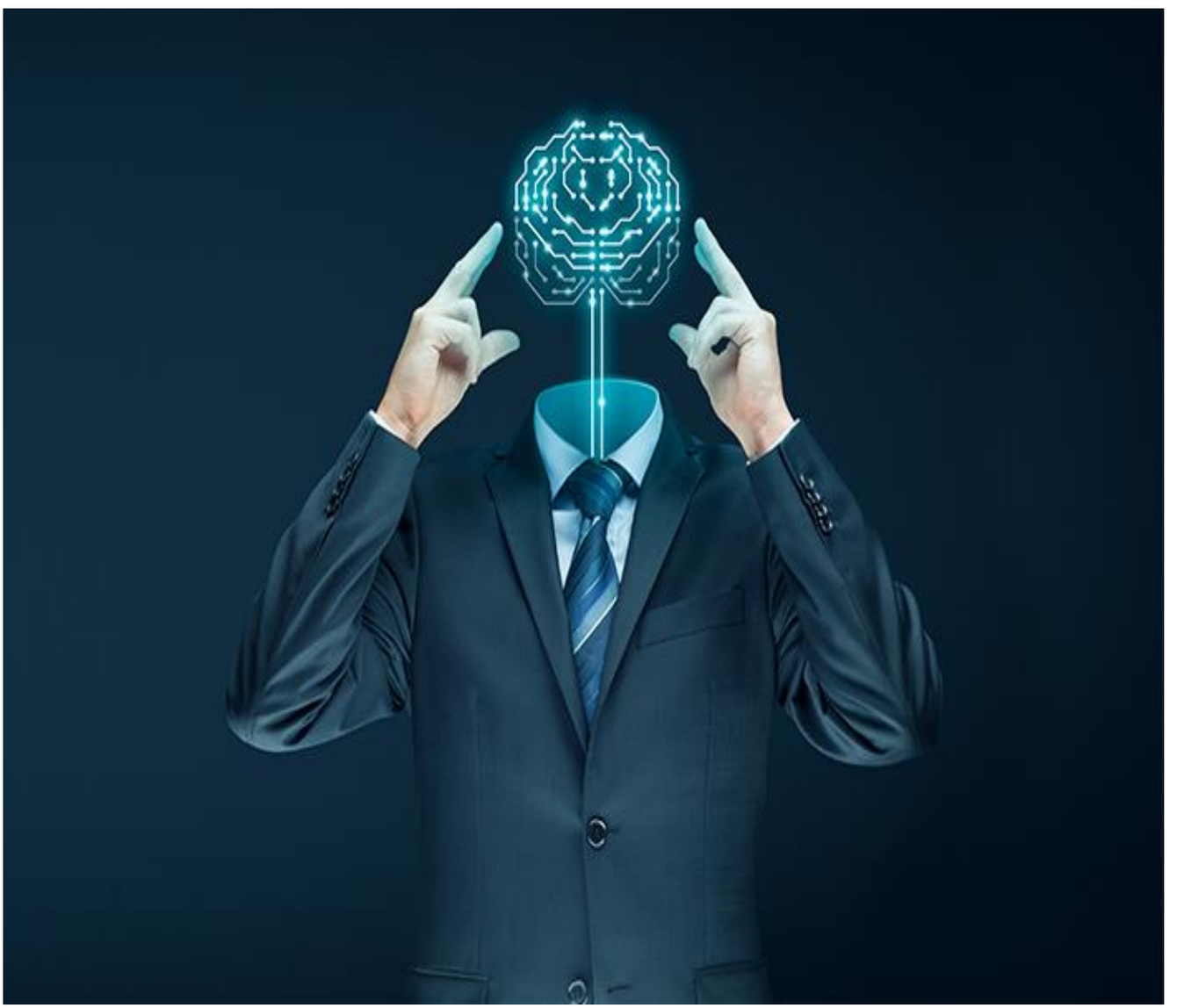
Summer of Science - Final Report

Auto Image Captioning

Gollavilli Srikanth
193310016
GNR, CSRE

June 05, 2020

Mentor : Anil Kumar



Final Report

Contents

1 Introduction

We can easily identify any image immediately after seeing it, but it is hard for the computer to do the same. Nowadays, deep learning has unveiled such difficulties and has facilitated us to build an application which can identify any image. The caption of the image is based on the huge database which will be fed to the system. This machine learning project of image caption generator is implemented with the help of python language. This project will need the techniques of convolution neural network and recurrent neural network.

So automatic Image captioning is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English. Image caption models can be divided into two main categories. A method based on a statistical probability language model to generate handcraft features and a neural network model based on an encoder-decoder language model to extract deep features.

1.1 Types

a) Handcraft Features with Statistical Language Model

This method is a Midge system based on maximum likelihood estimation, which directly learns the visual detector and language model from the image description dataset. First analyze the image, detect the object, and then generate a caption. Words are detected by applying a convolutional neural network (CNN) to the image area and integrating the information with multi-instance learning (MIL). The structure of the sentence is then trained directly from the caption to minimize the priori assumptions about the sentence structure. Finally, it turns an image caption generation problem into an optimization problem and searches for the most likely sentence.

The implementation steps are as follows:

- Detect a set of words that may be part of the image caption. We detect the words from the given vocabulary according to the content of the corresponding image based on the weak monitoring method in MIL in order to train the detectors iteratively.
- Running a fully convolutional network on an image, we get a rough spatial response graph. Each position in the response map corresponds to a response obtained by applying the original CNN to the region of the input image where the shift is shifted. By upsampling the image, we get a response map on the final fully connected layer and then implement the noisy-OR version of MIL on the response map for each image. Each word produces a single probability.
- The process of caption generation is searching for the most likely sentence under the condition of the visually detected word set. The language model is at the heart of this process because it defines the probability distribution of a sequence of words. Although the maximum entropy language model (ME) is a statistical model, it can encode very meaningful information.

- There are similar ways to use the combination of attribute detectors and language models to process image caption generation. Like a combination of CNN and k-NN methods and a combination of a maximum entropy model and RNN to process image description generation tasks. Kenneth Tran proposed an image description system, using CNN as a visual model to detect a wide range of visual concepts, landmarks, celebrities, and other entities into the language model and the output results are the same as those extracted by CNN. The vectors together are used as input to the multichannel depth similar model to generate a description.

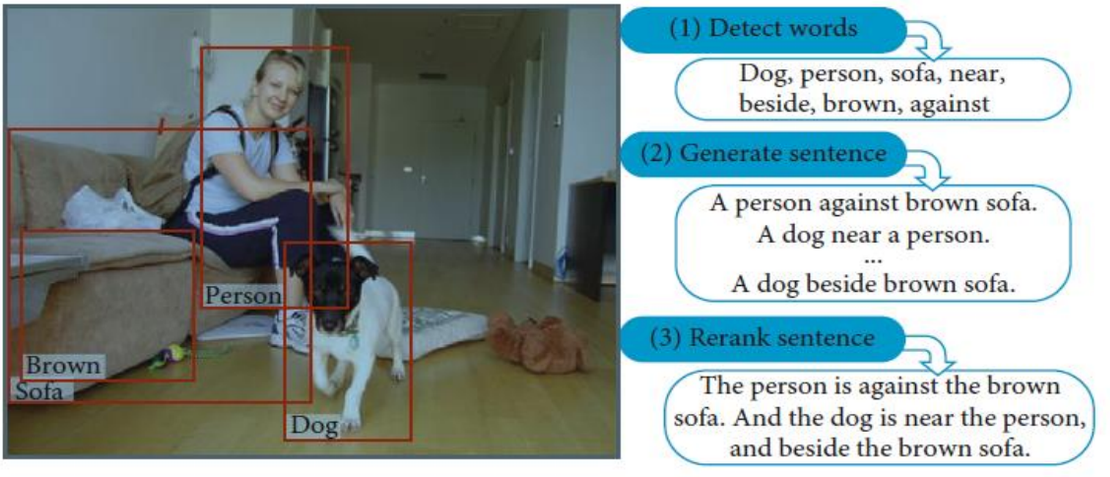


FIGURE 1: Method based on the visual detector and language model.

b) Deep Learning Features with Neural Network

The recurrent neural network (RNN) is used in the field of natural language processing and achieved good results in language modeling. In the field of speech, RNN converts text and speech to each other, machine translation, question and answer session and so on. Of course, they are also used as powerful language models at the level of characters and words. RNN is also rapidly gaining popularity in computer vision. The image description generation method based on the encoder-decoder model is proposed with the rise and widespread application of the recurrent neural network. In the model, the encoder is a convolutional neural network, and the features of the last fully connected layer or convolutional layer are extracted as features of the image. The decoder is a recurrent neural network, which is mainly used for image description generation. Because RNN training is difficult, and there is a general problem of gradient descent, although it can be slightly compensated by regularization, RNN still has a fatal flaw that it can only remember the contents of the previous limited time unit, and LSTM is a special RNN architecture that can solve problems such as gradient disappearance, and it has long-term memory. In recent years, the LSTM network has performed well in dealing with video related context. Similar with video context, the LSTM model structure is generally used in the text context decoding stage.

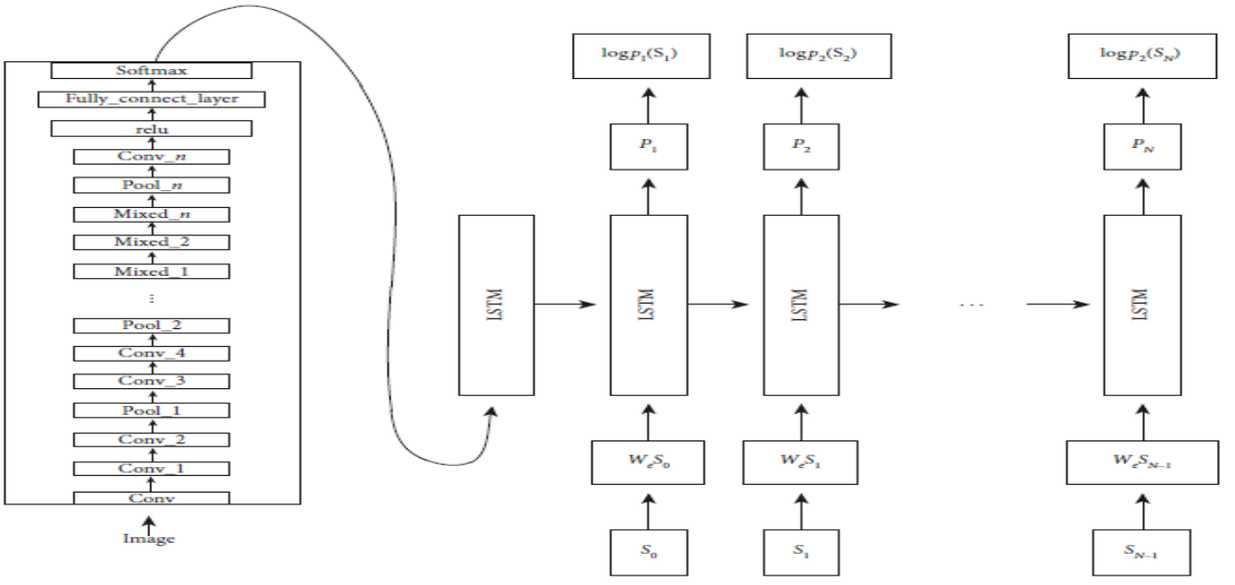


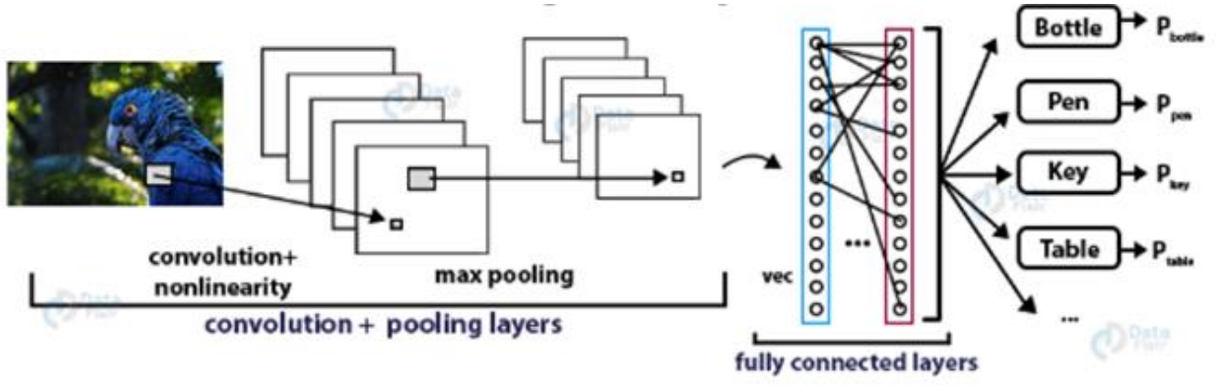
FIGURE 2: Model based on encoder-decoder.

2 What is CNN?

Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images. CNN is basically used for image classifications and identifying if an image is a bird, a plane or Superman, etc.

These are used particularly in the field of computer vision. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks developed for learning regular Neural Networks still apply. ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Working of Deep CNN



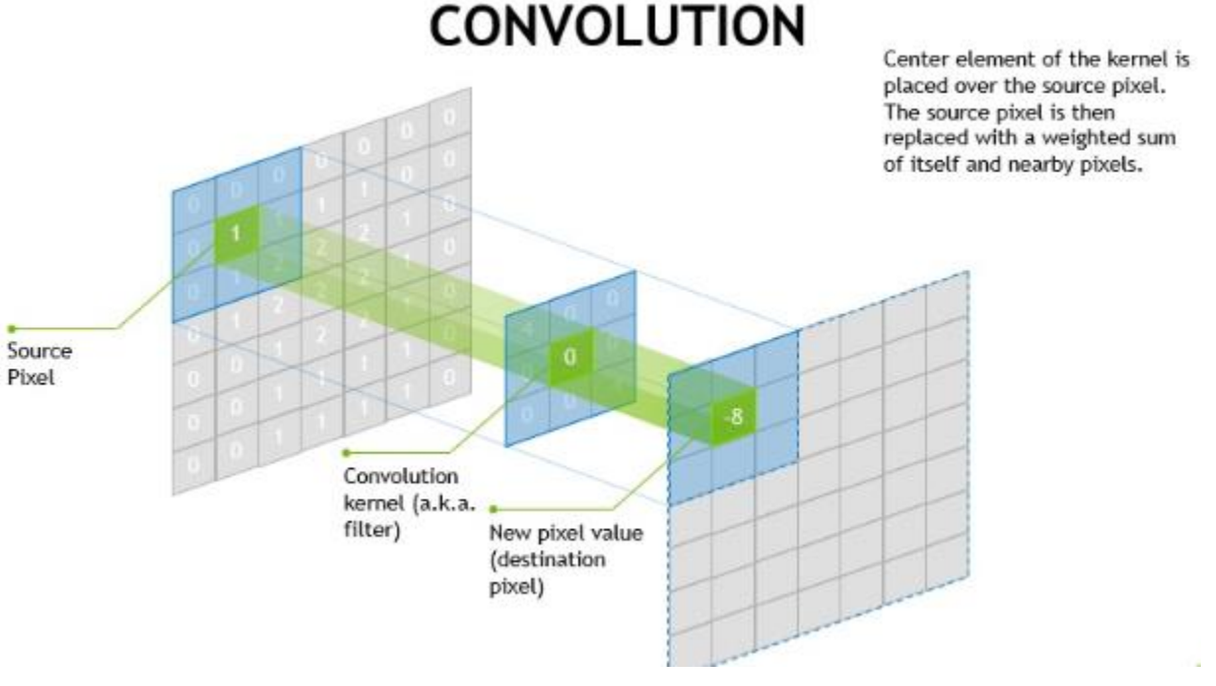
2.1 Architecture Overview

A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. We use three main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer as seen in regular Neural Networks.

2.1.1 Convolutional Layer

The convolutional layer is the key component of convolutional neural networks. Its purpose is to detect the presence of a set of features in the images received as input. This is done by convolution filtering: the principle is to "drag" a window representing the feature on the image, and to calculate the convolution product between the feature and each portion of the scanned image.

The convolutional layer thus receives several images as input, and calculates the convolution of each of them with each filter. The filters correspond exactly to the features we want to find in the images. An image and filter pair called as feature map is produced, which tells where the features are in the image. Higher the value, more the corresponding place in the image resembles the feature. Unlike traditional methods, features are not pre-defined according to a particular formalism but learned by the network during the training phase. Filter kernels refer to the convolution layer weights are initialized and then updated by back propagation using gradient descent.



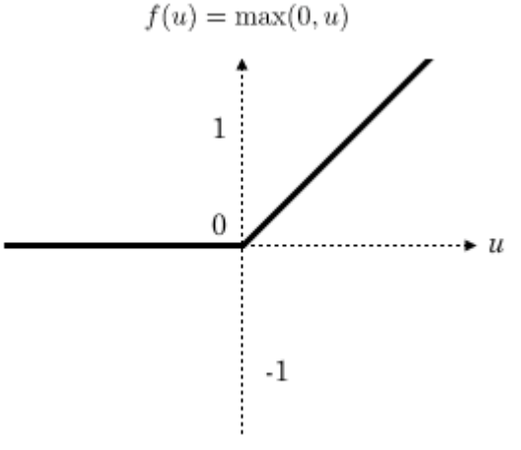
2.1.2 Pooling layer

This type of layer is often placed between two layers of convolution: it receives several feature maps and applies the pooling operation to each of them. The pooling operation consists in reducing the size of the images while preserving their important characteristics of input image.

This is done by cutting the image into regular cells then the maximum value is kept within each cell. In practice, small square cells are often used to avoid losing too much information. The most common choices are 2x2 adjacent cells that don't overlap, or 3x3 cells, separated from each other by a step of 2 pixels. We get in output the same number of feature maps as input, but these are much smaller. The pooling layer reduces the number of parameters and calculations in the network. This improves the efficiency of the network and avoids over-learning. The maximum values are spotted less accurately in the feature maps obtained after pooling than in those received in input which is a big advantage. So, when you want to recognize a dog, its ears do not need to be located as precisely as possible, as knowing that they are located almost next to the head is enough.

2.1.3 ReLU correction layer

ReLU (Rectified Linear Units) refers to the real non-linear function defined by ***ReLU(x)=max(0,x)***. Visually, it looks like the following:



The ReLU correction layer replaces all negative values received as inputs by zeros. It acts as an **activation function**.

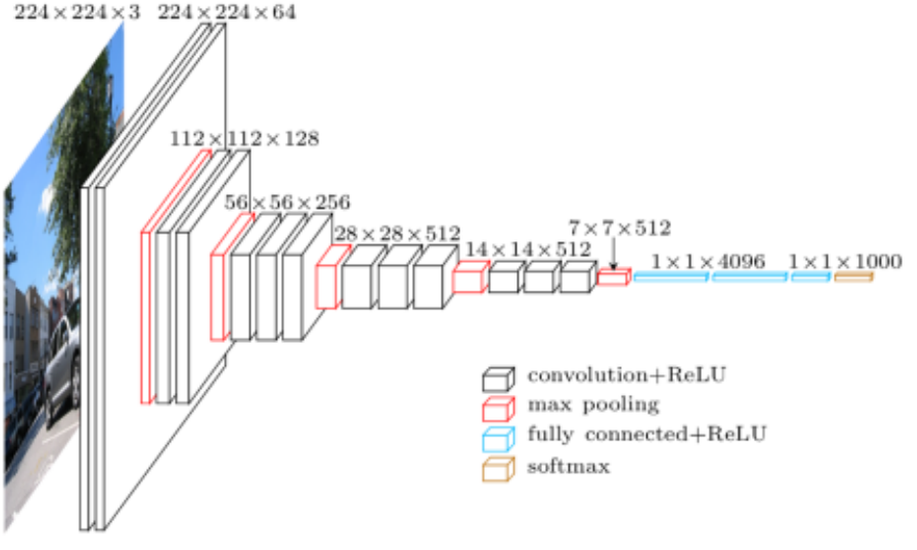
2.1.4 Fully-connected layer

The fully-connected layer is always the last layer of a neural network, thus it is not characteristic of a CNN. This type of layer receives an input vector and produces a new output vector. To do this, it applies a linear combination and then possibly an activation function to the input values received.

The last fully-connected layer classifies the image as an input to the network: it returns a vector of size N, where N is the number of classes in our image classification problem. Each element of the vector indicates the probability for the input image to belong to a class. To calculate the probabilities, the fully-connected layer multiplies each input element by weight, makes the sum, and then applies an activation function. This is equivalent to multiplying the input vector by the matrix containing the weights. The fact that each input value is connected with all output values explains the term fully-connected.

The convolutional neural network learns weight values in the same way as it learns the convolution layer filters: during the training phase, by back propagation of the gradient. The fully connected layer determines the relationship between the position of features in the image and a class. Indeed, the input table being the result of the previous layer, it corresponds to a feature map for a given feature: **the high values indicate the location of this feature in the image**. If the location of a feature at a certain point in the image is characteristic of a certain class, then the corresponding value in the table is given significant weight.

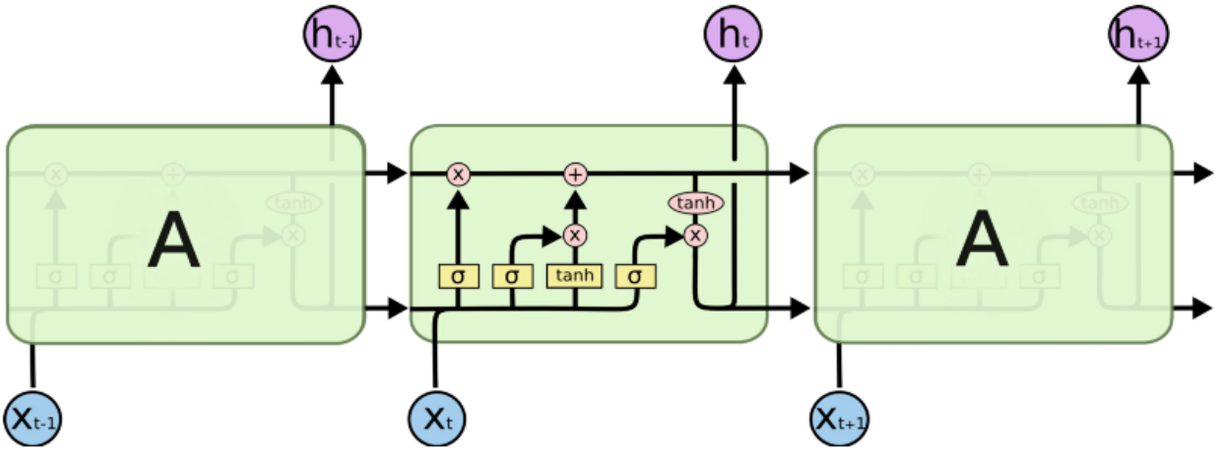
Typical CNN architecture



3 What is LSTM?

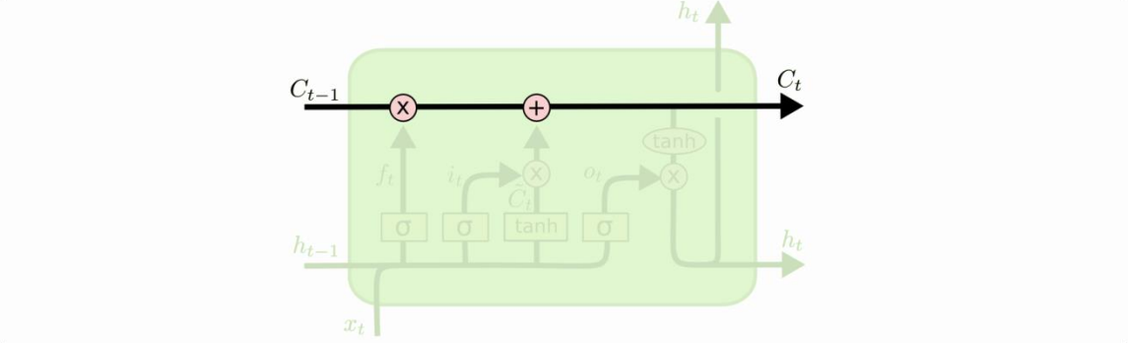
LSTM stands for **Long short term memory**, they are a type of RNN (**recurrent neural network**) which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information.

An LSTM consists of three main components: a forget gate, input gate, and output gate. Each of these gates is responsible for altering updates to the cell's memory state.

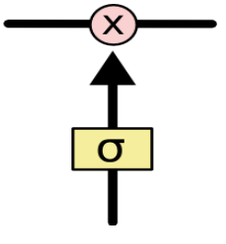


3.1 Working of LSTMs

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 point wise multiplication operation.

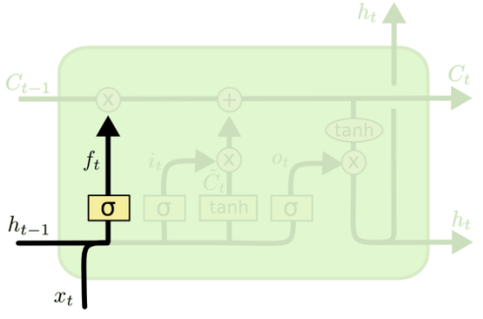


The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through". An LSTM has three of these gates, to protect and control the cell state.

3.2 LSTM Walk Through

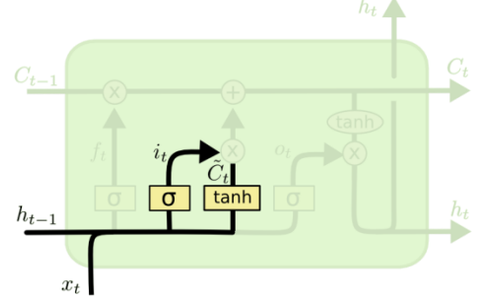
The first step in LSTM is to decide what information going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

In the problem of language mode, prediction of the next word is 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 new subject is seen, it needs to forget the gender of the old subject.



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

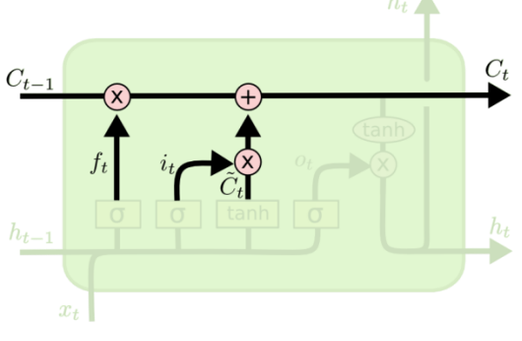
The next step is to decide what new information going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, (bar)C_t that could be added to the state. In the language model, to add the gender of the new subject to the cell state, replace the old one.



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

Next, the old cell state, C_{t-1} into the new cell state C_t. Multiply the old state by f_t, forgetting the things, decided to forget earlier. Then we add i_t*(bar)C_t. This is the new candidate values, scaled by as much decided to update each state value.

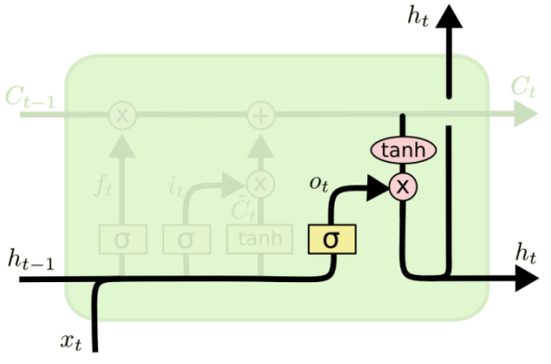
In the case of the language model, this is where the information about the old subject's gender is dropped and the new information is added, as decided in the previous steps.



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

Finally, decide the output value. This output will be based on our cell state, but will be a filtered version. First, a sigmoid layer is run which decides what parts of the cell state going to output. Then, putting the cell state through tanh and multiply it by the output of the sigmoid gate, so that only output the parts as decided.

For the language model example, as a subject is seen earlier, output information relevant to a verb is expected. For example, output might be whether the subject is singular or plural, so that form of the verb should be conjugated with what follows next.

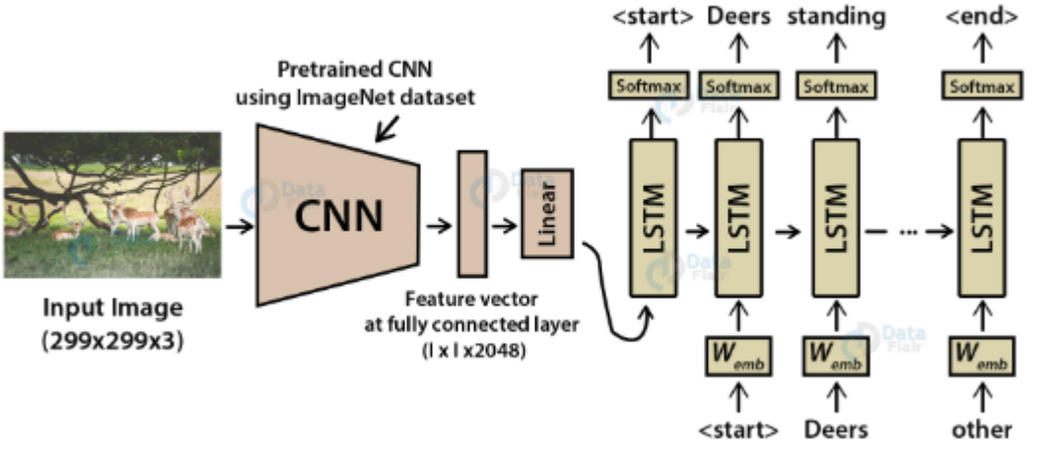


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

4 Image Caption Generator Model

To make our image caption generator model, we will be merging these architectures. It is also called a CNN-RNN model.

- CNN is used for extracting features from the image. The feature extraction model is a neural network that given an image is able to extract the salient features, often in the form of a fixed-length vector. A convolutional neural network is best used as the feature extraction submodel. This network can be trained directly on the images in dataset.
- LSTM will use the information from CNN to help generate a description of the image.



5 Implementation

5.1 Resources

- **Dataset :** **Flickr8k_Dataset**(Contains 8092 photographs in JPEG format) and **Flickr8k_text**(Contains a number of files containing different sources of descriptions for the photographs).
- **Keras** is an open-source neural-network library written in Python.
- **Google Colab** is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

5.2 Code:

- **Load Dataset and Model**

```
#Import Necessary Libraries
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model

# extract features from each photo in the directory
def extract_features(directory):
        # load the model
        model = VGG16()
        # re-structure the model
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
        # summarize
        print(model.summary())
        # extract features from each photo
        features = dict()
        for name in listdir(directory):
                # load an image from file
                filename = directory + '/' + name
                image = load_img(filename, target_size=(224, 224))
                # convert the image pixels to a numpy array
                image = img_to_array(image)
                # reshape data for the model
                image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
                # prepare the image for the VGG model
                image = preprocess_input(image)
                # get features
                feature = model.predict(image, verbose=0)
                # get image id
                image_id = name.split('.')[0]
                # store feature
                features[image_id] = feature
                print('>%s' % name)
        return features

# extract features from all images
directory = 'Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
```

- **Defining Captioning Model**

```
def define_model(vocab_size, max_length):
        # feature extractor model
        inputs1 = Input(shape=(4096,))
        fe1 = Dropout(0.5)(inputs1)
        fe2 = Dense(256, activation='relu')(fe1)
        # sequence model
        inputs2 = Input(shape=(max_length,))
        se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
        se2 = Dropout(0.5)(se1)
        se3 = LSTM(256)(se2)
        # decoder model
        decoder1 = add([fe2, se3])
        decoder2 = Dense(256, activation='relu')(decoder1)
        outputs = Dense(vocab_size, activation='softmax')(decoder2)
        # tie it together [image, seq] [word]
        model = Model(inputs=[inputs1, inputs2], outputs=outputs)
        model.compile(loss='categorical_crossentropy', optimizer='adam')
        # summarize model
        print(model.summary())
        return model
```

- **Train the Model**

```
# train the model
model = define_model(vocab_size, max_length)
# train the model, run epochs manually and save after each epoch
epochs = 20
steps = len(train_descriptions)
for i in range(epochs):
        # create the data generator
        generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
        # fit for one epoch
        model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
        # save model
        model.save('model_' + str(i) + '.h5')
```

- **Evaluate Model through BLEU score**

```
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
        actual, predicted = list(), list()
        # step over the whole set
        for key, desc_list in descriptions.items():
                # generate description
                yhat = generate_desc(model, tokenizer, photos[key], max_length)
                # store actual and predicted
                references = [d.split() for d in desc_list]
                actual.append(references)
                predicted.append(yhat.split())
        # calculate BLEU score
        print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
        print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
        print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
        print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

```
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
BLEU-1: 0.529691
BLEU-2: 0.277107
BLEU-3: 0.182869
BLEU-4: 0.080318
```

- **Auto Generate Image Caption**

```
from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.models import load_model

# extract features from each photo in the directory
def extract_features(filename):
        # load the model
        model = VGG16()
        # re-structure the model
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
        # load the photo
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        return feature

# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from training)
max_length = 34
# load the model
model = load_model('model_18.h5')
# load and prepare the photograph
photo = extract_features('Sample_Image.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

6 Results

Input:

Output:

```
photo = extract_features('Sample_Image.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

```
man rides bike on dirt path
```

7 Conclusion

In this implementation, I used a pre trained model as a feature extractor in an encoder trained on the Flickr8k dataset as part of a deep learning solution. This solution combines techniques in both **computer vision** and **natural language processing**, to form a complete image description approach, able to construct computer-generated natural descriptions of any provided images. Attempted to break the barrier between images and language with this trained model and provided a technology that could be used as part of an application, helping the visually impaired enjoy the benefits of the megatrend of photo sharing.

8 References

8.1 Books and Nodes

- **GNR602** Class notes
- **Deep Learning: A Practitioner's ApproachBook** by Adam Gibson and Josh Patterson
- J. Song, X. Li, L. Gao, and H. Shen, "Hierarchical LSTMs with adaptive attention for visual captioning," 2018, http://arxiv.org/abs/1812.11004.
- K. Xu, J. Ba, K. Ryan et al., "Show, attend and tell: neural image caption generation with visual attention," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2048–2057, Boston, MA, USA, June 2015.

8.2 Links

- Andrew Ng's machine learning course in Coursera
- Tensorflow tutorial by Sentdex in YouTube
- https://www.freecodecamp.org/news/building-an-image-caption-generator-with-deep-learning-in-tensorflow-a142722e9b1f/
- https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/