Network (CNN)*** technique can be used to solve this unique problem (Venugopalan, 2014). The ***project's specific goal is to create a prototype that automatically generates a caption that provides a brief description of the contents of video frames.*** Firstly, we aim to pass a video of four seconds to detect the images' visual and spatial information (video frames) and generate set off image vectors from the Convolution Neural Network (CNN) model. Secondly, text data was prepared by cleaning the description and generating the set of vocabulary. Finally, image vectors obtained from CNN are passed through the ***Recurrent Neural Network (RNN)*** and ***Long Short-Term Memory (LSTM)*** to generate a sequence of meaningful captions. ***The Experimental Research Methodology*** will be carried out in this research project. By doing this research, we tend to produce an innovative prototype that helps generate captions, which can be later be used in many real-life applications.

2. Literature Review:

The Neural network is a component of deep learning, a subset of artificial intelligence which is used to recognise certain objects or processes. In general, the Neural network maps one kind of variable to another kind of variable. Depending on the type of the problem, the mapping also varies. For example, in a classification problem, vector to vector mapping is performed, whereas regression maps vectors to scalar (Vinyals, 2015). Depending on the type of neural network processes, the neural network is classified into different categories.

Convolutional Neural Network (CNN) is an artificial neural network that specialises in detecting some patterns (like edges in the images, specific shapes within images, etc.) and producing the output by making sense of those patterns. The CNN has different layers. Input, Convolution, Pooling (shape recognition in distortion) following by fully connected layers are the basic components of the CNN. The filters are used to detect edges, shapes, etc., of an image, and they transform the variable of the image vectors in each layer. Hence, CNN is used in the image classification process. A Recurrent Neural Network (RNN) is a neural network that specialises in ***sequential data*** for predictions using the memory ***persistence mechanism***. The RNNs are typically trained using the mechanism called teacher force, which means the correct labels are used to train the recurrent networks for the next stage (CodeEmporium 2019). In the sequence of text, it could use word embedding methodology, which incorporates the meaning of the word and closeness of other words in terms of the meaning and the semantics.

As we already know, neural networks map vector to vector in classification problem and vector to a scalar in a regression problem. Similarly, the recurrent neural network uses mathematical functions, which as well maps sequence to vector and sequence to sequence. Currently, there exists a gap in mapping vector to sequence information where video captioning falls under this problem (CodeEmporium 2019). Hence, Both the CNN and RNN architectures can be utilised for this process. The input image/video frames are passed through the CNN process to transform to the meaningful vector output. The convolution feature extraction could be used by RNN to map the image with the sequences of the words that generate the meaningful sentence.

3. A brief overview on Prototype:

3.1 Dataset

A new data set was created for caption generation based on the KTH action data set, which consists of six human actions such as ***walking, jogging, running, handclapping, handwaving and boxing***. Each action consists of a video that lasts for 4 seconds on average, which is converted into a series of image frames. Where every action consists of 100 folders, with each folder consisting of a set of image frames. As a result, each action consists of 100 captions. Since there are six different human actions, we have 600 captions. Dataset is generated in JSON format with four key-value pair with the key consisting of ***"action", "folder name", "caption", and "total images".***

Task Type: **Caption Generation**

Database Link: https://www.csc.kth.se/cvap/actions/

Training dataset: **420 videos**

Validation dataset: **90 videos**

Test dataset: **90 videos**

The dataset is divided into 70% for training, 15% for testing and 15% for training.

Sample video(images):

(hand clapping)

(boxing)

(jogging)

(walking)

(running)

(hand waving)

3.2 Designing the System Architecture:

To solve the current gap existing in Video Captioning, we will explore state of the art existing techniques in Image processing and Natural language processing (NLP). But the uniqueness lies in designing a hybrid model by combining convolutional and recurrent neural network that connects visual data (images) and natural language utterances (captions) with a single network.

3.2.1 Broad Overview of the architecture:

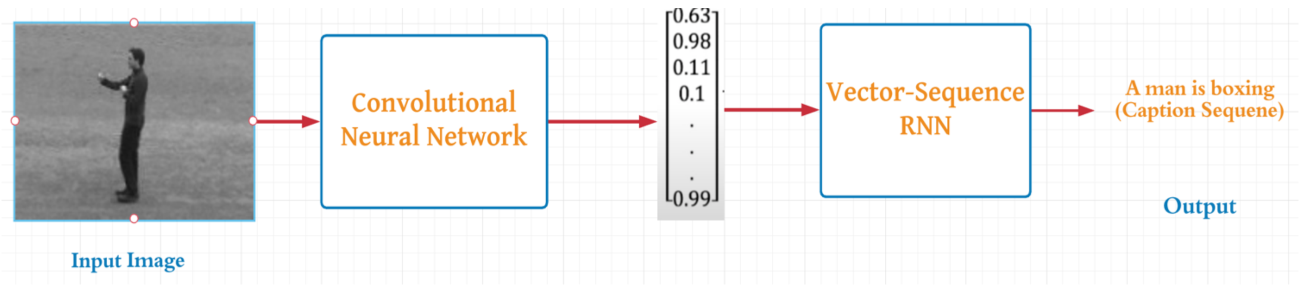

Input Image
Convolutional Neural Network
0.63
0.98
0.11
0.1
0.99
Vector-Sequence RNN
A man is boxing (Caption Sequene)
Output

Figure 1: Overview of the Architecture

Basically, Neural Networks transforms one kind of variable into another kind of variable. As stated in the above Figure 1, our architecture consists of the input image is passed through the ***Convolutional Neural Networks (CNN)*** to provide us with a ***vector representation*** of the image. Where the vectors generated through the CNN are fed as an input to the Recurrent Neural Networks. ***A recurrent Neural Network (RNN)*** is an algorithm of ***Natural Language processing*** that process the incoming vectors and produces an output as a ***Caption Sequence*** for the input images. Caption Sequence denotes the logical sentence or the set of variables that has some defined order to it based on the input images.

3.2.2 Implementation of Prototype:

Implementation of the prototype is carried out by breaking the task into three different sections.

a. ***Preparation of Image Data***
b. ***Preparation of Text Data***
c. ***Development of Deep Learning Model***

a. Preparation of Image Data:

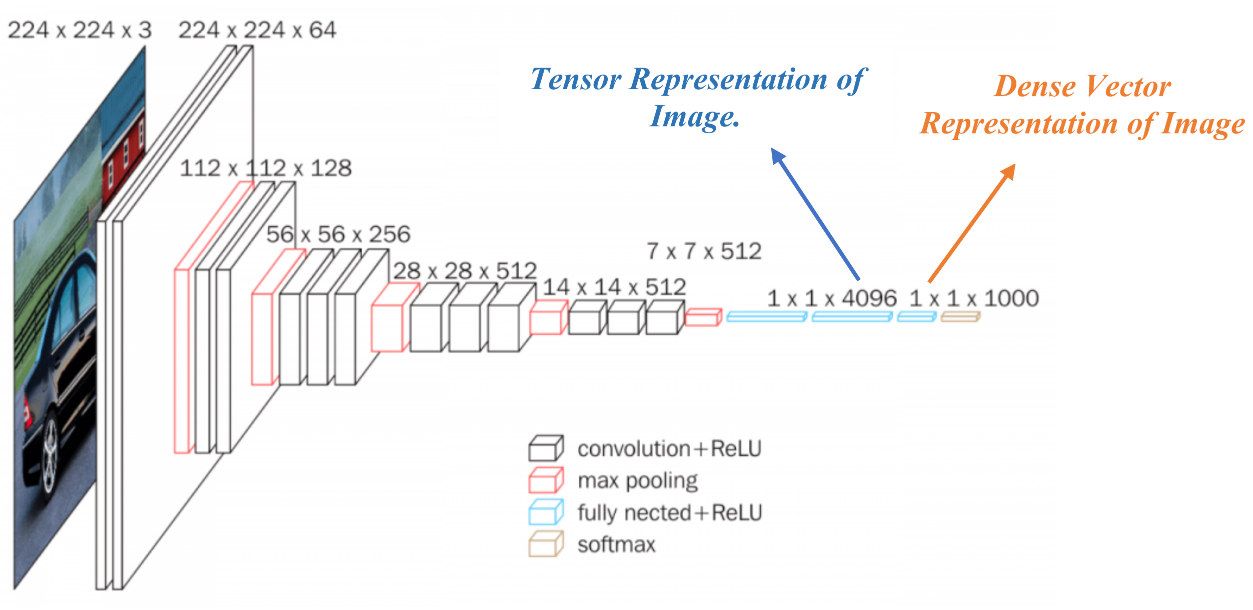


Figure 2: VGG-16 Model

As mentioned above, to provide input to the recurrent neural network, we need to generate the vectors of the input image matrix. Vectors can be generated in two ways, such as directly flattening the image to vector form but, doing this will result in sparse values. So, the best way to generate the image vector is by providing the input image through the Convolutional Neural Networks. ***Convolution Neural Network (CNN)*** is a kind of Artificial Neural Network which is most widely used for image analysis. Our prototype has been designed using a ***pre-trained VGG-16 model*** as a CNN to predict the contents of the photo. Figure 2 represents the brief overview of the VGG-16 model, where we remove the last layer from the model as it helps in the classification of an image. Because "***we are interested in extracting features of the input images rather***

than classification" (Vinyals et al., 2015), we take the final output of the model to be in the form of a ***"one-dimensional dense vector"*** with 4096 elements.

b. Preparation of Text Data:

The second stage in designing a prototype involves preparing the text dataset. We have prepared our text dataset by the following steps.

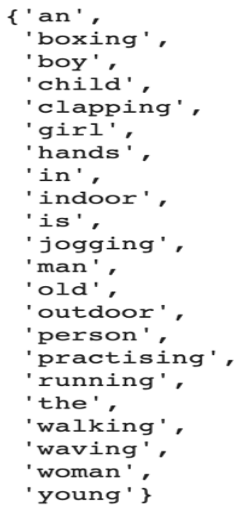
1. Clean the description:

The dataset consists of different descriptions for each image, and the text descriptions involve some minimum cleaning. Text data has been cleaned in the following ways to provide reduced words of vocabulary.

1. By converting all words to lowercase.
2. By removing all the punctuations.
3. By removing all words which have a single character or less in length.
4. By removing all the words which have numbers associated with them.

2. Generate Vocabulary:

The next stage in preparing the text data involves generating vocabulary. Once the text data has been cleaned, it is quite easy for us to summarise the vocabulary size. The vocabulary size for our dataset is 22. The following figure denotes the entire word set in our vocabulary.



```
{'an',
 'boxing',
 'boy',
 'child',
 'clapping',
 'girl',
 'hands',
 'in',
 'indoor',
 'is',
 'jogging',
 'man',
 'old',
 'outdoor',
 'person',
 'practising',
 'running',
 'the',
 'walking',
 'waving',
 'woman',
 'young'}
```

Figure 3: Vocabulary of Text Data

c. Development of Deep Learning Model:

Now that the output from the CNN is passed to the RNN. Recurrent Neural Network uses the "***teacher forcing***" algorithm to provide logical semantics or meaning of the output caption. Correct labels are used to train the next state of recurrent neural networks (Vinyals, 2015). The output of the recurrent neural network is words of the caption generated. Where each word is represented in the form of one hardcoded vector. In the teacher forcing algorithm, each vector of the previous word in the caption is fed as an input to the next iteration. Instead of using one hard encoded vector as an input to the next iteration, it is preferred to use the ***Word Embedding*** method. Word Embedding is a set of vectors that incorporates the meaning of the word and its closeness to the other words in terms of meaning and semantics. These word embeddings are passed to the ***Long-Shot Term Memory (LSTM)*** for memory persistence, whose aim is to store the output of its layer and feed it as an input to the next layer. Figure 5 denotes the structure of LSTM. The final stage is developing a deep learning model to create the caption generation prototype.

1) Tokenisation:

The text descriptions must be encoded with numbers before it has been fed as an input to the model. To encode, the first step involves a consistent mapping of words with unique integers, which is known as Tokenisation. The tokenisation process is done by creating a tokeniser that maps our vocabulary of words with a unique number.

2) Create sequences:

The model we have developed tends to generate a caption for the given photo, where the captions are generated one word at a time. We will now encode the text, where each text description is broken down into separate words. Initially, the model is provided with one photo and a word to generate the next word as an output. After which, the first two words of the text description is provided in a sequence format as an input to the model along with the photo to generate the next output. Thus, the model generates the descriptions by concatenating the generated words and provides the concatenated word as an input to generate the caption for an image.

An example of input sequence creation can be found below, where the input sequence is split into input-output pairs to train the model:

Input (X1)	**Text Sequence (X2)**	**Output-Word (y)**
photo	startseq	A
photo	startseq, A,	Man
photo	startseq, A, Man,	is
photo	startseq, A, Man, is	Boxing
Photo	startseq, A, Man, is, Boxing,	endseq

The below figure represents an overview of the RNN model that generates a sequence of text from the descriptions.

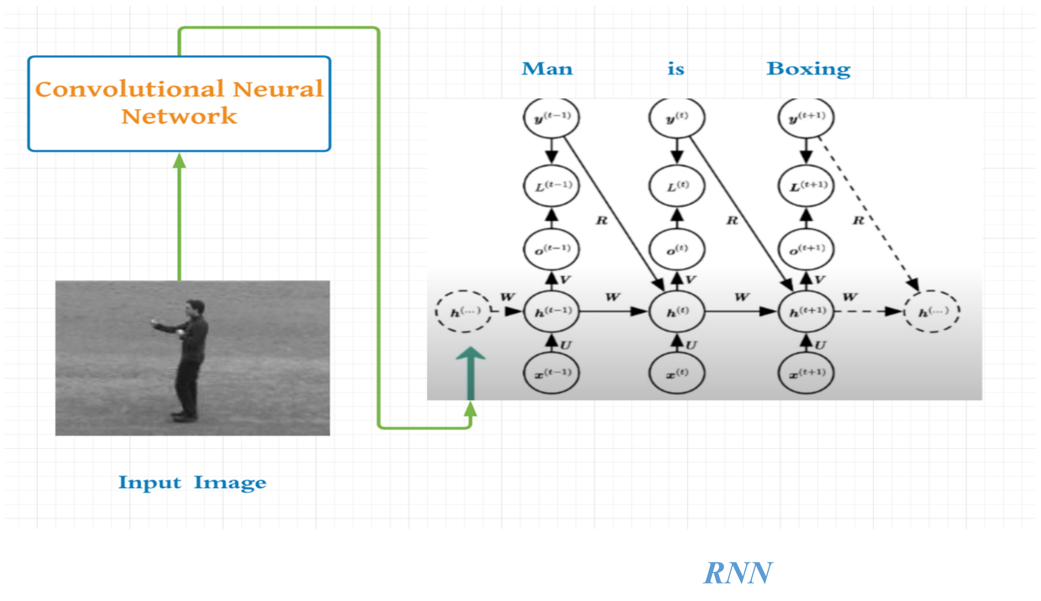


Figure 4: Recurrent Neural Network (RNN) workflow

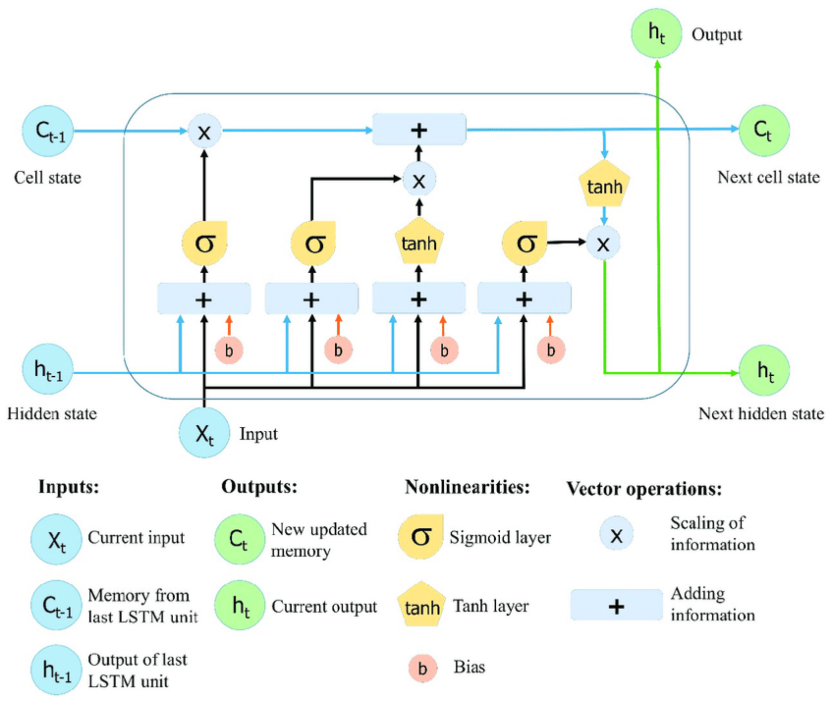


Figure 5: Structure of Long Short-Term Memory

c. Defining and fitting the model:

The prototype of our model has been defined by three different sections:

a. **Image Feature Extractor**:

We have designed our prototype with a pre-trained VGG-16 model. Pre-processing of images was done using the VGG-16 model, where the features extracted from the images will be used as an input to the model.

b. Sequence Processor:

This layer focuses on the word embedding layer where we handle text data input using ***Long Short-Term Memory*** (LSTM) which is incorporated in the recurrent neural network layer.

c. Decoder:

Both the output from the Image feature extractor and Sequence processor tends to provide a fixed-length vector which is then combined and processed by a dense layer to make final predictions.

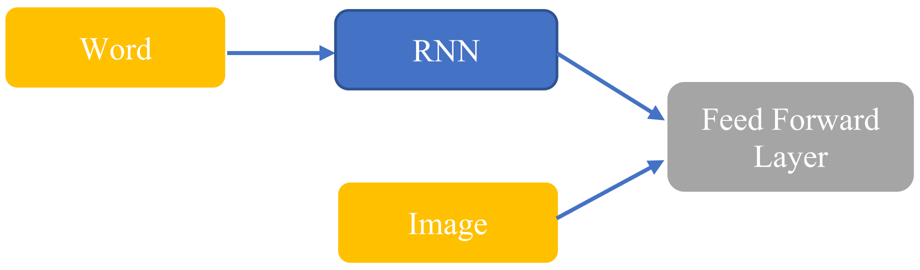
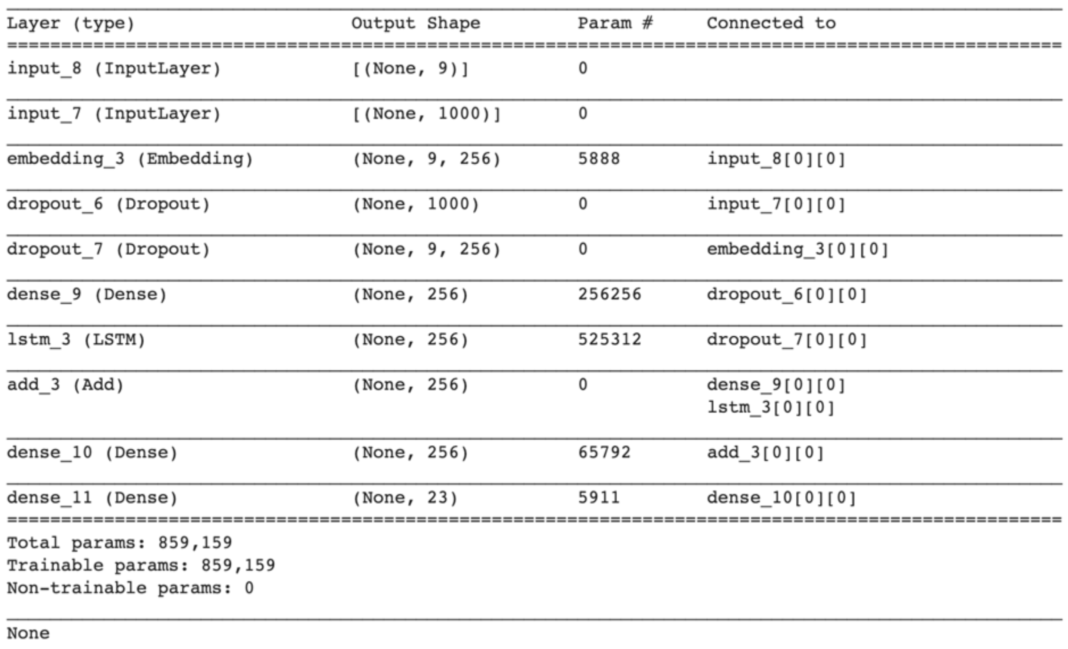


Figure 6: Schema of the model

The below figure denotes the summary of them, model specifically the shapes of the layer:

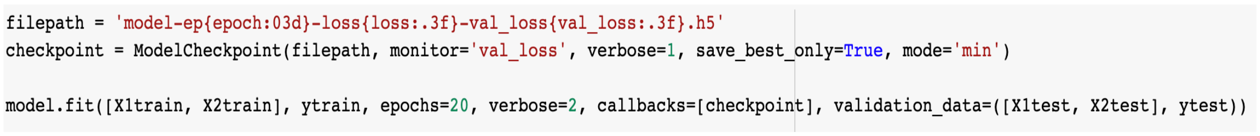


```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_8 (InputLayer)            [(None, 9)]          0
________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 1000)]       0
________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 9, 256)       5888        input_8[0][0]
________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 1000)         0           input_7[0][0]
________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 9, 256)       0           embedding_3[0][0]
________________________________________________________________________________________________
dense_9 (Dense)                 (None, 256)          256256      dropout_6[0][0]
________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 256)          525312      dropout_7[0][0]
________________________________________________________________________________________________
add_3 (Add)                     (None, 256)          0           dense_9[0][0]
                                                                 lstm_3[0][0]
________________________________________________________________________________________________
dense_10 (Dense)                (None, 256)          65792       add_3[0][0]
________________________________________________________________________________________________
dense_11 (Dense)                (None, 23)           5911        dense_10[0][0]
================================================================================================
Total params: 859,159
Trainable params: 859,159
Non-trainable params: 0
________________________________________________________________________________________________
None
```

Figure 7: summary of the model

d. Fitting the model:

Now that we have defined the model, the next stage involves fitting the model with the training dataset. The below figure denotes model fitting, where we have fit the model with 20 epochs.



```
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
```

Figure 7: fitting the model

3.3 Evaluation and Results

We are using the BLEU evaluation mechanism. BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of the text to one or more reference translations. The trained model is evaluated against a given dataset of image descriptions and image features. The predicted and actual descriptions are collected and evaluated based on the corpus BLEU score, which summarises the closeness between the predicted text to the actual one.

BLEU -1	BLEU -2	BLEU -3	BLEU -4
0.3425	**0.5852**	**0.7251**	**0.7650**

4. Conclusions and future work:

We have presented a prototype, an overall neural network that can take input as video (image frames) and generate a meaningful caption for the input provided. The prototype is based on encoding an image into vector representation which is done by convolutional neural network and generating a meaningful caption using recurrent neural network. The model is trained on the newly created KTH action dataset, which has six actions, each consists of a video on an average of four seconds. The evaluation was done based on BLEU metrics which is a quantitative evaluation that assesses the quality of the captions generated (Vinyals, 2015). In future, we would further investigate the "***Attention Mechanism***", which involves focusing on specific parts of the image while generating the different words for caption where it can help to produce a more detailed caption for an image (CodeEmporium 2019). Also, video caption generator is an effective tool that can be applied in different fields such as *automatic labelling of videos, automated description of images on websites, helps low vision people who can view larger text fonts, aids in organising photos based on the objects present in the image (google photos)* (Sehgal, 2020).

References:

1. Vinyals, O., Toshev, A., Bengio, S. & Erhan, D. 2015, 'Show and tell: A neural image caption generator', *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3156-64.

2. Amritkar, C. & Jabade, V. 2018, 'Image Caption Generation Using Deep Learning Technique', *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1-4.

3. Sehgal, S., Sharma, J. & Chaudhary, N. 2020, 'Generating Image Captions based on Deep Learning and Natural language Processing', *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 165-9.

4. Amirian, S., Rasheed, K., Taha, T.R. & Arabnia, H.R. 2020, 'Automatic Image and Video Caption Generation with Deep Learning: A Concise Review and Algorithmic Overlap', *IEEE Access*, vol. 8, pp. 218386-400.

5. CodeEmporium 2019. Building an Image Captioner with Neural Networks, YouTube, viewed 29 April 2021, <https://www.youtube.com/watch?v=c_bVBYxX5EU>.

6. Json Brownlee, 2019, How to Develop a Deep Learning Photo Caption Generator from Scratch, weblog, Machine Learning Mastery, viewed 26 April 2021, < https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/>.

7. Aafaq, N., Mian, A., Liu, W., Gilani, S.Z. & Shah, M. 2019, 'Video description: A survey of methods, datasets, and evaluation metrics', *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1-37.

8. Pan, Y., Yao, T., Li, H. & Mei, T. 2017, 'Video Captioning with Transferred Semantic Attributes', *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 984-92.

9. Ahuja, B., Coutinho, A., Bhangale, C., Sankhe, C. & Khedkar, S. 2020, 'Video Analysis and Natural Language Description Generation System', *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 205-8.

10. Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R. & Saenko, K. 2014, 'Translating videos to natural language using deep recurrent neural networks', *arXiv preprint arXiv:1412.4729.*

11. Wu, H., Zhang, J., Gao, Z., Wang, J., Yu, P.S. & Long, M. 2021, 'PredRNN: A Recurrent Neural Network for Spatiotemporal Predictive Learning', *arXiv preprint arXiv:2103.09504.*

Acknowledgement:

Our team would like to especially thank our supervisor Dr Nabin Sharma (University of Technology Sydney), for his continued guidance and support throughout our research project. We sincerely thank him for his supervision and his tremendous support by investing in his time within his busy schedule to provide suggestions based on the progression of our research work on a weekly basis.

We would also like to thank our subject coordinator, Dr Kyong Kang, for giving us the opportunity to do research-based learning and helping us prepare for the project.

Appendix (if applicable):

The following code provides an overview of the training and validation loss of our model.

```
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
```

```
Epoch 1/20
85/85 - 7s - loss: 1.7445 - val_loss: 0.7862

Epoch 00001: val_loss improved from inf to 0.78624, saving model to model-ep001-loss1.745-val_loss0.786.h5
Epoch 2/20
85/85 - 1s - loss: 0.6647 - val_loss: 0.6029

Epoch 00002: val_loss improved from 0.78624 to 0.60286, saving model to model-ep002-loss0.665-val_loss0.603.h5
Epoch 3/20
85/85 - 1s - loss: 0.5953 - val_loss: 0.5940

Epoch 00003: val_loss improved from 0.60286 to 0.59402, saving model to model-ep003-loss0.595-val_loss0.594.h5
Epoch 4/20
85/85 - 1s - loss: 0.5536 - val_loss: 0.5261

Epoch 00004: val_loss improved from 0.59402 to 0.52609, saving model to model-ep004-loss0.554-val_loss0.526.h5
Epoch 5/20
85/85 - 1s - loss: 0.5045 - val_loss: 0.5104

Epoch 00005: val_loss improved from 0.52609 to 0.51038, saving model to model-ep005-loss0.505-val_loss0.510.h5
Epoch 6/20
85/85 - 1s - loss: 0.4800 - val_loss: 0.4915

Epoch 00006: val_loss improved from 0.51038 to 0.49147, saving model to model-ep006-loss0.480-val_loss0.491.h5
Epoch 7/20
85/85 - 1s - loss: 0.4588 - val_loss: 0.4545

Epoch 00007: val_loss improved from 0.49147 to 0.45453, saving model to model-ep007-loss0.459-val_loss0.455.h5
Epoch 8/20
85/85 - 1s - loss: 0.4352 - val_loss: 0.4493

Epoch 00008: val_loss improved from 0.45453 to 0.44935, saving model to model-ep008-loss0.435-val_loss0.449.h5
Epoch 9/20
85/85 - 1s - loss: 0.4218 - val_loss: 0.4747

Epoch 00009: val_loss did not improve from 0.44935
Epoch 10/20
85/85 - 1s - loss: 0.4075 - val_loss: 0.4261
```

```
Epoch 00010: val_loss improved from 0.44935 to 0.42605, saving model to model-ep010-loss0.407-val_loss0.426.h5
Epoch 11/20
85/85 - 1s - loss: 0.3862 - val_loss: 0.4283

Epoch 00011: val_loss did not improve from 0.42605
Epoch 12/20
85/85 - 1s - loss: 0.3731 - val_loss: 0.4285

Epoch 00012: val_loss did not improve from 0.42605
Epoch 13/20
85/85 - 1s - loss: 0.3669 - val_loss: 0.4060

Epoch 00013: val_loss improved from 0.42605 to 0.40604, saving model to model-ep013-loss0.367-val_loss0.406.h5
Epoch 14/20
85/85 - 1s - loss: 0.3682 - val_loss: 0.4159

Epoch 00014: val_loss did not improve from 0.40604
Epoch 15/20
85/85 - 1s - loss: 0.3565 - val_loss: 0.4144

Epoch 00015: val_loss did not improve from 0.40604
Epoch 16/20
85/85 - 1s - loss: 0.3413 - val_loss: 0.4086

Epoch 00016: val_loss did not improve from 0.40604
Epoch 17/20
85/85 - 1s - loss: 0.3377 - val_loss: 0.4004

Epoch 00017: val_loss improved from 0.40604 to 0.40038, saving model to model-ep017-loss0.338-val_loss0.400.h5
Epoch 18/20
85/85 - 1s - loss: 0.3373 - val_loss: 0.4081

Epoch 00018: val_loss did not improve from 0.40038
Epoch 19/20
85/85 - 1s - loss: 0.3314 - val_loss: 0.4093

Epoch 00019: val_loss did not improve from 0.40038
Epoch 20/20
85/85 - 1s - loss: 0.3204 - val_loss: 0.4063

Epoch 00020: val_loss did not improve from 0.40038
<tensorflow.python.keras.callbacks.History at 0x7fa4459dac10>
```