Cluster Computing
https://doi.org/10.1007/s10586-018-1885-9

Fast image captioning using LSTM

Meng Han[1] · Wenyu Chen[2] · Alemu Dagmawi Moges[2]

Received: 14 November 2017 / Revised: 8 January 2018 / Accepted: 17 January 2018

Abstract
Computer vision and natural language processing have been some of the long-standing challenges in artificial intelligence. In this paper, we explore a generative automatic image annotation model, which utilizes recent advances on both fronts. Our approach makes use of a deep-convolutional neural network to detect image regions, which later will be fed to recurrent neural network that is trained to maximize the likely-hood of the target sentence description of the given image. During our experimentation we found that better accuracy and training was achieved when the image representation from our detection model is coupled with the input word embedding, we also found out most of the information from the last layer of detection model vanishes when it is fed as thought vector for our LSTM decoder. This is mainly because the information within the last fully connected layer of the YOLO model represents the class probabilities for the detected objects and their bounding box and this information is not rich enough. We trained our model on coco benchmark for 60 h on 64,000 training and 12,800-validation dataset achieving 23% accuracy. We also realized a significant training speed drop when we changed the number of hidden units in the LSTM layer from 1470 to 4096.

Keywords Computer vision · Natural language processing · Image annotation · LSTM

1 Introduction

Automatic Image annotation also known as automatic image tagging or captioning is the process by which a computer system automatically assigns a caption metadata to a digital image. A typical image annotation system takes in consideration of two vital ingredients, a semantic understanding of digital images and natural language processing unit to decode the sematic information of the image into human readable output.

✉ Wenyu Chen
cwy@uestc.edu.cn

Meng Han
hmuestc@126.com

Alemu Dagmawi Moges
dagisky@gmail.com

[1] School of Political Science and Public Administration, University of Electronic Science & Technology of China, Chengdu, People's Republic of China

[2] School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, People's Republic of China

An automated image annotation system is one of the systems that plays a key role in Image retrieval systems by automatically generating image metadata which later be indexed for searching. One of the major advantage of automatic image annotation system against CBIR systems is that the user, which allow users more freedom, can more naturally specify queries.

So far, different Architectures of Automatic image annotation systems have been proposed with wide range of algorithms in use. Some of the methods used include the use of texture similarity [1], support vector machines [2, 3], statistical models and linguistic indexing models [4, 5], maximum entropy [6], etc. however, a major performance boost was only achieved in the advent of deep learning breakthrough in computer vision and language processing.

Deep neural networks are an implementation of neural network in which a multiple hidden layers of neurons and then run massive amounts of data through the system to train it. Today machines trained using deep neural networks in some scenarios outperform humans with significant margin. This makes using deep artificial neural network ideal in solving many of AI problems like computer vision, natural language processing and automatic image annotation.

Great advances in both computer vision and natural language processing also resulted in further advancement in automatic image annotations system. The state of the art at the time of writing this paper is NIC (neural image caption generator) model from Google [7]. This model uses an encoder–decoder recurrent neural network (RNN) to generate a description for the given image and deep convolutional neural network (CNN) (InseptionV3) for image classification. Another remarkable work from Stanford uses an object detection model with bi-directional recurrent neural network [8]. Our model stands on an in-depth understanding of both models. We also took consideration of the strength and weakness of different automatic image captioning models and their sub models including other potential sub models for more effective result [9–11].

Automatic image annotation as a retrieval problem suffers from different key shortcomings. One of the key disadvantage being that it requires a ridiculously huge amount of data to predict within relatively sound accuracy. This is because, as discussed above, these models try to find a semantically close match image with its respective description to transfer or generate a caption.

In this work, we try to simulate this capability of understanding high-level representation by generating image description from high-level image representation as well as the lower level representation of the same image. We call the objects in the scene and their spatial position a high-level representation, and the row feature map of the image a low-level representation.

The objective of research lies on the fact that humans are capable of describing an image in one or may ways. We can choose to describe a scene on its entirety or use a portion of it. This capability of recognizing an object and its relation with an adjacent object and generate a description out of it is an important value for deep semantic understanding. In order to achieve this target using an object detection model instead of an image classification as an initial module for the automatic annotation makes more sense, as different detection and sub regions can be subjected to the language model to generate localized description out of the whole. Thus, we aim to use a faster detection model that can detect objects real-time and an encoder–decoder language model to achieve a fast captioning system. This in turn will serve as a groundwork for sub-annotation and other elegant works.

We focus our efforts in exploring different variation of convolutional neural network for Image detection challenge and combine them with the state of the art machine translation model to achieve effective, fast automatic image annotation.

We take in-depth look at different image detection model's exploring their strength and weakness, further more we explore previous deep neural network based automatic image annotation systems.

Finally, we examine architectural variation based on our model to achieve a better performance and speed. Our work revolves around the key insight of including class probabilities and bounding box information to the training of caption generating recurrent neural network.

The major challenge in our model is the understanding and learning of the inter-object relation of the different objects in the given image. However, working on deep learning, one faces different challenges ranging from steep-learning curve, data preprocessing, architectural-flow to fine tuning, high computational requirement. When we consider the general design of our model these are the major challenges we faced.

- Segment images into meaningful visual segments/tokens.
- Determine correlation between associated keywords and visual tokens.
- Determine the relation between objects in an image.
- Generate a sensible natural language description for the observed image.

With regard to the first problem, we rely on deep convolutional neural network detection system to identify image regions and their classification. As compared to earlier systems that used of low level features like color, texture, and shape to represent regions/objects, deep CNN models can understand high level features and make very accurate prediction. On top of that the vision model in use in our system treats bounding box of detection as regression problem, thus it uses a single pipeline for classification and segmentation.

Using a single pipeline for bounding box predication and classification, gives us unprecedented advantage to get an easy access to generating an image description and display the objects detected.

By adopting this final result into our language generating network we can achieve a robust model that understand the latent relation between detected objects, their spatial position and the relationship between these objects.

When we consider data preprocessing and preparation, the major challenge was optimization for high training speed and GPU optimization. We tackled this problem by generating preprocessed image representation vectors as will be described later.

The other was Hyper-parameter optimization to achieve a good learning, i.e. to prevent over or under fitting of the model. Our model has 50 time steps thus it is very easy to over-fit due to recurrent multiplication of weight, thus we implemented dropout and other regularization methods like clamping the weights to ensure a steady learning.

Our model is rather big one, thus on 12 Gb Titan X 1080 GPU we were only able to run 60 h.

2 Model design

Fast Image annotation is designed to use detection model with Natural language model (encoder and decoder) to generate automatic Image annotation for the given images. In this work we made an intensive analysis on different object detection models and automatic image captioning systems that lade us to design this novel architecture that implements the current state of the art fast detection model with the state of the art Language Translation Architecture.

This architecture is based on the robust detection model YOLO that treats detection as simple regression problem rather than classification over wide range of areas and scales as the preliminary models used. This architectural change over object detection, apart resulting unprecedented speed boost, it also lit a way to different potential improvements over different areas.

YOLO model sees detection as regression, thus the bounding box is also a learnable parameter unlike the previous models. The final output of YOLO includes the class probabilities for each grid cell, thus a single tensor will be encoding both the class probability and bounding box information. This means the class and their relative position is encoded as a single tensor output, which makes it easier to feed into another network for further processing. The final output Tensor YOLO model is $S \times S \times (B * 5 + C)$. Where the image is divided into $S \times S$ grid and each for each grid cell B number of bounding boxes are predicted and C is the class probabilities. Here if the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. This dense tensor contains a detailed unsorted image information on whereabouts of detected objects with its respective confidence level.

The intuition of this architecture is to use the learned box regression and class classification of the given image as an input for the recurrent neural network that learns to generate a sentence description based on the classes of the detected objects and their spatial position. This intern, to some extent encodes the internal relationship that exists between the detected objects (Fig. 1).

Yolo has two fully connected layers our model takes the final output of the fully connected layer of YOLO network and feed it as the initial state of our LSTM network.

2.1 Vision model

We can consider a CNN as feed forward neural network that can extract topological information of an image.

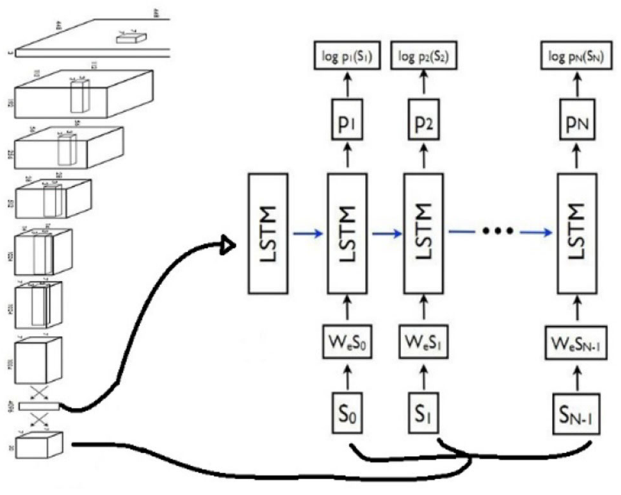

Fig. 1 Fast automatic image caption model

Convolutional neural networks have only recently become mainstream in computer vision applications. Over the past few years CNNs have achieved state-of-the-art performance in a broad array of high-level computer vision problems, including image classification [12], object detection [11, 13], fine-grained categorization, image segmentation, pose estimation, and OCR in natural images among others. Usually in these works, the CNNs are trained in an end-to-end manner and deliver strikingly better results than systems relying on carefully engineered representations, such as SIFT or HOG features. This success can be partially attributed to the built-in invariance of CNNs to local image transformations, which underpins their ability to learn hierarchical abstractions of the data.

We use YOLO small YOLO model with 24 layers of convolution and two fully connected layers. The output of the final FC layer is what we are concerned about. The final layer tensor has 7*7*30 shape. This tensor is flattened to be 1470 long vector, which is used as the initial state of our LSTM Network.

This Tensor contains the class probabilities of the detected objects and their spatial position (result of the bounding box regression network). This network will give a clear granular classification of image regions and their spatial positions in the given image. This intern infers the internal relationship that exist between detected objects (Fig. 2).

Using the class probability and bounding box information vector is our key insight as it tells our language model all the key objects detected in our model with their relative positions, this information can provide a clear high-level information to the language model.

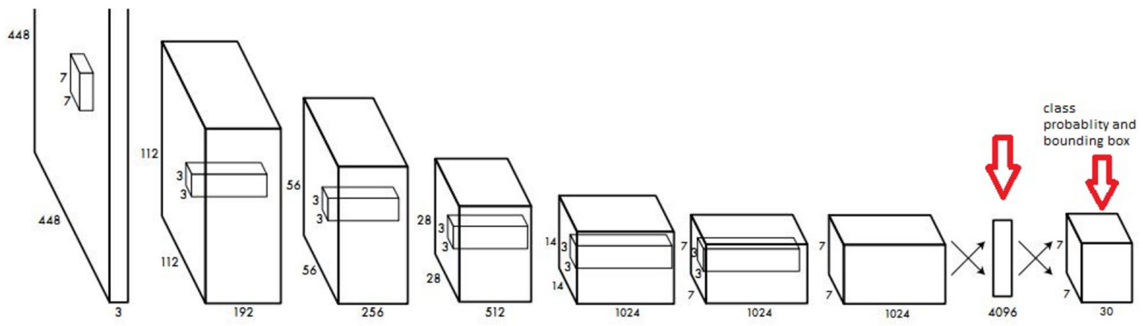

Fig. 2 The last two fully connected layers are used for image representation in our model

2.2 Language model

Our language model makes use of recurrent neural network (RNN) for generation of caption sentence for the given Image representation input. Recurrent Neural network is a kind of artificial neural network where connections between units form a directed cycle. These networks create internal state, which allows them to create dynamic temporal memory. Given a sequence of inputs $(x_1, x_2, x_3, \ldots, x_{n-1}, x_n)$, a standard RNN computes a sequence of Outputs $(y_1, y_2, y_3, \ldots, y_{n-1}, y_n)$ by iterating the following equation

$$\begin{aligned} h_t &= sigm\left(W^{hx}X_t + W^{hh}h_{t-1}\right) \\ y_t &= W^{yh}h_t \end{aligned} \tag{1}$$

where h_{t-1} and h_t are the previous and current state respectively;W^{hx}, W^{hh} and W^{yh} are the weights corresponding to the input x, hidden state h and the output y_t respectively. As can be seen from the equation there is an information flow within states through a sigmoid function symbolized with $sigm(.)$. The first part of the above equation is iterated to obtain the states of each RNN cell.This iterated multiplication and addition of similar terms without any causes unstable gradient flow.

The basic RNN models suffer from vanishing or exploding gradients. Vanishing gradient is a phenomenon where the gradient tends to be smaller as we move backwards through the time or sequences. On the other hand, exploding gradient problem have quite the opposite effect on the gradient. In the latter case, the gradient will grow exponentially as the gradient traverses through time. Note that the fundamental problem is not vanishing or exploding gradient rather it is because of neural networks suffer from unstable gradient. One of the models proposed to amend this problem is LSTM (Long-Short term memory) and another is GRU (Gated Recurrent Unit). Both LSTM and GRU have proven to yield superior performance in comparison with the traditional method [14, 15]. The LSTM Cell Block can be express in the following formula.

$$\begin{aligned} i_l &= \sigma(W_{ix}X_l + W_{im}M_{l-1} + b_i) \\ f_l &= \sigma(W_{fx}X_l + W_{fm}M_{l-1} + b_f) \\ O_l &= \sigma(W_{ox}X_l + W_{om}M_{l-1} + b_o) \\ C_l &= f_i \odot C_{l-1} + i_l \odot h(W_{cx}X_l + W_{cm}M_{l-1} + b_c) \\ M_l &= O_l \odot h(C_l) \end{aligned} \tag{2}$$

where i_l, f_l, O_l represent input, forget and output gets of LSTM cell l respectively and C_l represent cell state of the LSTM network. $\sigma(.)$ and $h(.)$ represent a sigmoid and tanh (tanhyperbolic) function. The operation $\odot$ is to an element wise multiplication. The weights of the LSTM cell W_{ix}, W_{fx}, W_{ox} correspond to x for input gate, the forget gate and the output gate respectively. On the other hand W_{im}, W_{fm}, W_{om} and b_i, b_f, b_o are the counterpart weights for the hidden vector and biases of the LSTM cell. Finally, W_{cx} and W_{cm} are the weights of the input x and the previous state M_{l-1}. Similarly, b_c is the bias associated with these weights.

Note that the first three gates employ a sigmoid function. One property of sigmoid function is that it has a range of (0, 1). This implies these functions serve as a simple switch, which controls the flow of information through them. The flow information can range from approximately full flow to approximately nil.

On the other hand, the tanh function is like a get that controls the gradient flow of the LSTM cell. tanh function have a range of (– 1, 1), this shows it is a perfect get to control the gradient flow of the network. This can have a negative or positive gradient.

The Language model of our work is based on the recent success in Language Translation model, which is sequence-to-sequence model. The language model used our annotation model which is similar to NIC model from Google [7] follows this sequence to sequence model, However the model use image representations instead of language sequence.

Deep neural network are powerful models unfortunately, it is difficult to train them in sequential data this is because these networks require the input dimension to be fixed.

However, using a straightforward application of LSTM Networks solve the sequence-to-sequence problem robustly. This model is sensitive to sequence order and performs better when the target sequence is reversed as studied on language translation French to English [16].

RNN's can easily map sequence-to-sequence inputs whenever the alignment between the input and the output is known ahead of time. However, the application of RNN networks in complicated and non-monotonic relationships is uncharted waters. A simple yet a genius solution to this is to map the input sequence to a fixed sized vector using One RNN and then map the vector into the target sequence.

In our image annotation model we use the right end of the sequence to sequence model, however we made different architectural variation in order to pass an optimal representation of the image through our LSTM model.

Compared to preliminary image caption models we believe our architecture can better represent an image for generating a comprehensive image description as it contains the localization of the detected object and its classification in a single vector.

Finally, our model directly maximizes the probability of the correct sentence description given image by using the following formula.

$$\lambda^{*} = argmax_{\lambda} \sum_{(I,S)} \log(S|I;\theta)$$
$$\log(S|I) = \sum_{t=0}^{N} \text{logp}(S_t|I, S_0, \ldots, S_{t-1}) \tag{3}$$

where λ is the parameters of the model, $S, (S_0, \ldots, S_{t-1})$ represent the full caption and fragments of the Image caption sentence and I is an image representation. Here θ are the free parameters of the model and I is the input image representation. Now the model optimizes for the supremum of the conditional probability of the caption over an image I then it calculates for the joint probability over $S_0, \ldots, S_{t-1}$ using the chain rule.

Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep nets can understand.

Differences between word vectors, as they swing around the origin like the arms of a clock, can be thought of as differences in degrees. We will measure the angular distance between words using something called cosine similarity. We normalize the measurements so they come out as percentages, where one means that two vectors are equal, and zero means they are perpendicular, bearing no relation to each other.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\left\|\vec{b}\right\|} \tag{4}$$

In our experiments, we used Glove (global vector for word representations) which is an unsupervised learning algorithm for generating vector repetition of words. To calculate the nearest neighbors, the Euclidean distance or Cosine similarity between two word vectors will provide effective method for measuring the sematic similarity of the corresponding words. The cosine of two non-zero vectors a and b can be derived by using the Euclidean dot product formula. The cosine similarity, cos(θ), is represented using a dot product and magnitude as shown in Eq. (4). The resulting similarity ranges between − 1 and 1. That is − 1 meaning exactly opposite and 1 meaning exactly the same, with 0 indicating decorrelation, and in-between values indicating intermediate similarity or dissimilarity of the two vectors.

The input sequence in our model is a word to vector representation of the given sentence sequence. In our experiments, we used 100-d word to vector representation of Glove word to vector trained over Wikipedia 2014.

However, using this representation directly as whole will unnecessarily inflate our vocabulary size, thus by increasing our final FC layer size beyond our processing capacity. To prevent this, we generated a 20,000-vocabulary size out of the training and validation annotation dataset combined with other data corpus (to fill the 20,000-vocabulary size). We used the word to vector representation of these words from Glove representation.

3 Architectures

Our models leverage high-level image representation to generate image description. Based on this core idea we experimented on different architectures to achieve a better and faster performance.

3.1 Class probabilities bounding box information as thought vector

Here we used the class probabilities and bounding box vector as a thought vector for the right side of Sequence to sequence model. The thought vector in sequence to sequence model will be a representational vector of our input image, which is the final output of YOLO model. In this case the thought vector has a dimension of 1470. This vector contains the class probability and the box regression information of the given image.

This model lacks more detailed information, thus forgets the image representation easily. This results on high loss and low accuracy. Another architecture to prevent the network from forgetting the image high-level information is to continuously remind the recurrent neural network each time step.

3.2 Using transformation layer for binding inputs

In theory there are different architectures that can generate a sentence description out of given image representation and word representation vectors. One possible way is to take the output of the convolutional neural network and fed it as an initial hidden layer of the LSTM neural network. On the other hand feeding the initial sentence-beginning token to the initial input. The model will progress feeding the previous prediction to the next input. However, in this model, as the model progresses predicting the next word it will quickly forget the content of the image.

In the effort of trying to prevent our network from forgetting the image, in this method, we used a single neural network layer (transformation layer) that transforms the inputs of the network to uniform dimension, in this particular case 200-d vector. This means the output of the convolutional neural network will be transformed to a 200-d vector, which later, will be joined with the respective word input. Notice that the embedding vectors of the vocabulary also have to go through their respective transformation layer to get an output of 200-d vector representation. These vectors then can be added with the image representation. Note that the final output of the detection will contain the image class probability and bounding box vector.

The class probability and bounding box vector has dimension of 1470 and the output expected is of 200-d vector, thus the weight for the network will be a matrix of size (1470, 200). On the other hand, for the word embedding we have a weight matrix of size (100, 200), that is because the word embedding vector is a 100 dimensional vector.

The output of these neural network layers is a fixed sized vector (200-d). As seen in the Fig. 9, the image detection output and the word embedding will be fed to the transformation layer. The transformation layer (the second layer in the figure) will generate a fixed size representational vector for both the image representation and word embedding. This out will be added element wise and fed to each LSTM cell accordingly. This way we supply a high-level sematic information about the image with each time step.

In this model, the initial thought vector is initialized as random float and it is not trainable. This model showed significant loss decrease and increase in accuracy.

3.3 Unified model

Our unified model takes advantage of both the above systems. In this model instead of initializing the thought vector with random initialization, which does not give any information to the network, we initialize it with last layer of convolution feature map. This way the recurrent neural network will have access to both low-level representation (detail) representation and high-level image representation to which it can refer repeatedly.

3.4 Other architectures

We have tried different other architectural changes to find the best image annotation model given the image representation and word sequence as an input. One significant architecture that has showed promising result was an implementation of the model with a modified input containing the concatenation of word sequence, 1470-D image representation and 4096-D image representation. To generate this concatenation, first each individual input is passed through a transformation layer generating an output of 200 dimensional vector. On this architecture we used 600 LSTM units with 64 batch size and 20,000 vocabulary size. We also used RMSProp optimization with initial learning rate of 0.05. We also implemented a regularization with regularization coefficient of 0.01. However, the model was quickly exposed to overfitting problem.

Event though, we have implemented different architectures with combination of different hyper-parameters we will only take an in depth look at the architecture we deemed to be promising.

4 Implementation details

The implementation of our model follows strictly object oriented programming structure, which enables us to design multiple architectural variations during training. In each variation, we try to find what works best. As described in above sections our model comprises of two sub-modules working as a single system. The first is the computer vision sub-system for which we used YOLO object detection system and the second is probabilistic language model. The language model is adopted from the famous sequence-to-sequence model.

When we are considering the architectures most of their implementation is similar. However, there are some significant difference on input transformation layers used, the input of the language model and different hyper-parameters. There could also be difference on which optimizer used and implementation of regularization. In this section we will discuss the general implementation of the model.

4.1 Implementation

The implementation of our model is rather a simple and straight forward. The first stage is to generate image high-level representations from the training dataset. These representations will be paired with their respective sentence caption and processed using YOLO object detection model.

These pair of images and respective captions are stored on the disk, this is a straightforward implementation technique used to speed up the training process and prevent the model from reprocessing the same batches of images in each epoch.

We have generated these epochs for batch size of 64 and 128. These batch files are generated from the COCO image dataset that has been downloaded and processed. Each batch files contain two fully connected layer vectors from YOLO, an input sequence and a target sequence. Here the input sequence is a vector of word sequence embedding generated from the image caption and the target sequence is one-hot encoding caption sequence with shifted index by one. Using these ground preparations, we have experimented on different architectures of Image annotation model.

Here we will take an in-depth look at our input and expected output parameters. For the training, we need four multi-dimensional parameters these are: -

- Image representation
- Input sequence
- Output sequence
- Input mask.

4.2 Hyper parameters and overfitting

During the implementation, different hyper-parameters were defined. Each of these parameters affect the behavior of our model one way or another. These hyper-parameters are defined as follows: Initial learning rate of 2, optimization (Adagrad), number of LSTM Units 1470, dropout rate of 0.6, sequence length of 50, batch size of 128 and an embedding size of 100. Note this hyper-parameter configuration is tied to the result given below. We also have implemented different configuration with each architecture in search of the best accuracy result. We noticed that increasing the LSTM units beyond 1500, would cause lot of problems including overfitting, high time and space complexity and above all and surpassingly, we found resource limitation. The model was constantly flowing out of the GPU pool limit.

On another implementation, we have used a transformation layer of 4096 by 200 for the second FC layer of the YOLO output, 1470 by 200 for the last FC layer of YOLO and 100 by 200 for word sequence embedding. For this variation, we have used RMSProp optimizer, 600 LSTM units, an initial learning rate of 0.05 with implementation of dropout (rate of 0.6) and L2 regularization on the transformation layer.

5 Dataset and experiment

In our experiments, we used coco dataset for image captioning benchmark. We used the JSON annotation for image caption coco dataset to download and preprocess the dataset. As explained in the above sections we used a pre-trained YOLO model for object detection and trained our LSTM model for captioning, thus for training speed boost we generated pairs of YOLO output tensor and caption input with a size of batch and dump them on the disk. This technique gave us remarkable training speed boost. YOLO can process approximately 155 frames per second. However, it is much faster to load the preprocessed output from file on the other hand loading the batch from disk will also prevent the need to process each image over and over again on each epoch. After generating batch files on the disk, we generate an input, targets and mask from the given caption on the fly. This does not require too much time; thus, it does not have much effect on the training speed.

As mentioned above our model takes 50,000-dictionary size. This dictionary is generated from the most frequent 50,000 words in the training caption data. After this 20,000 of these most frequent words are extracted from the intersection between the dictionary and Glove word2vec embedding. We used these 20,000 words and their vector representation for the training of our model after generating a dictionary out of this set.

5.1 Experiments

We tested different types of architectures in order to minimize the loss of our model. However, we kept the key ideology of the research, which is to integrate an object detection system that takes the object spatial location into its learning parameters. There are two Major Deep Neural networks that are able to achieve this, Faster RCNN and YOLO. Since our models desired key feature is speed, we decided to go with YOLO, but this choice change didn't come without a tread-off. In our case YOLO is performance lag as compared to other state of the art detection systems like Faster RCNN.

In our work we used YOLO model. Implementing our model, we experimented with different architectural

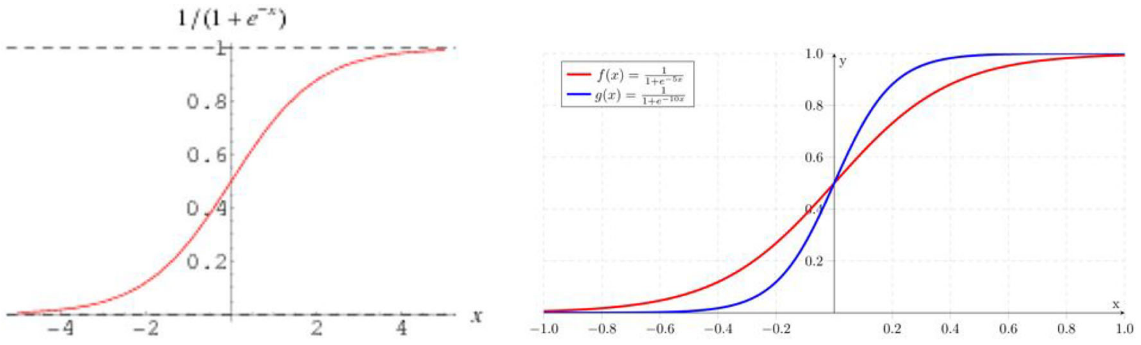

Fig. 3 Sigmoid function

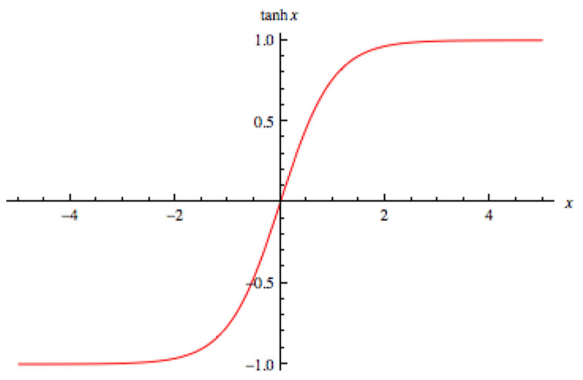

Fig. 4 tanh function. It has an asymptote at (− 1, 1)

variation on top of different hyperparameters for our deep neural network. The first architecture follows Google's work img2txt as it feeds the image-embedding vector as a thought vector for the right side of sequence-to-sequence model.

Note that the YOLO object detection system has two fully connected layers. These FC layers have a dimension of 4096 and 1470 respectively. The last FC layer, which is 1470-d vector, represents the class probabilities and box regression result of the object detection system.

5.2 Results

In this paper more than two architectural designs were proposed however only the result of two architectures are presented. This is mainly due to limitation of resource and time. However, the other architectures are currently underway. For this thesis report, we have displayed the mid progress of two of our working architectures in detail. We also will discuss the failure of our initial model and its main reasons.

Our initial model was using the last fully connected layer of YOLO model as a thought vector for our Natural Language model (LSTM) (Figs. 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13). The LSTM in our model have 50 sequences, thus

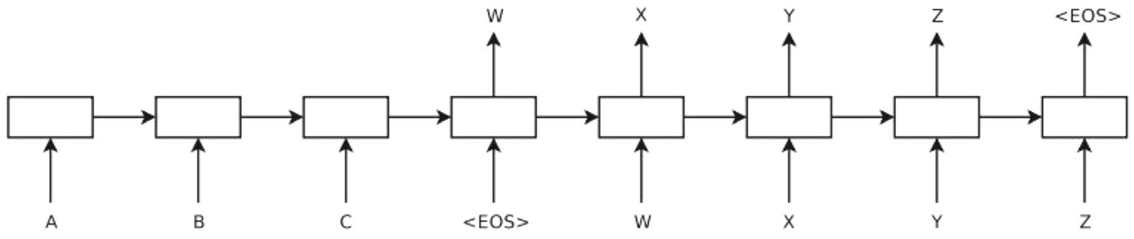

Fig. 5 LSTM sequence-to-sequence model

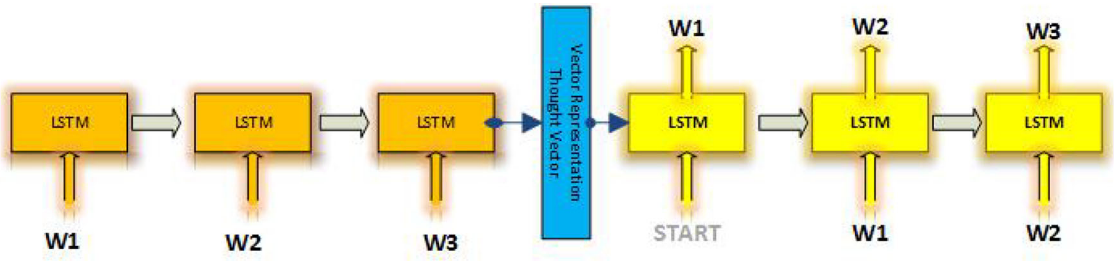

Fig. 6 LSTM language encoder decoder model

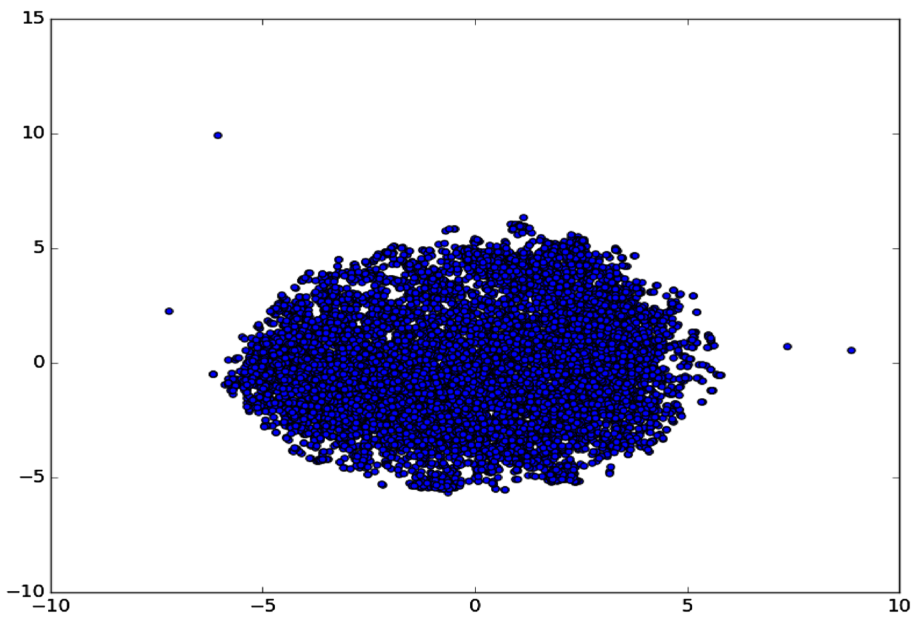

Fig. 7 Word to vector representation for 10,000 words of our vocabulary, two-dimensional representation of the original 100-embedding vector

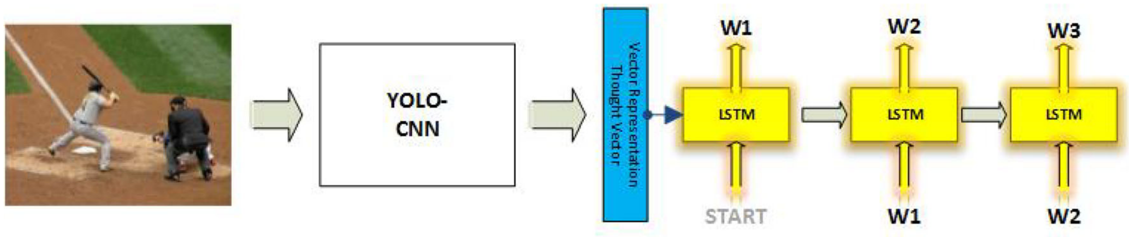

Fig. 8 Image to sequence model. Initial model which feeds a single FC layer (1470) of the image detection to the LSTM model

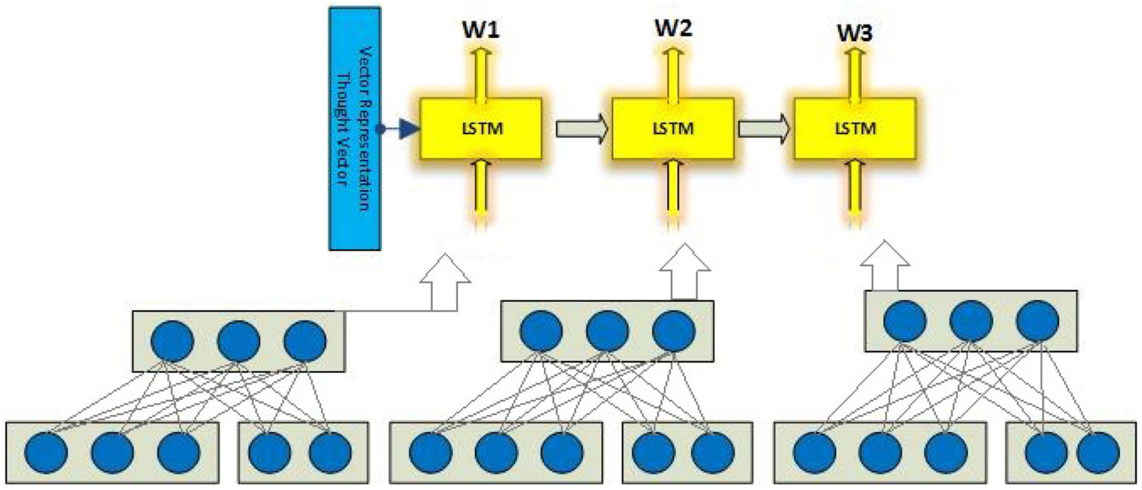

Fig. 9 Using Transformation Layer for image representation and word embedding feeding. On this variation, we feed the last FC layer (1470) of the image detection model to each time step of our LSTM time units by concatenating the image data with the word imbedding

feeding a high level as the thought vector in our model results with high loss as the model forgets the image reference as we go through the time steps of the LSTM model. This as result generated a very low accuracy.

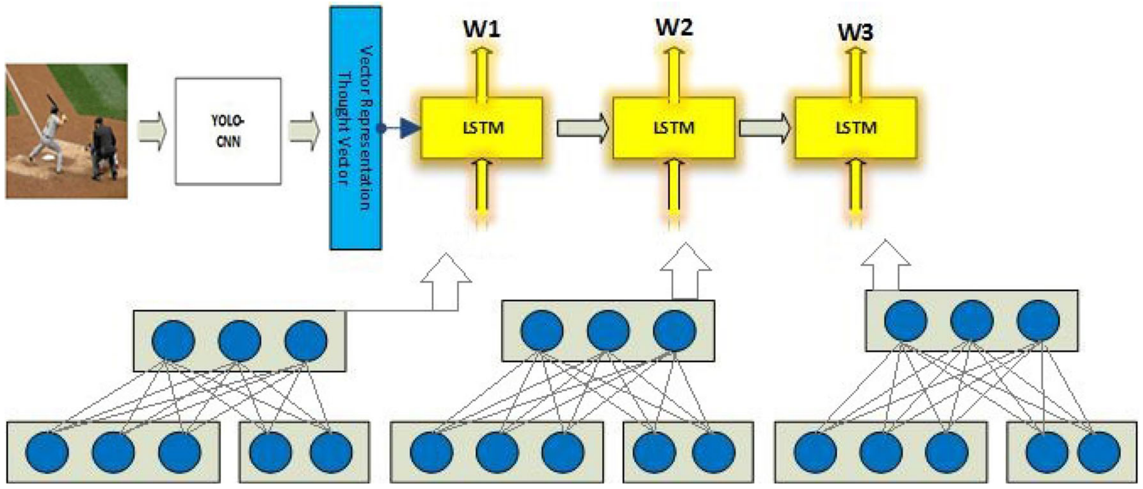

Fig. 10 Unified Model. This version takes two FC layers from YOLO output (1470 and 4096) to represent an image. This representation will be fed in as an initial vector (4096) for the LSTM model and as a concatenation of the input vector at each time step (1470)

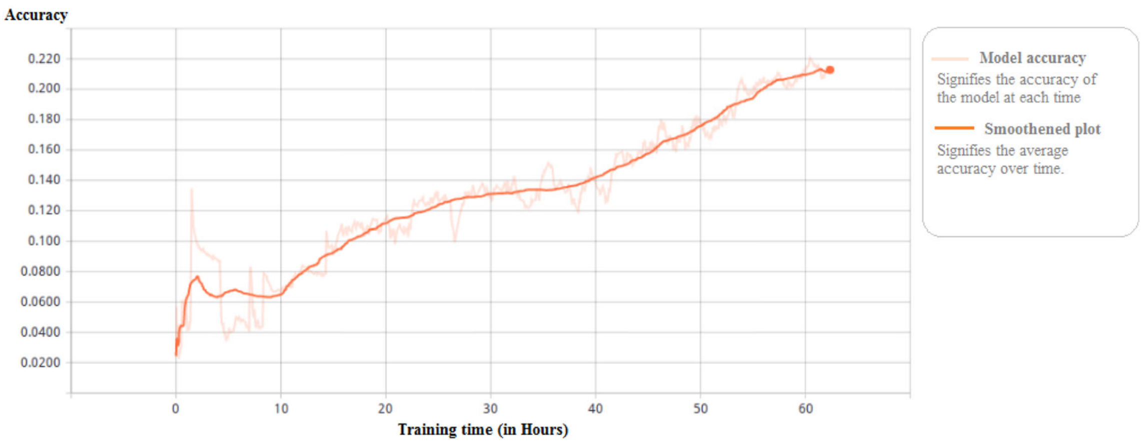

Fig. 11 Accuracy for the second architecture. Note this graph does not show the complete training of our network (resource constrain-interrupted). However, we can see a robust learning of the network. At the time of training termination, the model was around 23% accuracy and steadily climbing

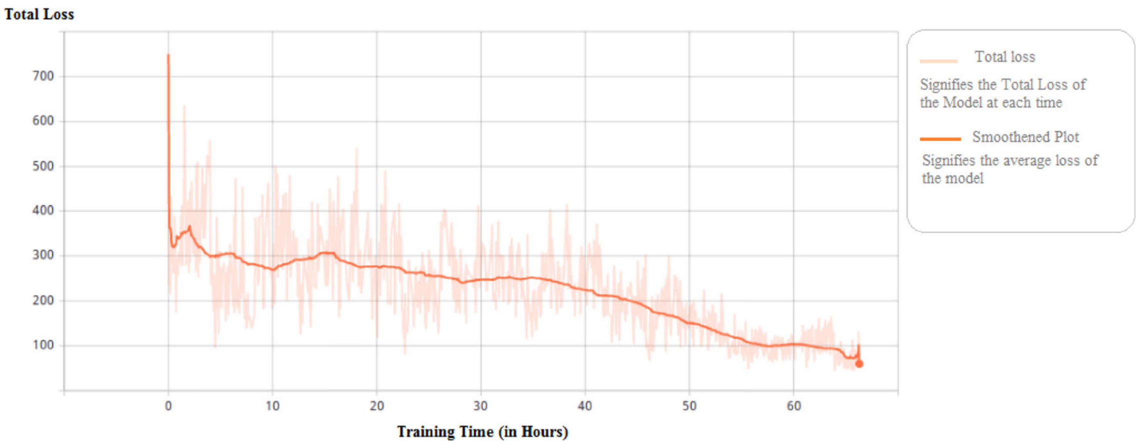

Fig. 12 Steady batch loss decreases is observed

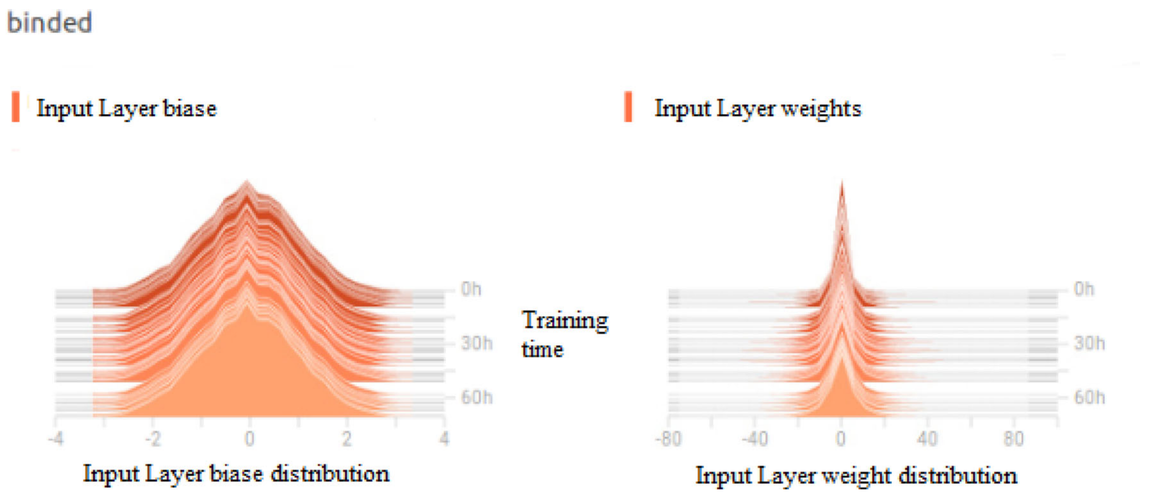

Fig. 13 Transformation layer for Image representation and word2vec embedding. As shown in the picture the weight distribution of the transformation layer is well normalized. As shown in the first figure (bias) and the second figure (weights) the gradient clamping is implemented with a value of a minimum − 5 and maximum of 5. This will reduce overfitting problems

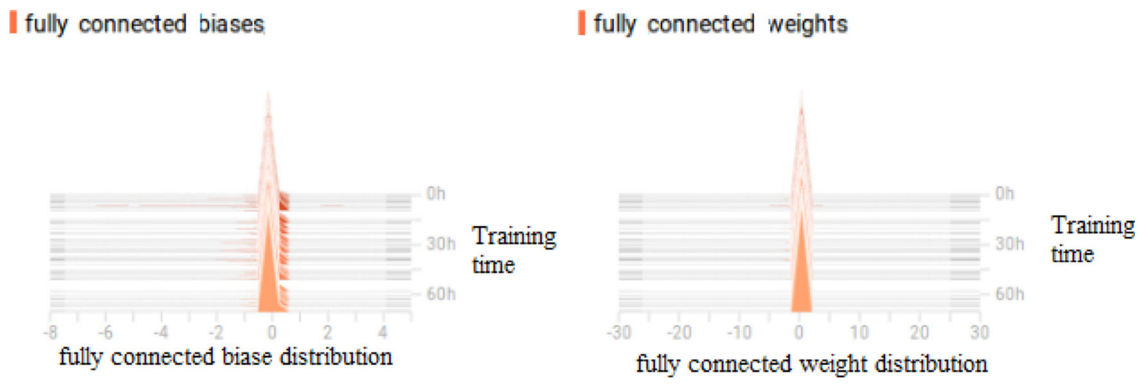

Fig. 14 Weight and bias distribution of the fully connected layer of the LSTM model. This shows that most of the values are close to zero and the rest fall between − 1 and 1

To amend this problem, we created a transformation layer under each time sequence feed of the LSTM model. These single-layered transformation neural network are created both for word embedding transforming a 100-d vector to 200-d vector and image representation transforming 1470-d vector to 200. These resulting vectors are added to get a single input for our LSTM model. This simple yet innovational architectural change created a robust model with a very steady accuracy incline during training and also well normalized gradients across the network.

After some time the accuracy grows in more steady and calm manner. This is a very good sign for a good learning. This value persists for both the training and validation data.

As show in the above figure the total loss exhibits similar properties with the increase of accuracy, as both graphs show rampant oscillation, but with time the loss will assume a more stable loss decrease. Which also show a good training.

Other variables also exhibited a well-normalized distribution throughout the training. On top of that, all the weights were clamped in our experiments this is a maximum of 5 and minimum of -5 clamping range was used (Fig. 14).

The above two pictures show the weight and bias distribution of the model on the transformation layer and the fully connected layer respectively we were able to reduce overfitting by implementation of gradient clamping the network as a result variable also exhibited a well normalized distribution throughout the training.

6 Conclusion and future work

As mentioned this work have not been fully trained, thus it is difficult to speculate to total accuracy of the fully trained model, however the current results seem promising, but this does not mean we haven't got a keen insight in what works and what does not.

From our experiments, we have observed that using class probability and bounding box as initial thought vector without allowing the LSTM model to revisit the image information generates bad performance while generating text description. On the other hand, initializing the LSTM model with random initialization and feeding image representation and word embedding through a transformation neural network layer and binding the result generates much better result, however this puts a time lag during training. To amend this problem, we can reduce the number of hidden layers in our LSTM network to generate much faster result. For this specific model anything greater than 600 layers is a reasonable choice. However, we also have to be careful not to over fit our model by stuffing unnecessary huge number of hidden layers.

In this work we showed that using high representation it is possible to generate excellent image description. On top of that our model understands image relation much better form the image classification and their relative position.

Our model is limited in a sense that it learns from high-level representation of the detected objects, thus this for one means that the model works much better as the number of detections increase. In this model we used 20 classes for detection, which limits the types of stuffs that can be detected in our model. This directly reflects on the variety of caption we can generate from it. We also do not include property descriptors; however, this can be easily amended by just feeding the last layer feature map to initialize the LSTM units.

This work open new uncharted opportunities in the way we describe our images. One promising future work would be embedding objects using the detection representation jointly with the vocabulary creating a multi-modal embedding space which defines the image region with their respective word embedding. This embedding also need to understand the object to object relation.

References

1. Picard, R.W., Minka, T.P.: Vision texture for annotation. Multimed. Syst. **3**(1), 3–14 (1995)
2. Cusano, C., Bicocca, M., Bicocca, V.: Image annotation using SVM. Proc. SPIE **1**, 330–338 (2003)
3. Tang, J., Lewis, P.H.: A study of quality issues for image auto-annotation with the corel dataset. IEEE Trans. Circuits Syst. Video Technol. **17**(3), 384–389 (2007)
4. Li, J., Wang, J.Z.: Real-time computerized annotation of pictures. IEEE Trans. Pattern Anal. Mach. Intell. **30**(6), 985–1002 (2008)
5. Li, J., Wang, J.Z.: Automatic linguistic indexing of pictures by a statistical modeling approach. IEEE Trans. Pattern Anal. Mach. Intell. **25**(9), 1075–1088 (2003)
6. Jeon, J., Manmatha, R.: Using maximum entropy for automatic image annotation. Proc. CVIR Lect. Notes Comput. Sci. **3115**, 24–32 (2004)
7. Vinyals, O., Toshev, A., Bengio, S., et al.: Show and tell: a neural image caption generator. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Boston, MA, 07–12 June, pp. 3156–3164 (2015)
8. Karpathy, A., Li, F.F.: Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 07–12 June, pp. 3128–3137 (2015)
9. Kulkarni, G., Premraj, V., Ordonez, V., et al.: Baby talk: understanding and generating simple image descriptions. IEEE Trans. Pattern Anal. Mach. Intell. **35**(12), 2891–2903 (2013)
10. Girshick, R., Donahue, .J, Darrell, T., et al.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
11. Ren, S., He, K., Girshick, R., et al.: Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017)
12. Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, 4–9 Feb, pp. 4278–4284 (2017)
13. Redmon, J., Divvala, S., Girshick, R., et al.: You only look once: unified, real-time object detection. In: CVPR 2016, pp. 779–788 (2016)
14. Karpathy, A., Johnson, J., Fei-Fei, L.: Visualizing and understanding recurrent networks. arXiv:1506.02078 (2015)
15. Chung, J., Gulcehre, C., Cho, K., et al.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555 (2014)
16. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS, 2014, pp. 3104–3112 (2014)

Meng Han received his Ph.D. degree of Computer Science in June 2015 at University of Electronic Science and Technology of China. He is currently an associate professor at Information and Information Systems Major University of Electronic Science and Technology. His research interests include E-government theory, advanced programming language and research on object oriented technology, natural language processing.

Wenyu Chen received his Ph.D. degree of Computer Science in December 2009 at University of Electronic Science and Technology of China. He is currently a professor at School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include pattern recognition, natural language processing and neural network.

Alemu Dagmawi Moges received his Master degree of Computer Science in December 2017 at University of Electronic Science and Technology of China. He is currently a doctoral candidate at School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include image annotation, knowledge representation, and text mining.