<u>Technical Communications Report</u>

Image2Text
or
A Neural Image Caption Generator

PREPARED BY:

SANJU PRABHATH REDDY	**(15114042)**
SRI HARSHA MAJETI	**(15114044)**
HARSHA VARDHAN MIRYALA	**(15114045)**

BATCH - CS2

Table of Contents

Image2Text

Introduction:

What is Image captioning?

A short piece of text under an image that describes the picture or explains what the object(maybe people) in it is/are doing.

Some of the examples are:

A person is walking along a beach with a big dog

A black and white dog carries a tennis ball in its mouth

A soccer player takes a soccer ball in the grass

A man is doing a trick on a snowboard

A surfer dives into the ocean

A black and white dog leaps to catch a Frisbee

In this report we delve into topics regarding captioning of images.

Describing the contents of an image is one of the fundamental problems of artificial intelligence which needs and connects both Computer Vision and Natural Language Processing. Our goal is to explain the general deep recurrent architecture that uses both these fields to output natural sentences and descriptions of an image simulating human behaviour. i.e our goal is to generate a Neural Image Caption Generator.

Model Architechture:

The main steps involved in the architecture are:

CNN-based feature extractor:

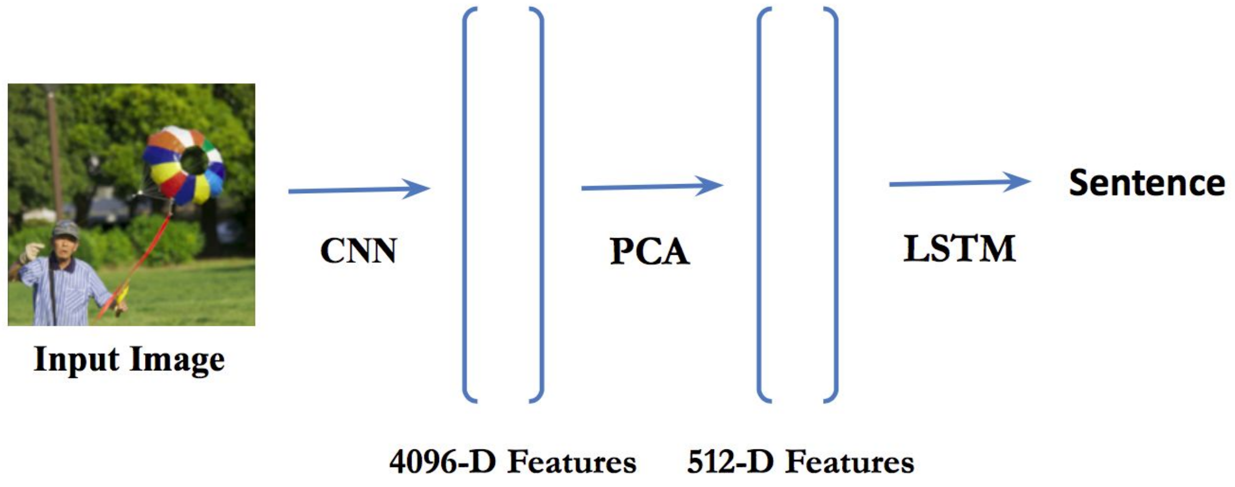
For feature extraction, we use a CNN (Convolutional Neural Networks). CNNs have been widely used and studied for images tasks, and are currently state-of-the-art methods for object recognition and detection . Concretely, for all input images, we extract features from the fc7 layer of the VGG-16 network pre-trained on ImageNet , which is very well tuned for object detection. We obtained a 4096-Dimensional image feature vector that we reduce using Principal Component Analysis to a 512-Dimensional image feature vector due to computational constraints. We feed these features into the first layer of our RNN or LSTM at the first iteration.

RNN-based Sentence Generator:

We first experiment with vanilla RNNs as they have been shown to be powerful models for processing sequential data. Vanilla RNNs can learn complex temporal dynamics by mapping input sequences to a sequence of hidden states, and hidden states to outputs. We will see them more in detail below.

LSTM-based Sentence Generator: (an extension)

Although RNNs have proven successful on tasks such as text generation and speech recognition , it is difficult to train them to learn long-term dynamics. LSTM networks provide a solution by incorporating memory units that allow the networks to learn when to forget previous hidden states and when to update hidden states when given new information. More about LSTM's below.



Let’s go with feature extraction using Computer Vision first and then text generation using NLP.

Computer Vision In Image2text

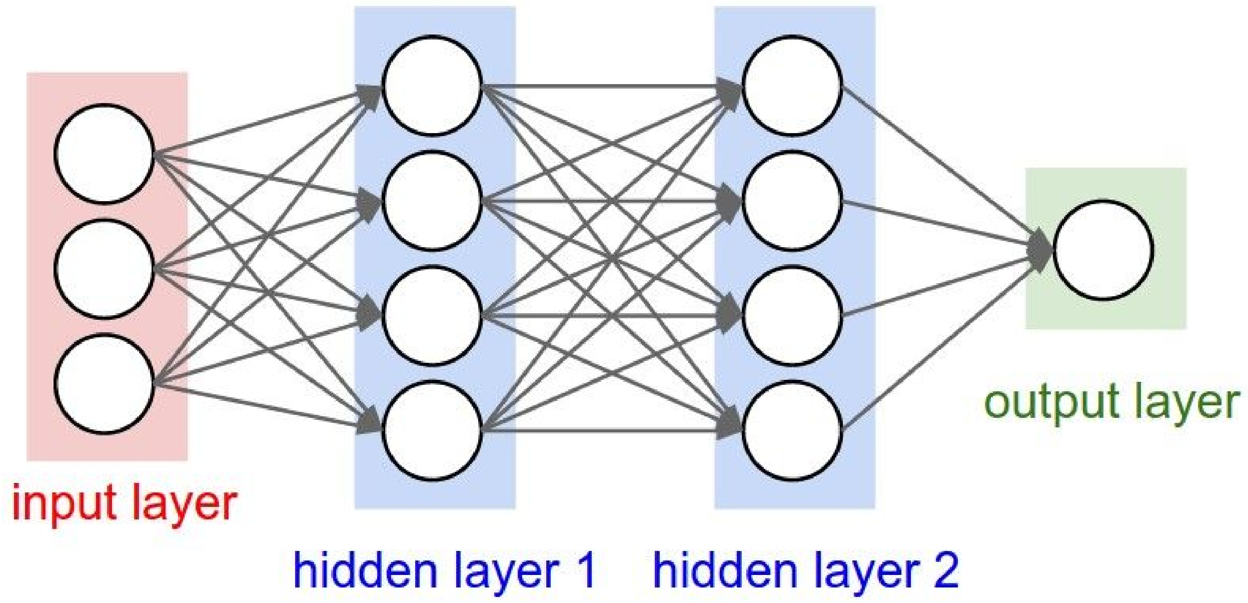
Our goal in this particular part of the task is to extract the features of the image.We can achieve it using convolutional neural networks.

- To be precise we use a pre-trained CNN on ImageNet so as to obtain the features of our image

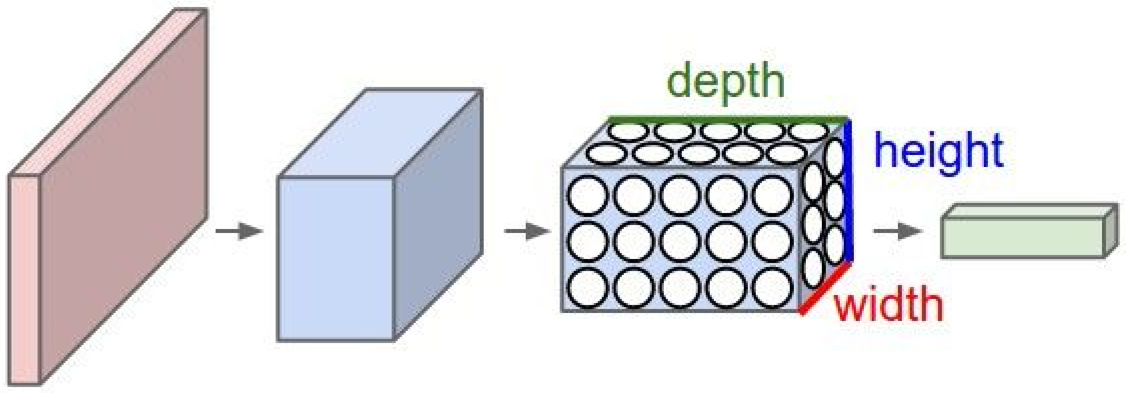
What is a Convolutional Neural Network(CNN)?

Unlike the traditional multi-layer neural network CNN contains neurons in 3 dimensions i.e width,height,depth.

Regular Neural Network



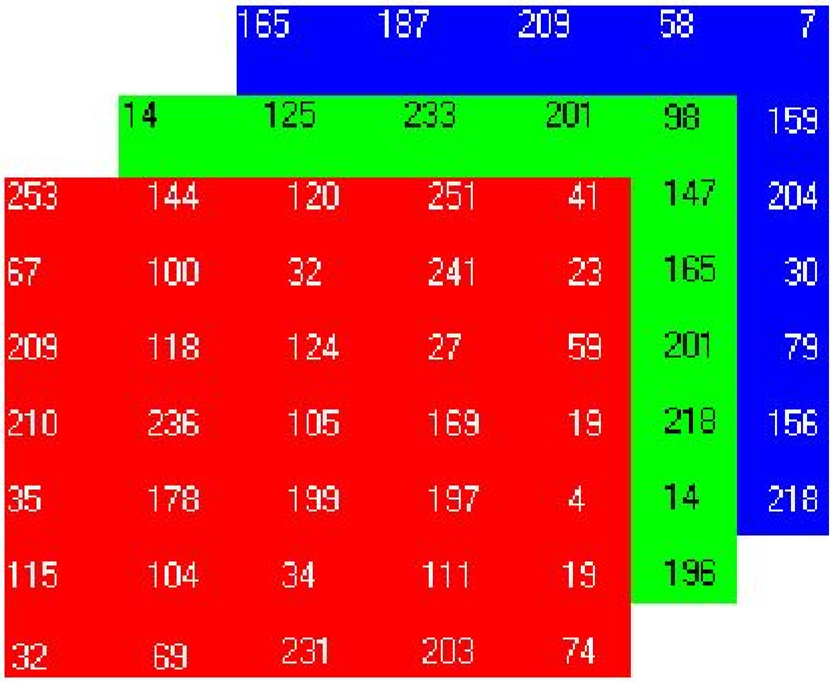
CNN



In our task we use a pre trained CNN which means we use a convolutional neural network which is already trained on a database of images called ImageNet

Input Image for CNN

The input image generally contains **depth of 3** because any computer image is described using red,blue,green layers where each layer contains pixel values related to that image.We feed this particular image to convolutional networks to extract the features of the image.



Description of a CNN

A CNN contains an input and an output layer, as well as multiple hidden layers. The hidden layers are either convolutional, pooling or fully connected.

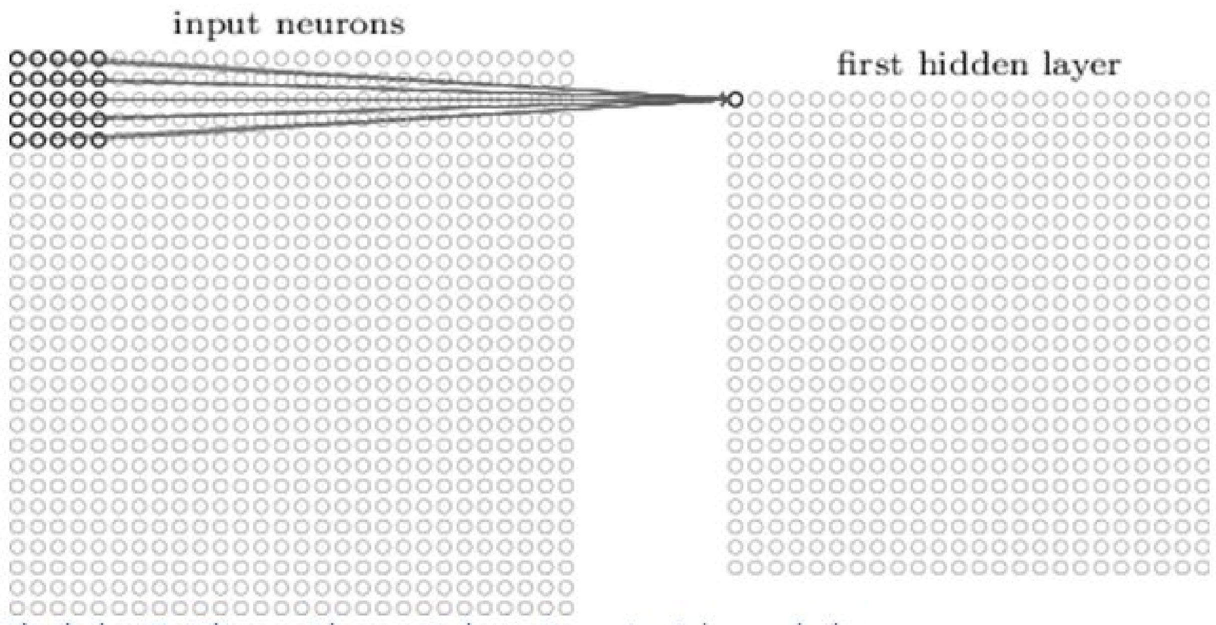
Convolutional

Convolutional layers apply a convolution operation to the input, passing the result to the next layer.For example we take an input image 32*32*3.Now We take a filter of dimension say 5*5 which must be of the same depth of the input image.So our filter is of dimensions 5*5*3.Each of these filters can be thought of as **feature identifiers.**Now we slide(convolve) the filter along the input image by 1 unit (generally).As we keep sliding we multiply the corresponding pixel values in the input image with the weights of our filter and store in the next hidden layer.If we do the same process for the whole input image we get a matrix of numbers called **activation map**.

In this example activation map is of the dimensions 28*28*1 if we use only one filter.

If we use ‘x’ filters it will be 28*28*x.

See the below diagram.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

What is the physical interpretation of the above operation?
The filters on the first layer convolve around the input image and “activate” (or compute high values) when the specific feature it is looking for is in the input volume.

Along with this there are also other layers like pooling and fully connected layers.

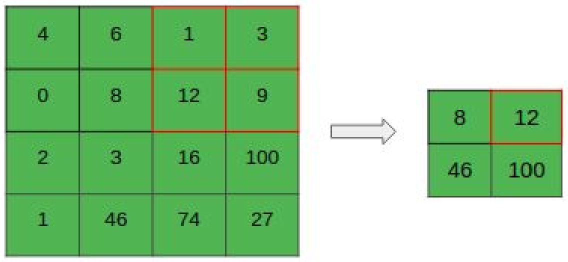
<u>Pooling</u>
CNN may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer.

Generally we use two types of pooling

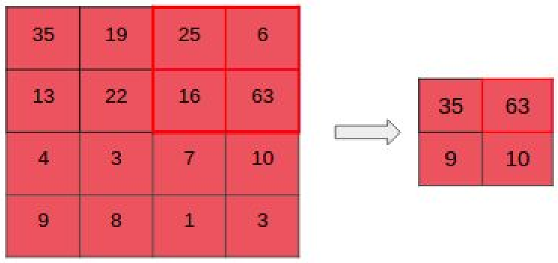
- **Max Pooling**
  Maximum value from each of a cluster of neurons at the prior layer is stored in the next layer.
- **Average Pooling**
  Average value from each of a cluster of neurons at the prior layer is stored in the next layer.

Pictorial Representation:

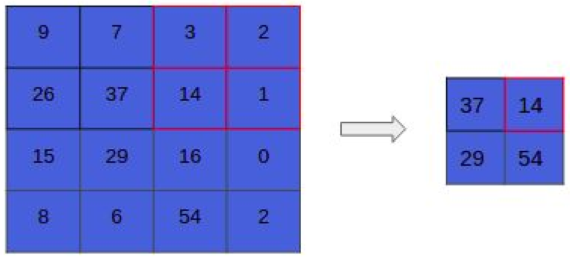
Max Pooling with a stride of 2 by the filter



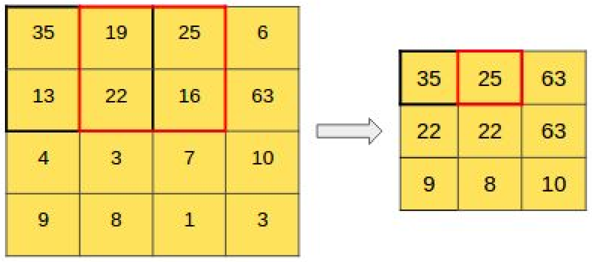
(i)



(iii)



(ii)



(iv)

Fully Connected layer

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer neural network.

From the above example of different pooling layers obtained through pooling are written in a single layer where each represent a neuron.So finally the fully connected layer from the above gives **(4+4+4+9)*1 matrix**.

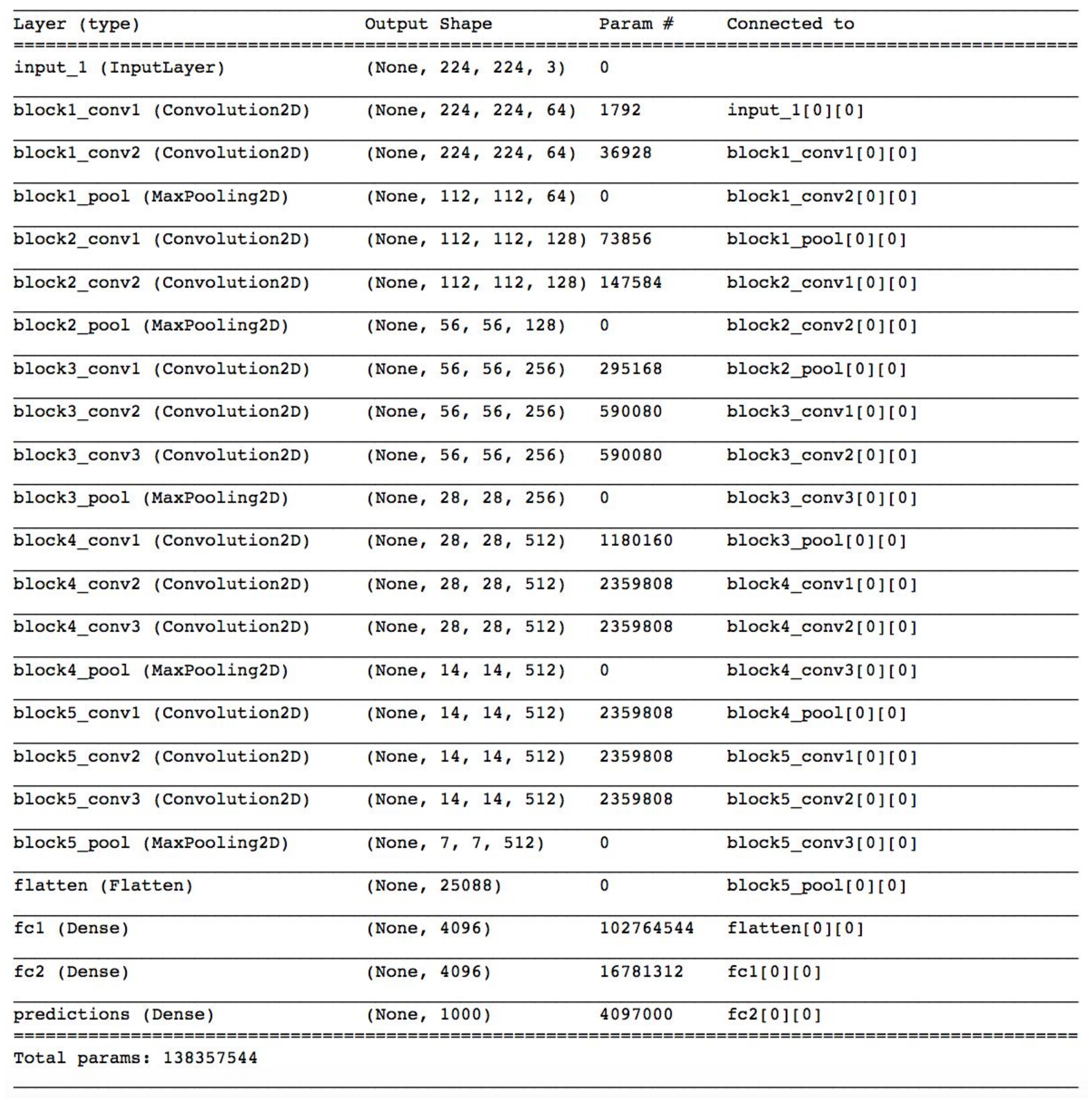
Now the traditional neural network analysis is done if we want to classify the image into different categories.

A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Where ReLU represents the activation map where all the negative values in the matrix are replaced with zero and then passed to the next layer.
At different layers of CNN different features of the input image are recognised by the neural network.
There are several architectures in the field of Convolutional Networks.Some of them are

AlexNet,ZF Net,GoogLeNet,VGGNet
For a VGGNet the following layers are used.



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 224, 224, 3)   0
____________________________________________________________________________________________________
block1_conv1 (Convolution2D)     (None, 224, 224, 64)  1792        input_1[0][0]
____________________________________________________________________________________________________
block1_conv2 (Convolution2D)     (None, 224, 224, 64)  36928       block1_conv1[0][0]
____________________________________________________________________________________________________
block1_pool (MaxPooling2D)       (None, 112, 112, 64)  0           block1_conv2[0][0]
____________________________________________________________________________________________________
block2_conv1 (Convolution2D)     (None, 112, 112, 128) 73856       block1_pool[0][0]
____________________________________________________________________________________________________
block2_conv2 (Convolution2D)     (None, 112, 112, 128) 147584      block2_conv1[0][0]
____________________________________________________________________________________________________
block2_pool (MaxPooling2D)       (None, 56, 56, 128)   0           block2_conv2[0][0]
____________________________________________________________________________________________________
block3_conv1 (Convolution2D)     (None, 56, 56, 256)   295168      block2_pool[0][0]
____________________________________________________________________________________________________
block3_conv2 (Convolution2D)     (None, 56, 56, 256)   590080      block3_conv1[0][0]
____________________________________________________________________________________________________
block3_conv3 (Convolution2D)     (None, 56, 56, 256)   590080      block3_conv2[0][0]
____________________________________________________________________________________________________
block3_pool (MaxPooling2D)       (None, 28, 28, 256)   0           block3_conv3[0][0]
____________________________________________________________________________________________________
block4_conv1 (Convolution2D)     (None, 28, 28, 512)   1180160     block3_pool[0][0]
____________________________________________________________________________________________________
block4_conv2 (Convolution2D)     (None, 28, 28, 512)   2359808     block4_conv1[0][0]
____________________________________________________________________________________________________
block4_conv3 (Convolution2D)     (None, 28, 28, 512)   2359808     block4_conv2[0][0]
____________________________________________________________________________________________________
block4_pool (MaxPooling2D)       (None, 14, 14, 512)   0           block4_conv3[0][0]
____________________________________________________________________________________________________
block5_conv1 (Convolution2D)     (None, 14, 14, 512)   2359808     block4_pool[0][0]
____________________________________________________________________________________________________
block5_conv2 (Convolution2D)     (None, 14, 14, 512)   2359808     block5_conv1[0][0]
____________________________________________________________________________________________________
block5_conv3 (Convolution2D)     (None, 14, 14, 512)   2359808     block5_conv2[0][0]
____________________________________________________________________________________________________
block5_pool (MaxPooling2D)       (None, 7, 7, 512)     0           block5_conv3[0][0]
____________________________________________________________________________________________________
flatten (Flatten)                (None, 25088)         0           block5_pool[0][0]
____________________________________________________________________________________________________
fc1 (Dense)                      (None, 4096)          102764544   flatten[0][0]
____________________________________________________________________________________________________
fc2 (Dense)                      (None, 4096)          16781312    fc1[0][0]
____________________________________________________________________________________________________
predictions (Dense)              (None, 1000)          4097000     fc2[0][0]
====================================================================================================
Total params: 138357544
____________________________________________________________________________________________________
```

Input->CONV3-64->CONV3-64->POOL2->CONV3-128->CONV3-128->POOL2->CONV3-256->CONV3-256->CONV3-256->POOL2->CONV3-512->CONV3-512->CONV3-512->POOL2->CONV3-512->CONV3-512->CONV3-512->POOL2->FC->FC->FC
Final three FC are of the following dimensions
FC[1*1*4096],FC[1*1*4096],FC[1*1*1000]

In figure 1 of the report we get a fully connected layer of 4096 features.

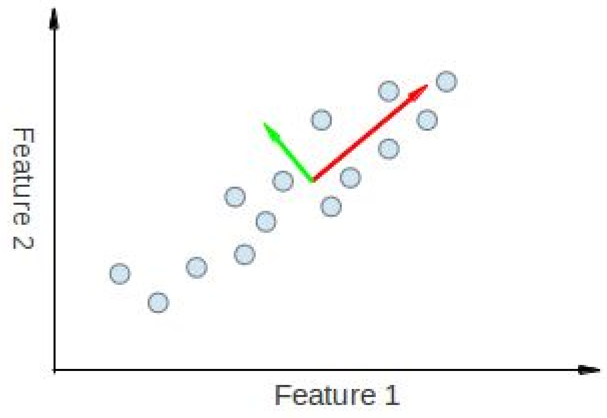
In the above VGGNet our feature extraction process is completed before reaching FC[1*1*1000](which represents the classification output of the input image).The last before layer i.e FC[1*1*4096] in VGGNet is final layer that we pass to the next step of our entire process that is laid out in the figure 1.This last before layer is called FC7 layer.By the time we reach this FC7 layer we have successfully completed our feature extraction.In this case(figure 1) it actually contains 4096 features.This FC7 layer features are reduced by applying PCA which we discuss in the following lines of the report.

Principal Component Analysis

In our case we finally got a fully connected layer which contains 4096 features after extracting features of the input image by using pre-trained CNN.We can directly pass this FC layer to LSTM instead of applying PCA.
But if we do so it will be computationally expensive.We need to reduce the number of features due to the limitations of the computational power.
In our case we reduced the feature vector from 4096*1 to 512*1.We achieve this by applying a technique called principal component analysis a.k.a PCA.So Let's discuss how this technique works.

PCA is a dimensionality reduction algorithm that can be used to significantly speed up our feature learning algorithm.

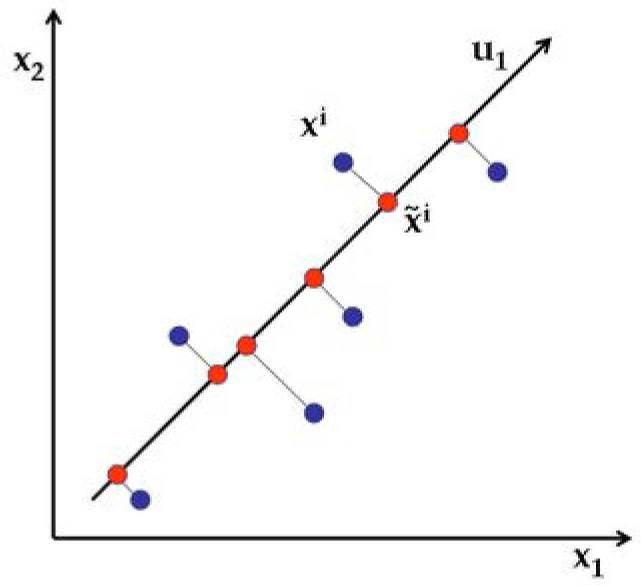
By simple figure we can explain what PCA technique does on the traing data.Let's take this example of training data defined by 2 features.Plotting the data will give the below figure.



We represented our input training data using feature1 and 2 by plotting as above.Now when we observe the training data we can replace the feature 1 and feature 2 vector with a single feature vector **along the RED line**.

Now we can take approximate values of the input data along the RED line

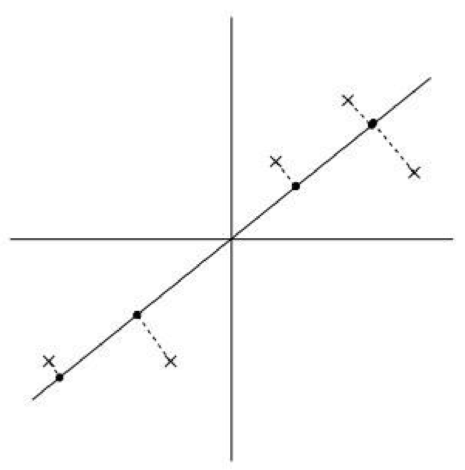
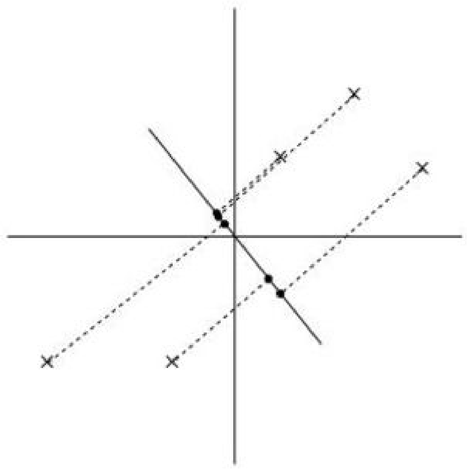
I.e only one feature vector.
We can replace the training data representation from two feature components x1 and x2 to a single feature component u1.
This is exactly what PCA does.Dimensionality reduction of the input feature vector.



<u>PCA Description</u>
Prior to running PCA , typically we first pre-process the data to normalize its mean and variance, as follows:
1. Replace each x (i) with x (i) − µ.
2. Replace each x (i) j with x (i) j / σ j .
where µ, σ j represent the mean and standard deviation of the input training examples.

After data pre-processing we need to find our new feature vectors such that projection of our original data onto the new feature vector retain maximum variance of our original data.
I.e projection error onto our new feature vectors is minimum.

For example let’s take two different vectors for the following data represented in the next page

In the first case the projection error onto the feature vector is much lower than the projection error in the second case.
Feature vector in the first case is more approximate and apt representation of our data.Because it our new data after projection in the first case more accurate to the original data than that of the second case.

This is how PCA principally works.We just showed basic training data.

But for our actual problem there are 4096 feature components.So to make the problem computationally efficient and cost expensive we reduce the dimensions of our 4096 featured data to 512 featured data using PCA.Since PCA finds out new feature components such the originality of the data is retained(more than 95%) we can use this 512 featured vector for the upcoming LSTM component in our process.

NOTE : PCA did not manipulate data.It just made sure that we use feature vectors that approximately represent the original data by retaining more than 95 or 99% originality of our actual input data.Moreover it reduced the number of features which speeds up our upcoming algorithms and makes the whole process computationally reliable and efficient.

Finally after reducing dimensions from 4096 to 512 using PCA we can successfully pass this 512 feature vector to LSTM which further formulates the description of the image using these extractions of the features from our original/input image and eventually outputs a natural sentence describing the contents of the image.

Before passing this 512 dimensional feature vector as input to LSTM to train the model to learn generating images let’s understand about what are word embeddings and how an LSTM network works.

Word Embeddings (word vectors or word2vec):

A word embedding *W* : words→R*n* is a paramaterized function mapping words in some language to high-dimensional vectors (perhaps 200 to 500 dimensions).

For example, we might find:

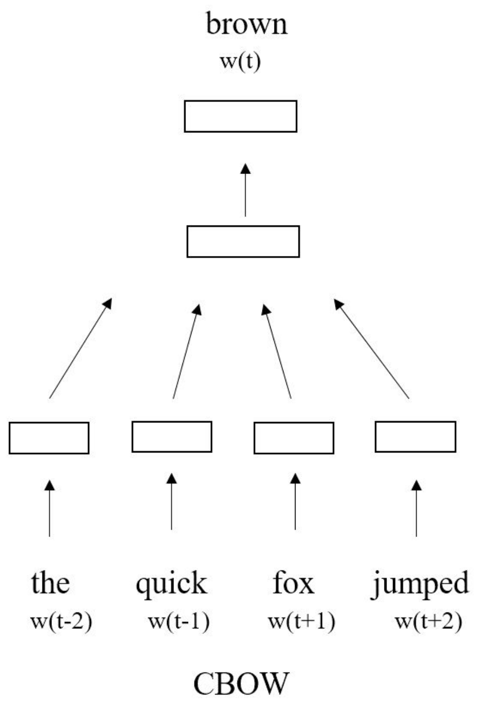
W(‘‘cat")=(0.2, -0.4, 0.7, ...) a 300 dimensional vector

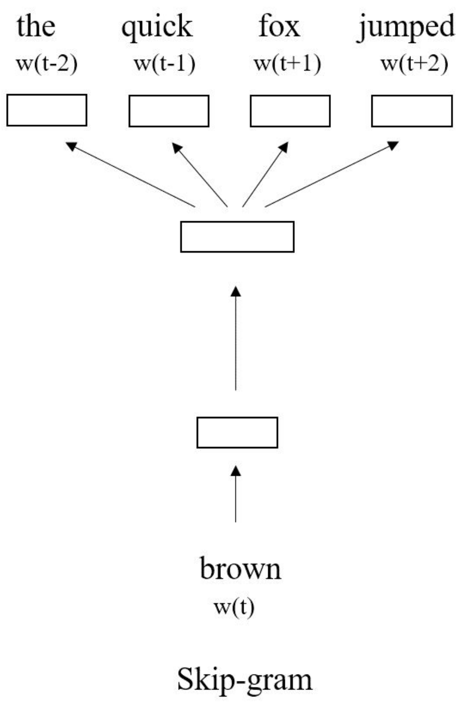
Now how we get these vector representations for words:

There are two main models which generate these word vectors:

CBOW model :
This approach is to treat { "the", "quick", "fox", "jumped" } as a context and from these words, be able to predict or generate the center word "brown". This type of model we call a Continuous Bag of Words (CBOW) Model.

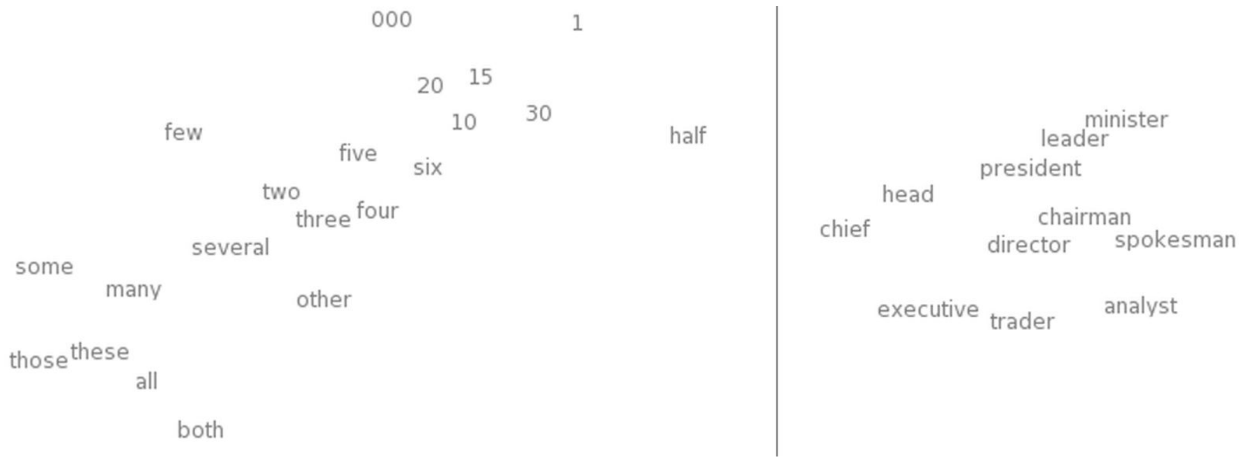
Skip-Gram Model :
Another approach is to create a model such that given the center word "jumped", the model will be able to predict or generate the surrounding words "the", "quick", "fox", "jumped" . Here we call the word "brown" the context. We call this type of model a Skip-Gram model.





Nearest Words :

After generating these word vectors for each word in the vocabulary one thing we can do to get a feel for the word embedding space is to visualize them with t-SNE, a sophisticated technique for visualizing high-dimensional data.

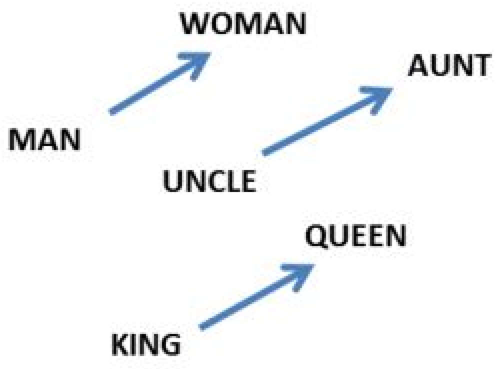


This kind of ‘map’ of words makes a lot of intuitive sense to us. Similar words are close together. Another way to get at this is to look at which words are closest in the embedding to a given word. Again, the words tend to be quite similar.

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Capturing relation between vectors:

Word Vectors exhibit an even more remarkable property: analogies between words seem to be encoded in the difference vectors between words. For example, there seems to be a constant male-female difference vector



$$W(\text{``woman"}) - W(\text{``man"}) \simeq W(\text{``aunt"}) - W(\text{``uncle"})$$

$$W(\text{``woman"}) - W(\text{``man"}) \simeq W(\text{``queen"}) - W(\text{``king"})$$

It turns out, though, that much more sophisticated relationships are also encoded in this way. It seems almost miraculous!

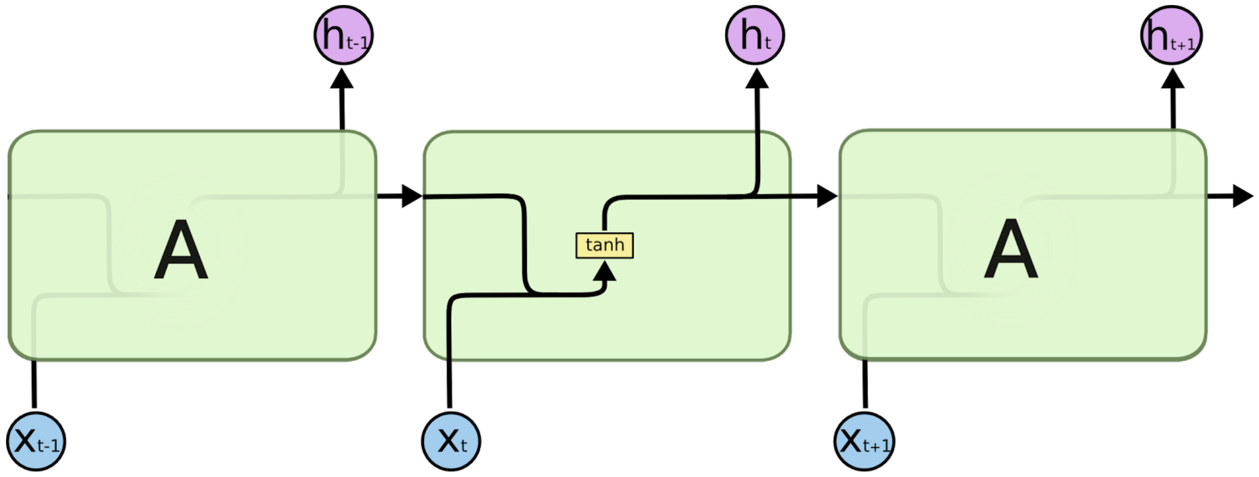
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

LSTM:

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.They work tremendously well on a large variety of problems, and are now widely used.

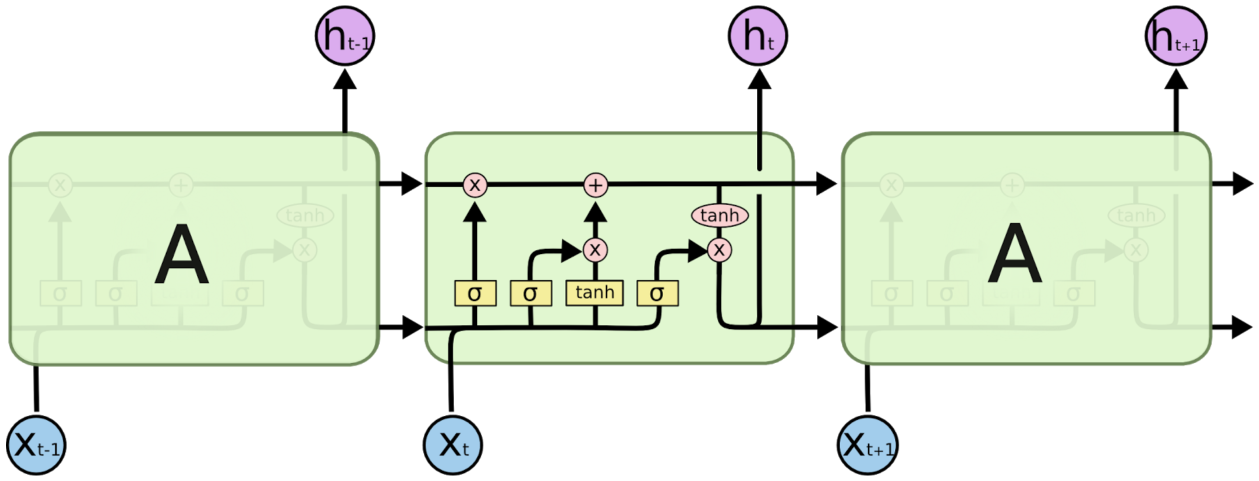
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



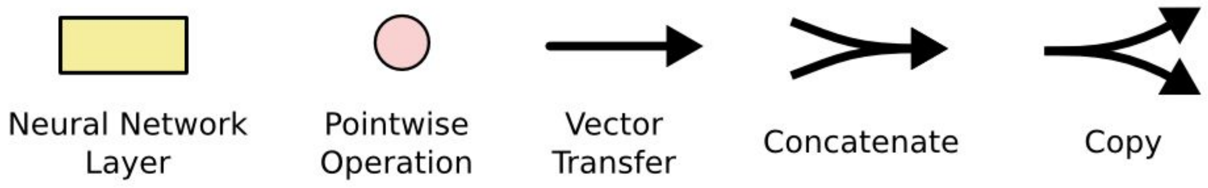
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.

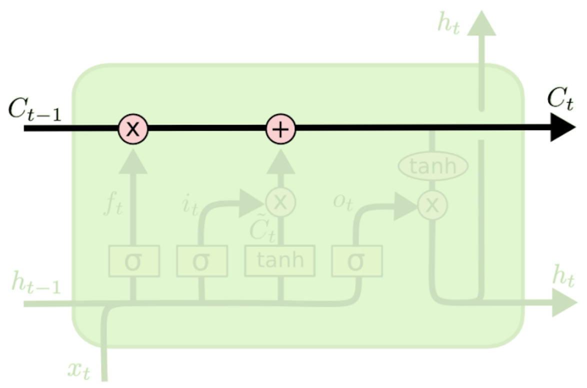


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

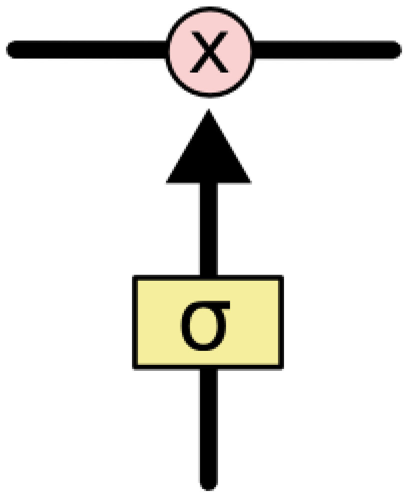
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



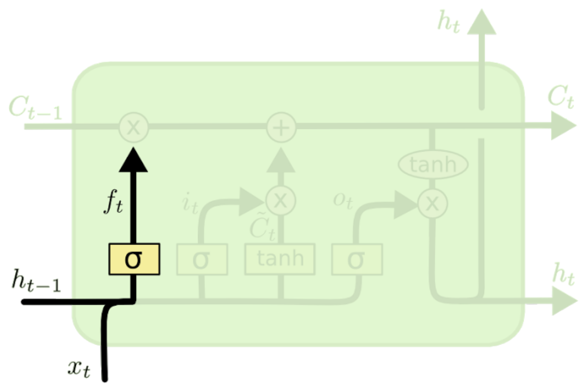
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at *ht*−1 and *xt* , and outputs a number between 0 and 1 for each number in the cell state *Ct*−1. A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

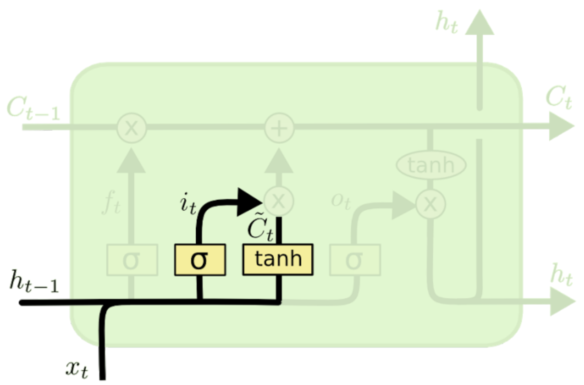
Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, *C~t* , that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

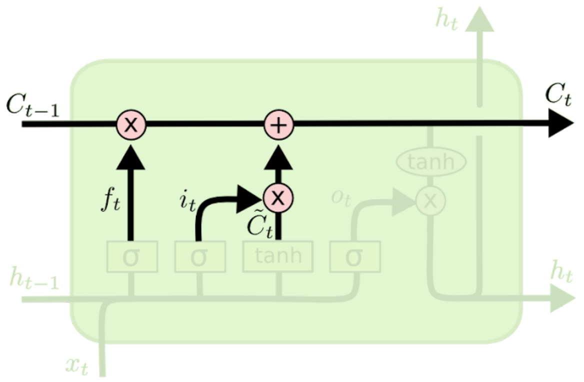


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, *Ct*−1 , into the new cell state *Ct* . The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by *ft*ft, forgetting the things we decided to forget earlier. Then we add *it*∗*C~t* . This is the new candidate values, scaled by how much we decided to update each state value.

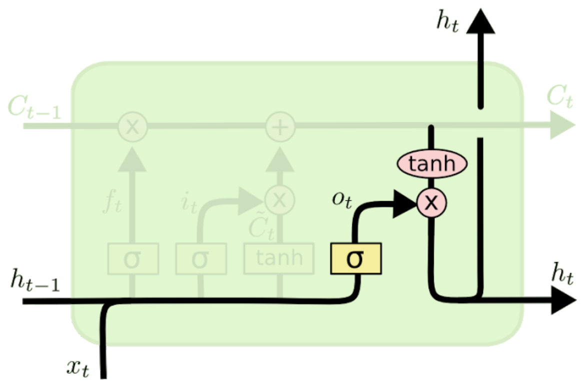
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Now that we are aware of all the basic parts of the model, let us see more about how we train and test the model :

Model Working:

Procedure:

Our LSTM model takes the image I and a sequence of inputs vectors (x_1, ..., x_T). It then computes a sequence of hidden states (h_1, ..., h_t) and a sequence of outputs (y_1, ..., y_t) by following the recurrence relation for t = 1 to T:

- $b_v = W_{hi}[CNN(I)]$
- $h_t = f(W_{hx}x_t + W_{hh}h_t−1 + b_h + 1(t = 1) \circ b_v)$
- $y_t = Softmax(W_{oh}h_t + b_o)$

Here W_{hi}, W_{hx}, W_{hh}, W_{oh}, x_i , b_h, and b_o are learnable parameters.

CNN(I) represents the image features extracted by the CNN. (i.e the 512 dimensional vector obtained after the PCA.)

Training:

- We train our LSTM model to correctly predict the next word (y_t) based on the current word (x_t), and the previous context (h_{t-1}).
- We do this as follows: we set h_0 = 0, x_1 to the START vector, and the desired label y_1 as the first word in the sequence. We then set x_2 to the word vector corresponding to the first word generated by the network.
- Based on this first word vector and the previous context the network then predicts the second word, etc. The word vectors are generated using the word2vec. During the last step, x_T represent the last word, and y_T is set to an END token.

Testing:

- To predict a sentence, we obtain the image features b_v, set $h_0 = 0$, set x_1 to the START vector, and compute the distribution over the first word y_1.
- Accordingly, we pick the argmax from the distribution, set its embedding vector as x_2, and repeat the procedure until the END token is generated.

Softmax Loss:

At every time-step, we generate a score for each word in the vocabulary. We then use the ground truth words in combination with the softmax function to compute the losses and gradients. We sum the losses over time and average them over the minibatch. Since we operate over minibatches and because different generated sentences may have different lengths, we append NULL tokens to the end of each caption so that they all have the same lengths. In addition, our loss function accepts a mask array that informs it on which elements of the scores counts towards the loss in order to prevent the NULL tokens to count towards the loss or gradient.

Optimization:

SGD(Stochastic Gradient Descent) or ADAM for minimizing the loss and getting the optimal parameters.

With all the optimal parameters of the model obtained, now we can use this to generate captions for any new images.

Datasets and Evaluation Metrics:

- Microsoft COCO dataset is a standard dataset for this task,
- BLEU (Bilingual Evaluation Understudy), METEOR (Metric for Evaluation of Translation with Explicit Ordering), and CIDEr (Consensus-based Image Description Evaluation) are some of the evaluation metrics that can be used for this task.

Applications of Image Caption Generator

- Making Technology accessible to visually impaired people
    - Smartphones made it possible for the visually impaired to take images of their surroundings. These images can be used to generate captions that can be read out loud to give visually impaired people a better understanding of their surroundings.
    - Also makes the web more accessible to visually impaired persons.

- Social Media platforms can automatically describe the images being uploaded

    - Captioning where you are,what you do etc..

Summary :

We have presented a deep learning model that automatically generates image captions. Our described model is based on a CNN that encodes an image into a compact representation, followed by a RNN that generates corresponding sentences based on the learned image features. The generated captions are highly descriptive of the objects and scenes depicted on the images. Because of the high quality of the generated image descriptions, visually impaired people can greatly benefit and get a better sense of their surroundings using text-to-speech technology. Future work can include this text-to-speech technology, so that the generated descriptions are automatically read out loud to visually impaired people. In addition, future work could focus on translating videos directly to sentences instead of generating captions of images. Static images can only provide blind people with information about one specific instant of time, while video caption generation could potentially provide blind people with continuous real time information. LSTMs could be used in combination with CNNs to translate videos to English descriptions

References

1. Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. Show and Tell: A Neural Image Caption Generator.
2. Keras Implementation (github repo)
3. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
4. Distributed Representations of Words and Phrases and their Compositionality (word2vec)
5. Convolutional Neural Networks for Computer Vision (cs231n).

Team Members :

Sanju Prabhath Reddy - 15114042
Sri Harsha Majeti - 15114044
Harsha Vardhan Miryala - 15114045