TEK5040 – Project Report – Image captioning

I. My methods

This concept of captioning an image proves to be quite a challenging task. In this project I've come up with multiple network topologies to observe changes in term of performance and text-captioning. I've experimented with **VGG16/ResNet50** as the 2 primary CNN-models for feature-extraction as these CNN-model has shown state-of-the-art results in imagenet.

Firstly, introducing how my image-captioning is done in general. Assume we have 25 images to train. To start, a convolutional neural network(CNN) extract certain features from an image also known as ***feature-extraction*** by taking the outputs from the second last layer. This will give us an image in parameterized vector containing said unique features with set of glimpses. Since every image in training has a caption, we need to perform some text-processing. First we find all unique words in the training set and this forms a ***vocabulary size*** and the **maximum length** of a sequence in captions. Using Keras' **Tokenizer** module we can then transform the vocabulary into integers ranging from 0 to the size of the vocabulary. So for instance, assume you have the following string «There is a dog in this image», representing it to integer would look similar to an array of integers «[1, 28, 32, 4, 2, 36, 128]» where 128 could be the last word in the vocabulary and also the size of it. The remaining integers are represented as different words. Reason behind this encoding is due to the fact that network has a huge difficulty working directly on strings. By using a representation of numbers it allows the network to learn pattern between these integers. This process is done 25 times and will give us two arrays where the first one contains 25 vectorized image-features for where each vector describes one image and the second one which contains 25 encoded-caption(sentences in integers) where each vector annontates an image.

Now by concatenating **image-features** and **encoded-caption** we now have an input-data. Creating the target is relatively simple, we take one encoded-caption sequence and pad the sequence into **maximum length.** Then the target-data is one-hot encoded consisting the next word in sequence where both input and target encoded-sequences are padded if the original encoded-caption is shorter than **maximum length** to be that length. Note that the input/target data is done with respect to the vocabulary and as such, for every new training we can not use the same **vocabulary** as the vocabulary will change every time. As such we must **save the tokenizer** object if we intend to keep the model and weights. Now that we have formatted input-data and target-data we can now feed these into our choice of language model to train.

II. Neural Network Architecture

Long-short-term memory

Recurrent neural networks(RNN) are a variant of neural networks that try to learn temporal data. It does accordingly by using previous knowledge to affect its output. Still, regular RNNs have a tendency to forget when working with longer sequences[4] [5]. Several approaches have been attempted to solve this issue, but argueably the most preferred model is the LSTM model. Long short-term memory (LSTM) is a RNN architecture designed to improve its property of storing and

accessing information than its RNNs counterpart by implementing a memory cell called LSTM-cell [6]. Advantages of such capability is generating sequences while reminiscing previous inputs over longer durations than standard RNNs. Hence, LSTM-networks prove to be more efficient for future prediction than regular RNNs[4]. This is why I use LSTM-network.

Convolutional neural networks

Convolutional neural networks (CNN) is neural network architecture used primarily for image classification and object detection. This can be done in both 2D-space [1] [9] and 3D-space[7][8], but notably convolution in 2D is the most common one. To summarize how CNN is performed. At the first layer it takes an image of any arbitrary size shaped with 4 dimension(batchsize, height, width, channel). The convolutional layer will then transform the input3D to an output3D volum through a differentiable functions. There exist many layers such as max-pooling, min-pooling, batchnormalization and etc. All of these layers can be combined or used single handedly depending on the task to be performed. These layers can be repeated for multiple steps depending on strides of the layers(pooling or convolutional layers) as strides on the layers downscales the image by a factor One can view these convolutional layers as finding unique features of the image. At the last layer is typical fully-connected layer that will contain class-probabilities based on the output3D from the convolutional layers [10] [11].

Image captioning

There are a lot of approaches to solve this task. One of such method is using a deep convolutional neural network to give a vectorized representation of an image(feature-extraction) and then feeding into a LSTM-network to predict caption is known as top-down approach for image captioning[12]. This can viewed as encoder/decoder model where CNN is the encoder and the RNN as the decoder. Relation between the encoder and decoder is also known as attention[13] and that mechanism determines where the network observes to make its captioning. Thus a good attention model plays a major role in producing image-captioning. In my project I use the top-down approach for captioning an image.

III. Deep-learning Library

In this project I've mainly used *'Keras'* which is a tensorflow backend library implementation for machine-learning. As such I've utilized its vast library in my implementation and I will briefly describe the following functions which I've used for this work.

IV. Important functions from Keras' library

Keras' Embedding layer – This layer turns positive integers(indexes) into dense vectors of fixed size(Vocabulary_size, dim_embedding, max_length). Assume you have the same string «There is a dog in this image», that equals to 7 possible index as there are 7 unique words in the vocabulary. The next parameter is the size of embedding vectors (dim_embedding) where this vector finds indexes(words) that are similar in dim_embedding length. And the last determines the size of each input sequence. In short, this layer attempts to find correlations of the input index with other words in another dimension based on the size of dim_embedding.

Keras' Bidirectional – This wrapper retrieves the outputs of both its next and previous output for the next prediction for an RNN, or more specific LSTM which is used in this work. As such, it is capable of providing additional context to the network and can potentially yield better results.

Keras' Time-distributed layer – This layer is a little bit special, because here we are applying Dense-layer for each of the hidden states for each time-step. Doing this you're only interacting the values between its own timestep. So predicting sequences are based on previous sequences in a seperate timestep rather than combined random interaction between different timesteps and channels. One downside of using this layer is training. By applying Time-Distributed as wrapper layer in cohesion with output layer could require more training as these predictions are sentences rather than single words and predicting sequences are in general more difficult.

V. Dataset

For the dataset I used the COCO(Common Object In Context) dataset which consists of over 100 000 images for training, 5000 validation images and 41k test images, however I only used the validation set in both training and testing in this project due to lack of computational resources and time.

VI. Model(s)

The first model is a ***simple CNN-LSTM*** model where the CNN extracts features and it is then concatenated with text-captioning and preprocessed. When the preprocessing is done, the sequence is then fed into basic LSTM-network to predict a text-description solely on its input sequence.

The second model is a ***CNN-Bidirectional-LSTM*** model where the same preprocessing is performed as the first model. The input sequence is then feeded forward into the Bidirectional-LSTM model. Difference here is that the LSTM now has access to both previous and next outputs for its next prediction giving it more context for next prediction.

The third model is a ***CNN-TimeDistributed-Bidirectional-LSTM*** model that predict novel sentences rather than one word-prediction. The bidirectional property allows the model to recieve both the previous and next sentence output for its next prediction and the TimeDistributed wrapper allows the LSTM to recieve the previous sequences as input. In short, more inputs are provided for the next sequence prediction.

VII. Training

The model's topology are created be as similar as possible to test the difference between in terms of performance and the qualitative analysis of the actual image captioning. These are the following hyper-parameters for all the 3 unique models for training.

	CNN-LSTM	CNN-Bidirectional-LSTM	CNN-TimeDistributed-Bidirectional-LSTM
Dropout	0.5	0.5	0.5
Dense units	512	512	512
LSTM units	256	256	256
CNN-model	ResNet50	ResNet50	ResNet50
Epochs	75	75	75
Dim-embedding	512	512	512
Batch-Size	32	64	64
L2-regularizer	0.01	0.01	0.01
Number of images	500	500	500
Percent train	90%	90%	90%

***Table I**. Figure showing the different hyper-parameters for training*

VIII. Results

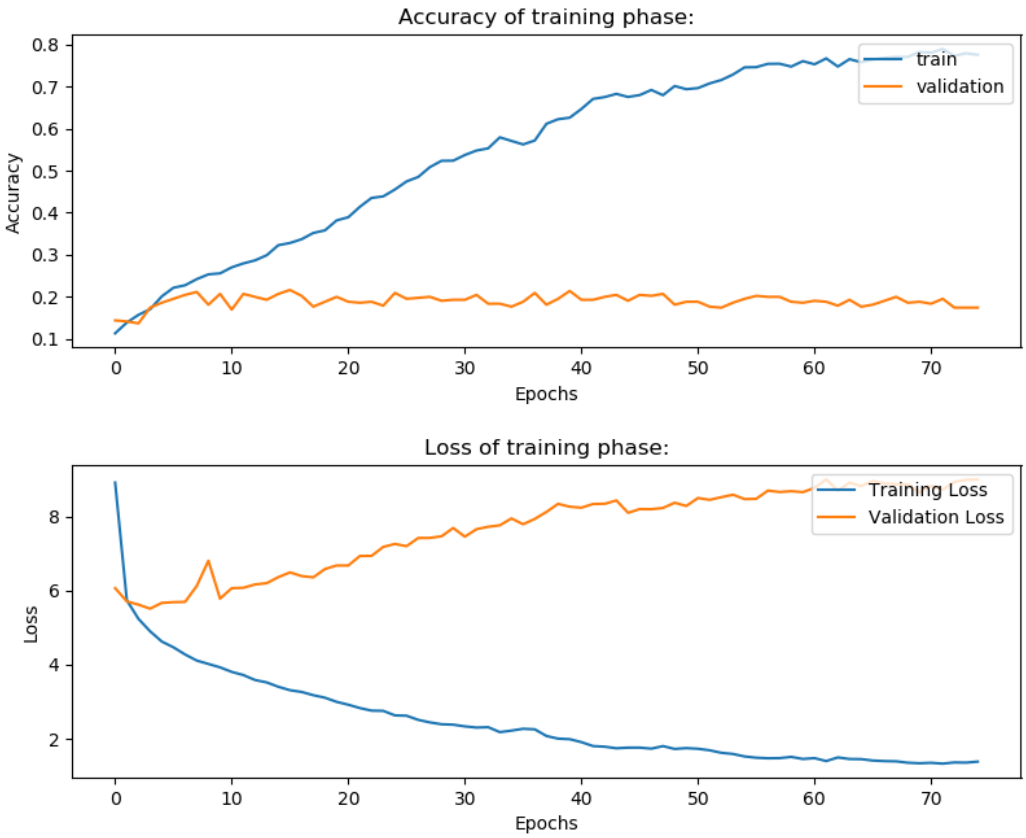

***Figure I**. Training accuracy, Validation accuracy, Training loss and Validation loss for CNN-LSTM model for 75 epochs*

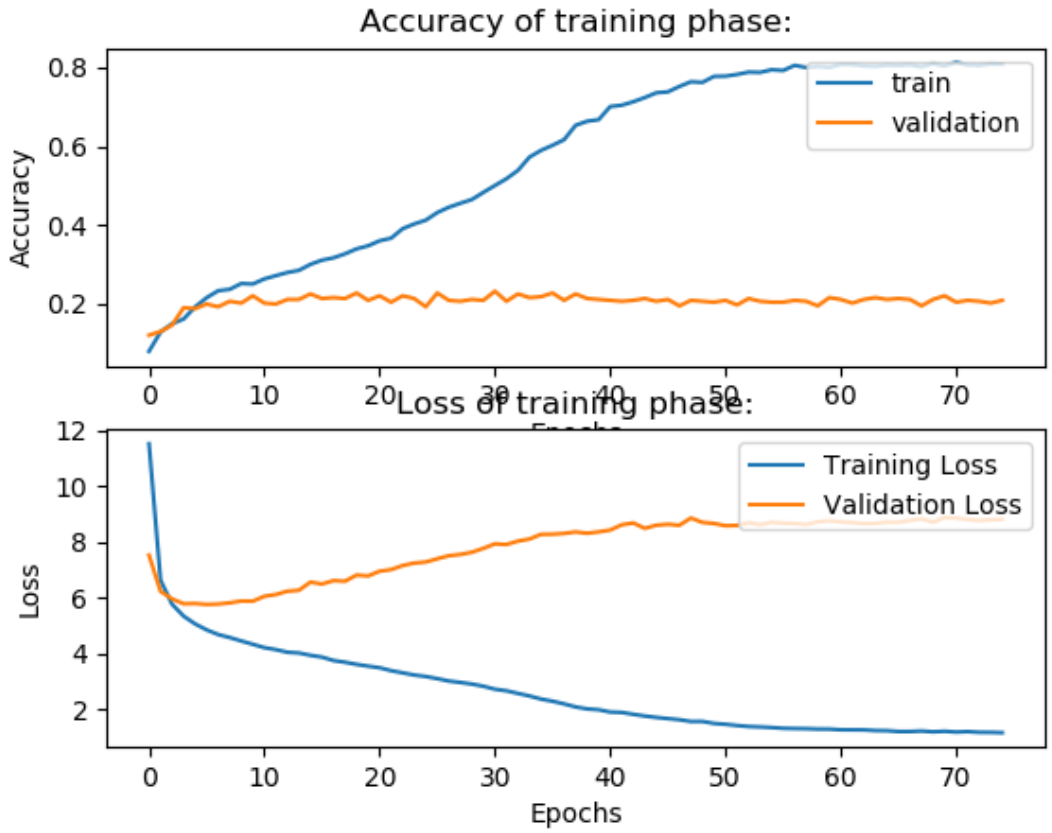

***Figure II**. Training accuracy, Validation accuracy, Training loss and Validation loss for CNN-Bidirectional-LSTM model for 75 epochs*

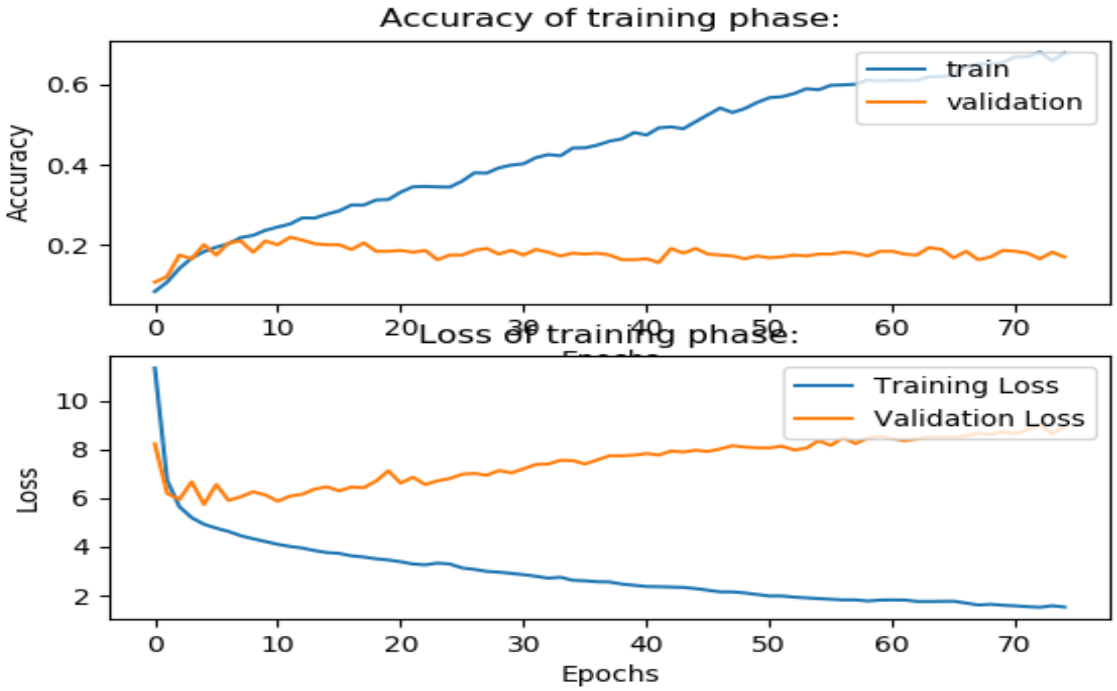

***Figure III**. Training accuracy, Validation accuracy, Training loss and Validation loss for CNN-TimeDistributed-Bidirectional-LSTM model for 75 epochs*

The validation loss and validation accuracy never seem to improve, but seem to get worse. Keras' performs validation accuracy/loss if the target data misses by 1 step, but sentences can start in different way, meaning that it could still make sense. The training losses and accuracy however proves otherwise. However I value more qualitative analysis rather than quantitative. The reason for this is because some of the captions do make sense, and some of them don't. This could suggest our model is overfitting quite fast because the model has only trained on 500 images which is barely enough to generalize. At the end of the report past reference section you can see some results from the 3 different models. Information about which model was used will also be specified in the images. Overall I think my language models can perform better had I trained all three models for longer duration with more images. Ideally 5000 images as said in the **''Dataset''**-section. All of the training was done on my computer with a CPU so training took quite some time. So far based on the qualitative analysis the simple ***CNN-LSTM*** is proven to be the best one but this is hard to tell since all 3 models haven't been trained long enough to form such critic and because of this my image-captioning yields horrendously errornos resutls.

IX. Discussion

Originally I also intended to train my own CNN to be used as feature-extraction but this proved to be time-consuming to train a seperate network and three seperate language models and hence I only used Keras' applications of available CNN models.First I tried to train my own CNN using CIFAR-10 dataset as stated in my project proposal where every image is sized 32x32. The problem arises when we must downscale the COCO-images substantially. Downscaling a COCO-image to 32x32 and applying this parameterized vector into the language model however yielded errornous result. Therefore my CNN-model was substituted with state-of-the-art CNN-model called ResNet50[1], where Keras has this implemented into its libraries. Reason could be ResNet takes an image with input-shape (224,224,3) ie 224x224 image which provide more effective parameterized vector and features for the language model and yielded good results.

Alternative choice for output layer ie instead of using a fully-connected layer(with softmax) at the end, one could potentially replace the last layer with a Mixture density network-layer that takes a sample from gaussian distributions with a probability(softmax) for next prediction[2]. Recent works has proven Mixture-Density Recurrent Neural Network, abbreviated MDRNN has shown promising results [3][4].

X. REFERENCES

[1] Kaimeng He, Xiangyu Zhang et al. *''Deep Residual Learning For Image Recognition*», Microsoft Research, December 10, 2015

[2] C. Bishop, ''*Mixture Density Networks*», Technical report, 1994.

[3] Charles Martin, Jim Tørresen, *''Robojam, Musikal Mixture density for Collaborative Touchscreen Interaction* », University Of Oslo, November 30, 2017.

[4] Alex Graves, *''Generating Sequences using Recurrent neural networks*», University of Toronto, June 5, 2014.

[5] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. ‘ ’*Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-term Dependencies.*” In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. 2001.

[6] S. Hochreiter, J Schmidhuber, ‘ ’*Long Short-term Memory*”, Neural Computation, 1997.

[7] Rui Hou, Chen Chen, Mubarak Shah, ‘ ’*An End-to-end 3D Convolutional Neural Network for Action Detection and Segmentation*”, Journal of Latex Class Files, Vol 14, NO.8, August 8, 2015.

[8] Pawel Mlynarski et al. ‘ ’*3D Convolutional Neural networks for Tumor Segmentation using Long-range 2D Context*”, July 23, 2018.

[9] Karen Simonyan, Andrew Zisserman, ‘ ’*Very Deep Convolutional Networks For Large-Scale Image Recognition*”, ICLR 2015.

[10] Jonathan Long, Evan Shelhamer, Trevor Darrell, ‘ ’*Fully Convolutional Networks for Semantic Segmentation*”, IEEE Transactions on Pattern analysis and Machine Intelligence Volume 39, Issue 4, April 1, 2017.

[11] ‘ ’*Convolutional networks*” [Internet page], University of Stanford, November 29, 2018, Retrieved from http://cs231n.github.io/convolutional-networks/

[12] Moses Soh, ‘ ’*Learning CNN-LSTM Architectures for Image caption Generation*“, Stanford Unviersity, 2016.

[13] Denny Britz, (3 January 2016), ‘ ’*Attention and Memory in Deep Learning and NLP*“ [Blog post], retrieved from http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/