IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 24, Issue 3, Ser. II (May. –June. 2022), PP 13-24
www.iosrjournals.org

A Deep Learning Model for Visually Impaired Person: Blind Sight

Rishi Mishra[1] & Bramah Hazela[1]
rishi.mishra1.777@gmail.com;bramahhazela77@gmail.com
[1] *Amity University, Lucknow UP, India*

Abstract
This paper is about creating a model, which can be used to extract data from image that is it is able to perform image captioning and then how to host it as a web application by integrating this model with the flask framework. I used Kaggle to create, train and test my model because the dataset which I used is of approximately 1 Gigabyte so it is not possible for my machine to train this model so for that I was required to use a cloud based platform which provides free of charge computing resources including GPU's like Kaggle, Google Colab. I have used flickr8k dataset for this model, it has 8000 images and each image is associated with five captions, which explains what is happening in the image. Kaggle and VS code helped me a lot to increase my productivity. I tried to create a deep learning model that can extract text from any image and render it on the screen. Image and caption related to that image will be given to us in the form of (x,y) pairs and can be extracted from Flickr8k dataset which I used to build this project, here my goal is to learn some mapping between x and y that is image and their respective captions y=f(x). The basic requirements for creating this model include MLP (Multilayer Perceptron Architecture), Convolution Neural Network, Recurrent Neural Network, Word Embedding's, Transfer Learning.

Keywords: *Neuron, Perceptron, CNN, MLP, RNN, LSTM, Transfer Learning*

Date of Submission: 29-05-2022 Date of Acceptance: 10-06-2022

I. Deep Learning

1.1 Artificial Neural Network

Human brain consists of millions of neurons. Neuron is nothing but a basic smallest and undividable unit of human brain. Neurons sends the input to the human brain by taking it from the surroundings via sensory organs. Brain or the neural network takes that input, processes that input, interprets it and generates the appropriate output to take the appropriate action at a given instance. When this same functionality we try to achieve artificially then that falls under artificial neural network. Artificial neuron is the replica of biological neuron. Artificial neuron has two main components: summation and activation function.

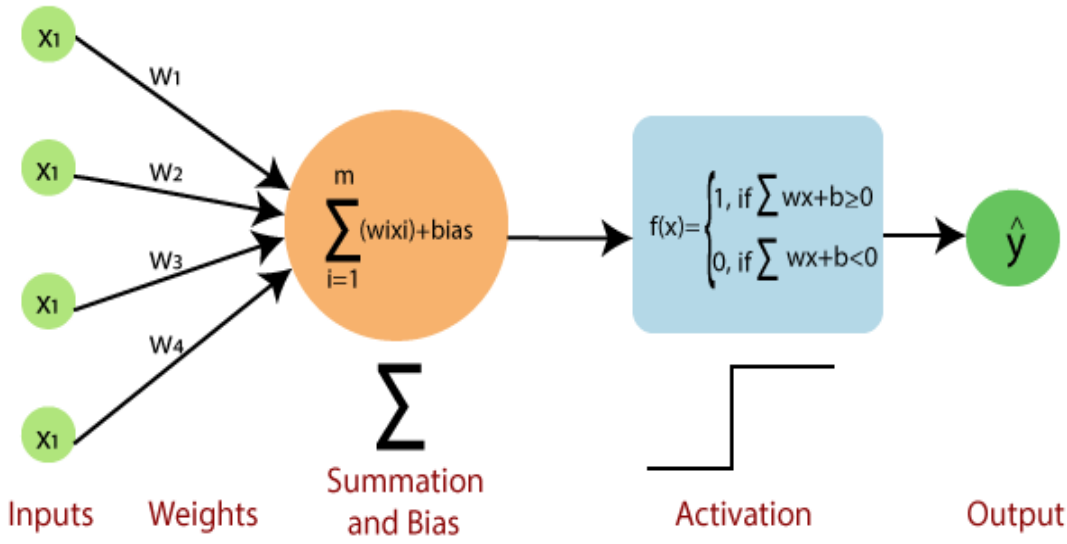
Let us understand with the help of analogy when a human touches the hot object then using the sensory receptors of skin this signal is transmitted to the brain and brain collects all these signals which are coming from different neurons and if brain finds that the all the signals when combined together gives a value greater than the threshold value then it responds by taking the action of removing the hand from that hot object. Similarly, a node or an artificial neuron is taking the input from various other node's output and associate that input with a weight and then the summation inside the node calculate the weighted sum ($w_1*x_1 + w_2*x_2 + ... + w_n*x_n$) and gives that weighted sum to the activation function which generates the particular output for a given node based on the input which is being provided to it.

Similarity between artificial neuron and biological neuron

BIOLOGICAL NEURON	**ARTIFICIAL NEURON**
Dendrites	Input
Synaptic gap	Weight is associated to each input
Axon	Output
Soma	Summation and Activation Function

1.2 Perceptron

Perceptron is very small neural network containing only one unit, which is called as single layer neural network. Perceptron is used for classification of linear networks that is when data is linearly separable then we use perceptron (meaning we can draw at least one line between the two classes, which separates the two classes, which we are trying to identify). This network will contain some inputs, weights and a constant called as bias.[2]



Input is represented by x1 and different weights are associated with each input. Summation is responsible for summing together all the weighted inputs with one constant called as bias then this weighted sum is given to activation function which categorizes the input into any one of the two different classes so f(x) represents the equation of a line if by putting the values of input into equation of line we get either positive value(or zero) or negative value so on the basis of the sign of the value which we obtain we classify whether the inputs belong to first class or the second class.[1]

1.3 MLP

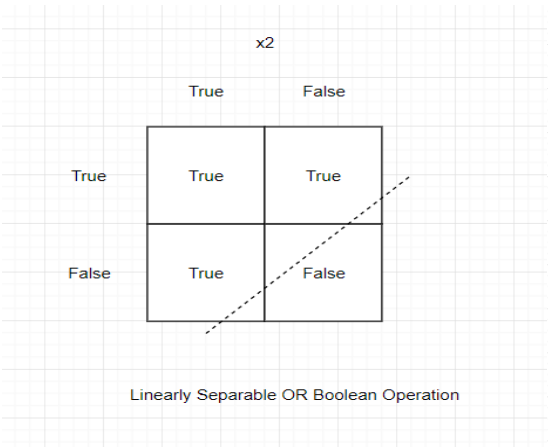
A multilayer perceptron is a perceptron with multiple layers. Each neuron has its own activation function. It consists of three parts namely input layer, hidden layers and output layer.

1. Input Layer:
- It is responsible for introducing input values into the network.
- It does not contain any activation function or processing unit.

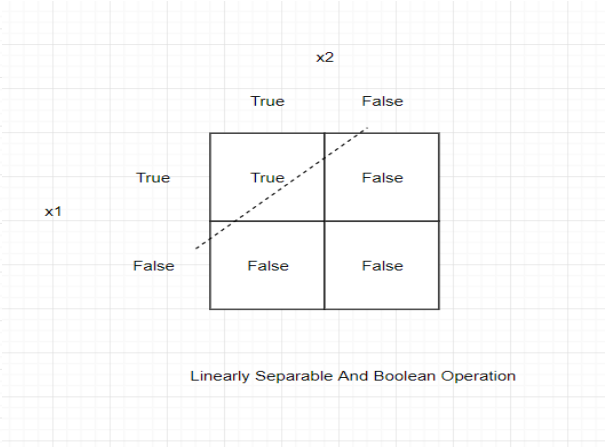
2. Hidden Layer
- Hidden layers performs classification of features. After getting the desired features, we perform mathematical operations on it like multiplying the features with their associated weights and find the summation of all inputs, etc.
- Two hidden layers are sufficient to solve almost any problem.

3. Output Layer
- The only difference between the hidden layers and the output layer is that hidden layer is hidden from outside world but functionally both hidden and output layers are same.

More complex problems that are not linearly can be solved using multi-layered perceptron. For example if we consider the case for Boolean operations like AND (&&) or OR (||) and draw the truth table for them then we get to know about that these are linearly separable problems.
In the figures we can see that, we have to use only one line in order to separate the cluster of true and false for Boolean Operation AND and OR.

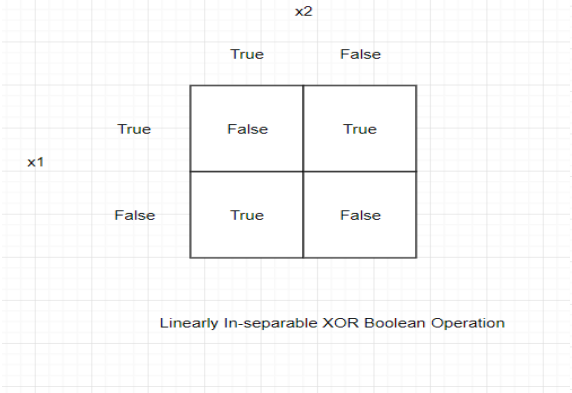


Linearly Separable OR Boolean Operation

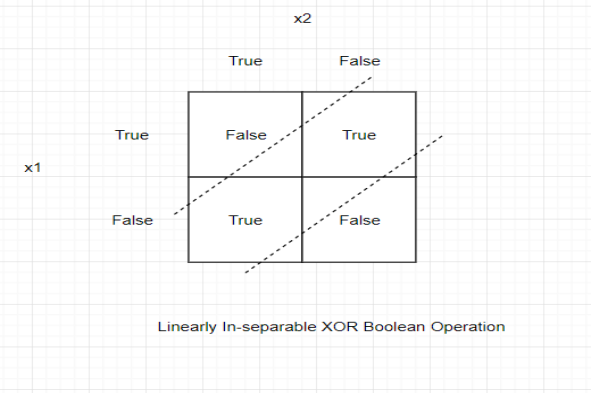


Linearly Separable And Boolean Operation

But in this figure as we can see that there we have literally no option to divide cluster of true and false by using only one line, at least two line are required to divide the clusters of true and false for Boolean operation XOR.



Linearly In-separable XOR Boolean Operation

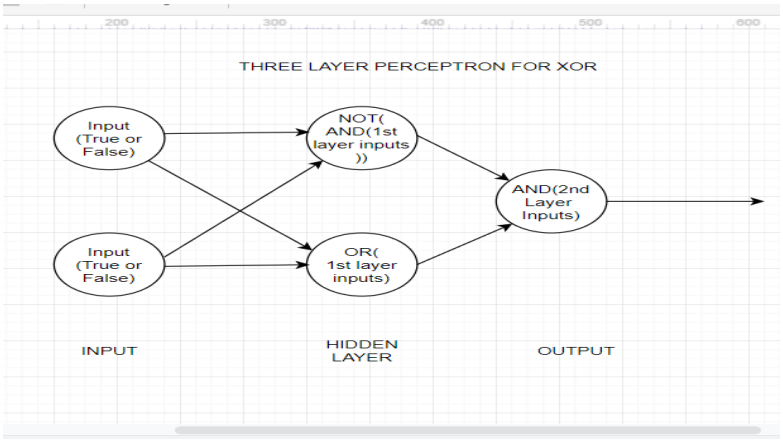


Linearly In-separable XOR Boolean Operation

So, using single layer perceptron it is not possible to even solve the simple XOR operation that's why multi-layer perceptron came into existence. Since we know that

$$a \wedge b = a'b' + ab = ((a.b)' \,.\, (a+b))' \quad (1)$$

Therefore, using three layer perceptron we can easily create that (analogy in this case it is same as creating a digital circuit using AND and OR gates to turn on the LED for XOR operation)[2]



1.4 CNN

CNNs (Convolutional Neural Network) are powerful deep neural networks that are widely used in image related tasks like Image Recognition, Segmentation, Computer Vision, etc. Generally, input to these networks are images.

The Need of CNN:

Problem with Multi-layer perceptron

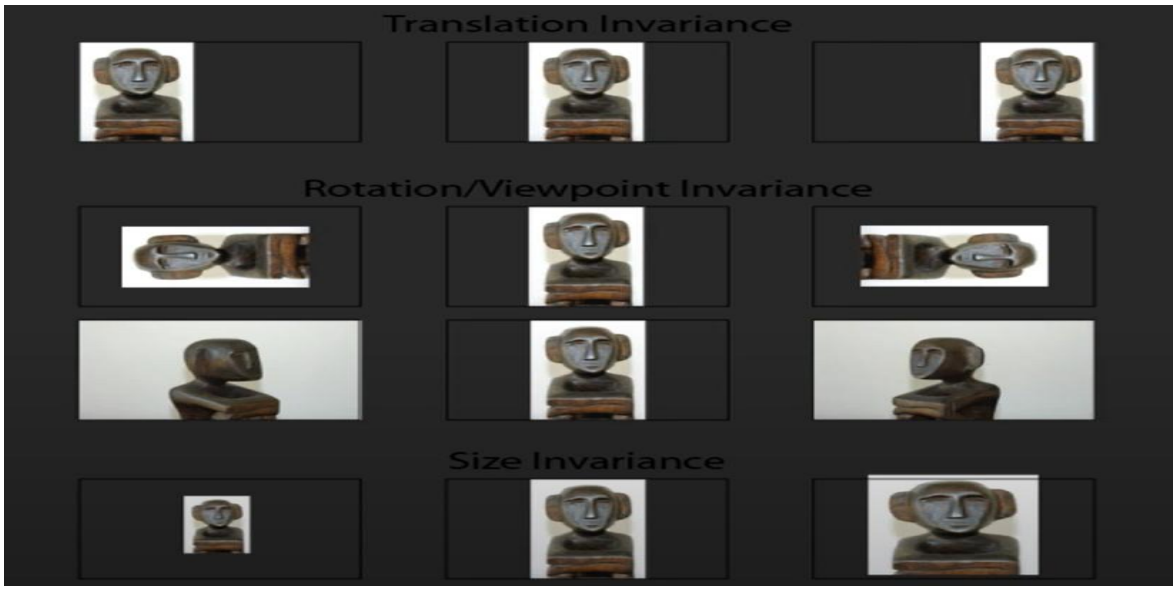
- Overfitting occurs due to too many parameters (~millions), while working with medium-large sized images which means that our model performs exceptionally well with the training data but in case of new data it produces very poor result, this happens because our model tries to cover all the data points present in the dataset, Because of this, the model starts caching noise and inaccurate values and this multitude of variables lessen the effectiveness and exactness of the model.

For example suppose we have an image of 100*100 pixels so it results in making almost 10,000 features and assuming that we have 200 neurons in our first layer then this will result in making the weight matrix of shape [200*10,000] = [20,00,000] ~ which approximates to million parameters and if we suppose that we have only 1000 images and that many parameters describing our model then our MLP model will over-fit.

- MLP is not very good at handling variance in the images like translation, rotation, illumination, size, etc.

For example: If we give an image to MLP and say that it is a statue then MLP learns that following pixels in the given region are activated which means that it is a statue. After training, the MLP model if we again give the image of statue to our MLP model then if it founds that in the similar region pixels are activated then it predicts that the image is of statue. But if we give the translated image of statue (that is it is somewhat shifted in x or y direction) then due to the translation, the pixels present in different region gets activated when we flatten the image into a linear layer which results in activating different set of neurons and in that case our MLP model will fail to predict that it is an image of statue because it treats every single pixel value as a feature but in reality this is not the case.

These are the various reason why MLP does not work very well in case of images because they can only give the good results on similar images, however if we give the image of same object to the MLP which may be taken from different angle, may be rotated or translated or may be of different size then in those case MLP will not be able to detect that the image corresponds to same object.



Intuition behind CNN:

In CNN, we do not consider all the pixels of image as relevant. The pixels or features that helps us to classify whether the image contains the specific object or not are concentrated in a local region. Therefore, instead of flattening all the pixels or features and then giving it to the neural model we can give the specific region of our image to the neural model, which is more helpful/useful in predicting that the image is of the specific object. The step of getting an activation map by applying the filter over an entire image is called as convolution.

Intuition is that if we take one input image and one filter/kernel/template where filter represents the image, which we want to find in our original image and try to match the filter with whole of the image by sliding the filter. If we find something similar to filter in our original image then we say that this image belongs to the specific class with which the filter is associated to, if not then our original image does not contains that specific object.

If the particular region of the original image is more similar to our filter then it will result in yielding the higher value in the activation map and if it is less similar to the filter, kernel, or template then it will result in yielding the lower value in the activation map. This means that when the feature is present in the part of an image, the convolution operation between the filter and that part of the image results in a real number with the high value. If the feature is not present, then the resulting value is low.

For Example detecting vertical edge in the greyscale image. So, for this an input greyscale image and a vertical edge filter is required. Assuming that we have an image of 4*4 pixel and a vertical edge filter of 3*3 pixel.

VERTICAL EDGE FILTER (3*3)

1	0	-1
2	0	-2
1	0	-1

IMAGE (6*6)

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

ACTIVATION MAP (n-f+1=6-3+1=2=>4*4)

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

Now after getting this activation map we can again retransform this image by using min-max scaler. In the activation map we find out what is the min value (-4) and max value (0) and change the min value to minimum value in terms of image (which is 0 because minimum number of pixels in an image is 0) and max value to maximum value in terms of image (which is 255 because maximum number of pixels in an image is

255). The transformed image we get is:

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

Now in the transformed image we see that the middle layer (2 and 3 column) is completely our white side and first and last column represents the dark side. The white layer is specifically determining the details of 3 and 4 column of our original image. Hence, with the help of convolution operation and vertical edge filter we are able to determine whether our image contains the vertical edge or not. Since in our original image vertical edge is present so, it determined that.[2]

1.5 Transfer Learning

In the real life scenarios, humans learn from experiences or from things that they have not done personally but they have observed them or someone told them that. For example if you have a car and if you do rash driving you know that can it may happen you met with an accident but it is possible that this knowledge may not be gained by you with experience instead you know it because you saw a person who is doing rash driving got crashed or your elders told you that or your read it from newspaper. So, this knowledge is transferred to you from various sources that is called as transfer learning.

Similarly, if in computer vision we have a very small dataset assuming it contains only 2000 images and we want to train very complex convolutional neural network to classify images, which has some 50 million parameters. So, this kind of a network is easily going to over-fit may be in 1-2 epochs and not produce good generalization the reason being is 2000 images (less data) are not sufficient to train complex neural network and because of less data the network will not be able to learn features well. This is where transfer learning comes into picture. The idea in transfer learning is to use pre-trained networks like Alexnet, VGG, Resnet50, Resnet101, Inception, Resnet-Inception, MobileNets, etc. Since the networks that are built by training the models are very large so we cannot train them from scratch but we can use transfer learning to train our model on our dataset by using feature extraction or fine-tuning.

VGG and MobileNet has almost similar accuracy advantage of using MobileNet model over VGG is that MobileNet are light weight networks that is if we assume that VGG has 140 million parameters then Mobilenet will have only 4 million parameters so it results in less computation

There are three ways in which we can use pre-trained networks/models for the purpose of transfer learning:

1. Feature Extraction
2. Fine Tuning
3. Prediction

Feature Extraction

When we give input to the model then first it passes through convolutional base where feature extraction takes place and then goes to classifier that assigns probability value for each class and the class having highest probability value the input belongs to that class.

So, in feature extraction transfer learning what we do is that we freeze the convolutional base which is responsible for feature extraction because it is generic for all images and has already calculated weights and then define our own classifier for classifying the images and train that the classifier to classify the image in the given number of classes (suppose the initial classifier is classifying the image into 100 classes now the classifier which we created classifies the image into 3 classes only). So, instead of training the model for feature extraction we can use pre-trained model for feature extraction and then create our own classifier for classification of images.

Prediction

Since modern Convolutional Networks take 2-3 weeks to train across multiple GPUs on ImageNet, it is common for people to release their final Convolutional Networks checkpoints for the benefit of others who can use the networks for fine-tuning. We can directly load these models and use them for predicting in which of the following class our input image belongs to, without explicitly training our model.

Fine Tuning

In fine-tuning, we not only replace and retrain the classifier present on top of the Convolutional Network but we also fine tune the weights of the pre-trained network by using the back propagation. We can perform this fine-tuning of the whole convolutional network but generally this fine-tuning is performed on higher-lower portion of the network because lower-level layers are very generic since they extract edges, patterns, what kind of texture is present in image, etc. but higher layer layers determine how different parts are combine to form specific object thus we do not need to tune the lower level layers. Suppose we have a different dataset and we want to combine the features learnt by the smaller layers in the different manner then fine tuning is useful in such scenarios.

Another thing which we need to keep in mind when we are applying fine tuning then we have to keep the learning rate very small because when we apply weight update rule, $w=w-\eta * \Delta w$, then learning rate should be kept small because if we make it large then we are effectively destroying the weights learnt by the model earlier. So, in this case we freeze the lower level layers and just fine-tune the higher level layers to make the model learn the classifier weights.

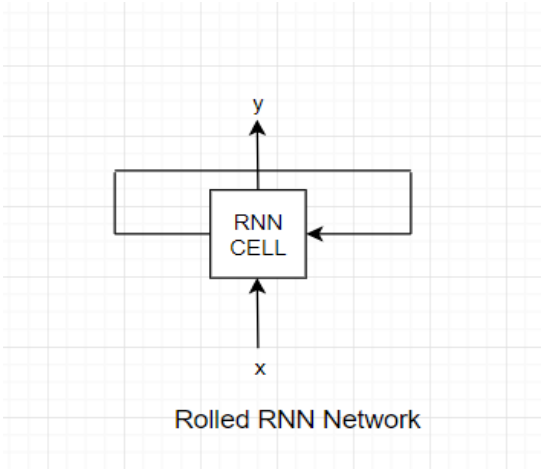
1.6 RNN

RNN is used for effectively drawing information from textual data. For example we have textual data like "This is a great movie" on the basis of this sentence we want to assign the movie which are talking about in the sentence a rating of 4 out of 5. One way to do this we can take our entire sentence and transform into a fixed length vector and we feed to a multilayer perceptron to do the classification which basically will classify the move in the category of 4 star. But, this approach does not work very well with the sentences the reason being in MPL, the ordering of the words does not matter, it just sees what all words are present in the sentence and it makes the prediction. So, if in case we have sentence like this "This is not a great movie" then it might again give you positive result or put in the 4 or 3 star category.

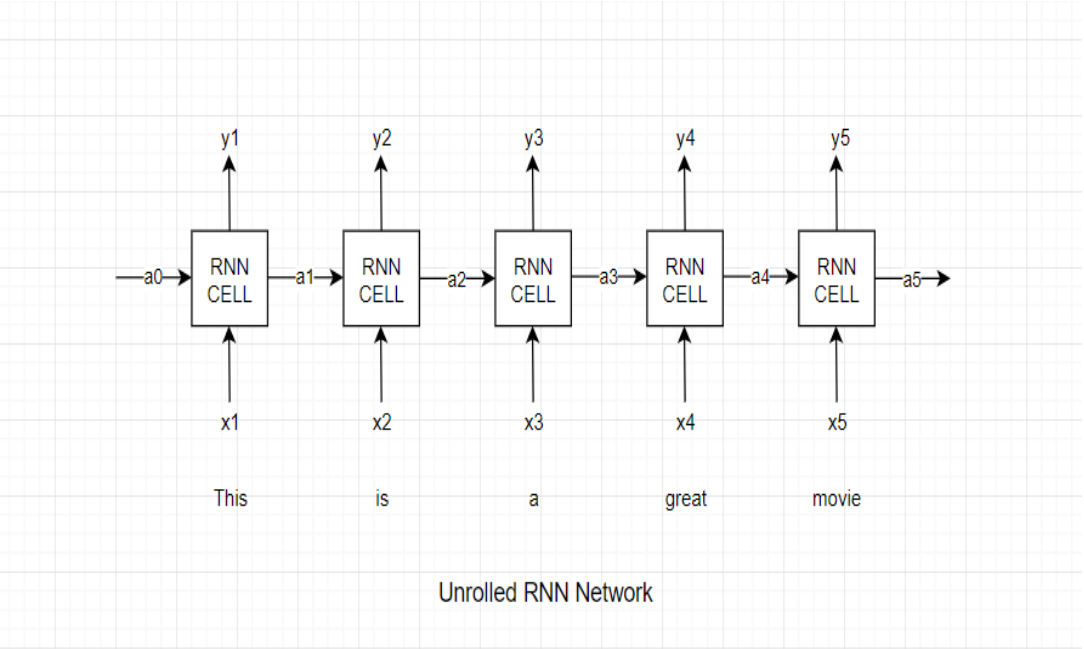
Therefore, to get rid of this drawback we can use sequence models. The whole idea of sequence models is to process information word by word. Suppose we have sentence like this "This is a great movie" then as humans process the sentence from left to right processing each word at a time and maintaining a state in brain about what we have read so far in the similar fashion sequence models also process the data.

For example if we have a sentence "This is a great movie but the ending was not up to the mark".

So, if we give this sentence to CNN or MLP as they are stateless models they do not remember what previous word or sentence they have processed but here in this sentence as we can see that the sentence should be given 3 start out of 5 stars because in the beginning the client told that the movie was great but then after he told that the ending was not up to the mark. So, this is a kind of mixed review and hence the feedback should be given neutral class but if the model was not able to remember the previous states then it might have not classified the sentence in the correct category. Therefore, Recurrent Neural Network came into existence to avoid such problems, which is a sequence model.



Recurrent Neural Network consists of RNN cells, which performs some function on the given inputs (x) and produces some outputs (y) that is word by word processing takes place and a state-vector (a) is also maintained which is subsequently passed to each RNN cells and consequently gets updated by it also and the update state vector is passed to next RNN cell. Here, the state vector maintains at each step what the model has seen so far.



1.7 Word Embedding

When we give textual data to our model we need some kind of representation to represent that data. One of the way is to make a vocabulary by combining all the unique words, which are present in the dictionary and we associate each word with a unique index. Now, if we want to represent a word "movie" then we will pick word "movie" from vocabulary and find its unique index then convert that index into one hot encoded vector of total words present in that vocabulary (one hot encoded vector means all the indexes of the vector will have value 0 except the index which represents the word "movie" in the vocabulary). But this type of representation has one type of problem that is it is sparse which means that it has a lot of zeroes depending on the size of vocabulary. If we pass that vector to our model then it will increase the time consumed by the model when we train it. So, here the goal is to use compact representation instead of using a very large sized sparse one hot encoded vector for representing word. Using the compact vector for representation purpose is called as embedding and the compact vectors are also called as distributed representations.

There are two ways achieving this goal if we have very large training dataset and we are using deep learning to train our model then we can train it along with whatever task we are trying to achieve. Second option is we can start with pre-trained embedding.

There are two popular embedding's that are available:

- Word2Vec- generate by Google
- Glove Vectors –generated by Stanford

These embedding's are specifically useful when we have smaller dataset. In keras, we also have one layer called embedding layer. It consist of huge matrix called embedding matrix having number of rows equal to the size of vocabulary and number of columns (k) as size of compact vector. Earlier if our vocabulary was of size 10,000 then we were using a sparse vector of size 10,000 to represent a word but now if we try to convert that sparse vector to compact vector using glove embedding then we can reduce its size to k which are number of columns that are present in the word embedding matrix. The task of embedding layer is to convert one very large sparse vector into a smaller size compact vector.

1.8 LSTM

In RNN, there are is a problem of long-term dependency, which results in reducing the effectiveness of the algorithm.

There are basically two types of dependencies:

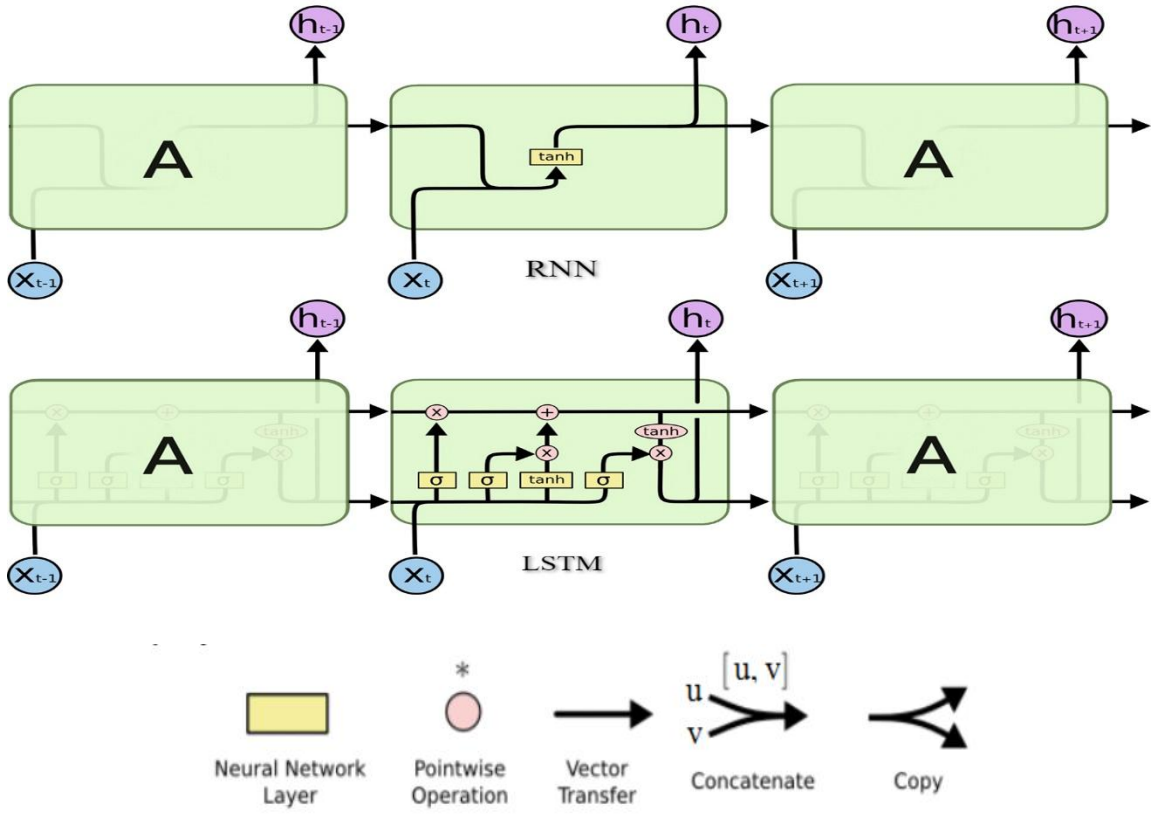
- Short-term dependency
- Long-term dependency

Let us understand **short-term dependency** with the help of an example. Suppose we have built a model using RNN, which predicts the next word based on the incomplete sentence passed to the model. For example, we

passed a sentence "The clouds are in the _____" and the model has to predict the word after "the" in the sentence so here the most important word which the model has take into consideration in order to predict the next word is the word "cloud". Since the most important word that tells us about the context and word which we have to predict are very close to each other. So, it becomes the case of short-term dependency and in those cases our regular RNN model can effectively predict the next word.

However, in case of **long-term dependency** suppose we have given a sentence "I grew up in France. I have done my schooling from Joseph Fourier University (UJF). After my education got completed I got a job. As I was able to speak fluently, _____" now using this paragraph the RNN model has to predict the next word. When it sees the words "speak", "fluently" RNN will be able to understand that the next word must be some sort of language. In order to determine which language it has to write in the blank space it has to go the very first sentence "I grew up in France", to understand the context of the paragraph then only RNN will be able to determine it has to use "French" word for prediction. However, because of vanishing gradients, RNN cannot remember the word "France" and hence it will not be able to produce the correct result. So, if our input word is at time-stamp (position) 1 or 2 and the word, which we have to predict, is at position 30 or 40 then the problem of predicting the next word comes under long-term dependency. Which creates a problem in predicting the next word because RNN will not able to remember that at time-stamp 1 or 2 you have given the word "France" to RNN because of vanishing gradients.

To solve the problem LSTM is used, architecture of regular RNN and LSTM is shown in below figures:



As we can see the from the RNN has only 1 neural network whereas LSTM has 4 neural networks. So number of parameters in LSTM becomes exactly the four times of parameters present in RNN.

II. About Model

2.1 Model The model takes the image as an input and then uses certain algorithm to describe the image that is it performs captioning of the image. We have treated this problem as a supervised learning problem in which we can extract X (particular image) and Y (captions associated with that image) from the Flickr8k dataset. Our goal here is to learn some mapping between X and Y that is we have to learn y as a function of x (y=f(x)).

2.2 Dataset The dataset that I used for building this model is Flickr8k, There are different variation of Flickr dataset that are available on internet like Flickr30k which has 30,000 images. Flickr8k is a kind of subset

of Flickr30k, which consists of 8000 images and with each image five captions are associated to describe the image in different ways. Therefore, in total, there are 40,000 captions that are available and using this dataset, we have to train our model to predict the caption of any given input image. The dataset is already is split into 3 parts, out of 8000 images 6000 images are used for training, 1000 images for validation and 1000 images for testing.

2.3 Data Cleaning Before feeding data to our model, we have to clean that data so that our model will be able to process our data in the best possible way. Sometimes we perform this cleaning via removal of stop-word, lemmatization, etc. However, for this model we will follow different approach for cleaning data. For this model, we cannot remove stop-words like is, am, an, etc. and then process the captions because if we do not teach our model that how it should use stop-words to construct the English sentences then it won't be able to generate the correct sentence. For data cleaning, we cannot use stemming also because if we feed in stemmed words to our model then our model will only be able to learn stemmed words for example if the word is "drawing" and we feed "draw" to our model then our model will use the word "draw" instead of "drawing" wherever "drawing" is required. Hence, it will result in prediction of sentence like "Girl is draw something on paper" instead of "Girl is drawing something on paper". Therefore, first we have cleaned the data by converting everything into lower case so that the word "The" and "the" are treated same. Then we have removed non-alphabetical entities from captions like punctuations, special characters, numbers, etc., so that our model just generate sentences without any numbers or special characters and which in turn helped in reducing the vocab size. Reducing vocab size is very important because if we have less vocab size then it means less number of neurons, which in turn results in less parameters, and hence reduces the chances of overfitting of the model. It also reduces the computation time because the model assigns every word of vocab the probability, which indicates that how much, is the chance for any word to be the next word of the sentence, if we have small number of words in vocab then probability distribution will be small which in turn will reduce the computation time.[3]

2.4 Creating Vocab Vocabulary is the set all possible unique words that our model can predict. From the cleaned descriptions, which we have found in the data-cleaning step, we will find the set of all unique words and then use those unique words to construct the vocab. Generating a vocab is important because at the end of each iteration, our model will predict the probability distribution for numbers, where each number is mapped with the unique words present in the vocab and this vocab will help in finding the word associated with the maximum probability number predicted by model. One additional thing, which we can do, is after finding out all the unique words, we can shorten our vocab size by keeping only those words in our vocab, which are most frequent. That is we can set a particular threshold value for frequency if any word crosses that threshold then only we put that word in our vocab. In order to generate text we will use LSTM based layer to generate text from image. At every step word generated from previous step will go as an input the current LSTM cell but this process can repeat infinite number of times so in order to make our model know when to stop we have to use special tokens like <e> which marks the end of sentence. However, the model will only be able to generate <e> end of sentence token if we have used this token while training the model. So what we do we have to add two tokens to our caption <s> and <e> which are used for marking the start and end of sentences respectively. With the help of these token our model is going to recognize from where our sentence started and from where it ended.[3]

2.5 Image and Caption Preprocessing We will use transfer learning to convert our image and caption into features that is we will extract features from image and caption using transfer learning. There are two ways using which we can use transfer learning one is pre-trained model and other is fine-tuning. So, for this model to extract features from image we have used pre-trained model. The pre-trained model, which we used for feature extraction, is resnet50. Resnet50 model is already trained on imagenet dataset. It is very deep model having 50 layers and it has also skip connections which means the gradients can back-propagate easily hence, it does not suffers from vanishing gradients problem. Resnet50 model is not sequential model because of presence of branches in form of skip connections. These skip connections describes that the input layer is connected to which layers that is why when we print summary of the model we also see one other column "Connected to". Activation_98 layer is responsible for giving the activation maps which has the shape of (7, 7, 2048) and after that one way is that we can directly use the output and flatten it but it will result in making 7*7*2048 neurons, which will result in a lot parameters. Other way is that we can use global_average_pooling layer to squeeze every channel into a single scaler the value of which is calculated by taking the average of all the parameters present in the 7*7 channel, that will result in reducing the neurons by 49 times i.e. we wil1 just get a list of 2048 neurons (1*1*2048). These 2048 neurons tells what all features present in the particular image. If any neuron amongst those 2048 neurons is having a higher activation value then it indicates that, that particular feature is present in that image. So, what we did instead of using the whole resnet50 model, using the functional API of keras we have created a new model by taking the resnet50 model only till the GAP (global_average_pooling) layer. It will give the activation map of 2048 features and from that feature-map we will be able to identify that what features are present in the particular image. So, using that we will able to convert our image into numbers.

Now, for caption preprocessing we have converted the captions into numbers using two dictionary idx_to_word and word_to_idx which help in mapping the index numbers to words and words to index numbers. So, when we give any caption to our model we give it in the form of numbers by converting words to index numbers and when model predicts any caption for the input image in the form of index numbers then we can again convert those numbers into words using those dictionaries.[5][3]

2.6 Image Captioning As A Supervised learning Model In supervised learning, we try to predict y when value of x is given. For this model, x is input image and y is the caption for that image which we have to predict. We cannot predict whole sentence at a time. The goal is to predict one word at a time until we hit the end of the sentence. During training time of the model at every step, we will feed ground truth along with previously generated words to the model instead of giving the actual word predicted by it so that our model is able to effectively determine the weight matrix for LSTM. But during test phase since we do not have ground truth available with us so at each step we will give the LSTM only the previously generated words to predict the next word. This concept is used in language modelling.[3]

	Xi		Yi	
i	Image feature vector	Partial Caption	Target word	
1	Image_1	startseq	the	data points corresponding to image 1 and its caption
2	Image_1	startseq the	black	
3	Image_1	startseq the black	cat	
4	Image_1	startseq the black cat	sat	
5	Image_1	startseq the black cat sat	on	
6	Image_1	startseq the black cat sat on	grass	
7	Image_1	startseq the black cat sat on grass	endseq	
8	Image_2	startseq	the	data points corresponding to image 2 and its caption
9	Image_2	startseq the	white	
10	Image_2	startseq the white	cat	
11	Image_2	startseq the white cat	is	
12	Image_2	startseq the white cat is	walking	
13	Image_2	startseq the white cat is walking	on	
14	Image_2	startseq the white cat is walking on	road	
15	Image_2	startseq the white cat is walking on road	endseq	

Data Matrix for both the images and captions

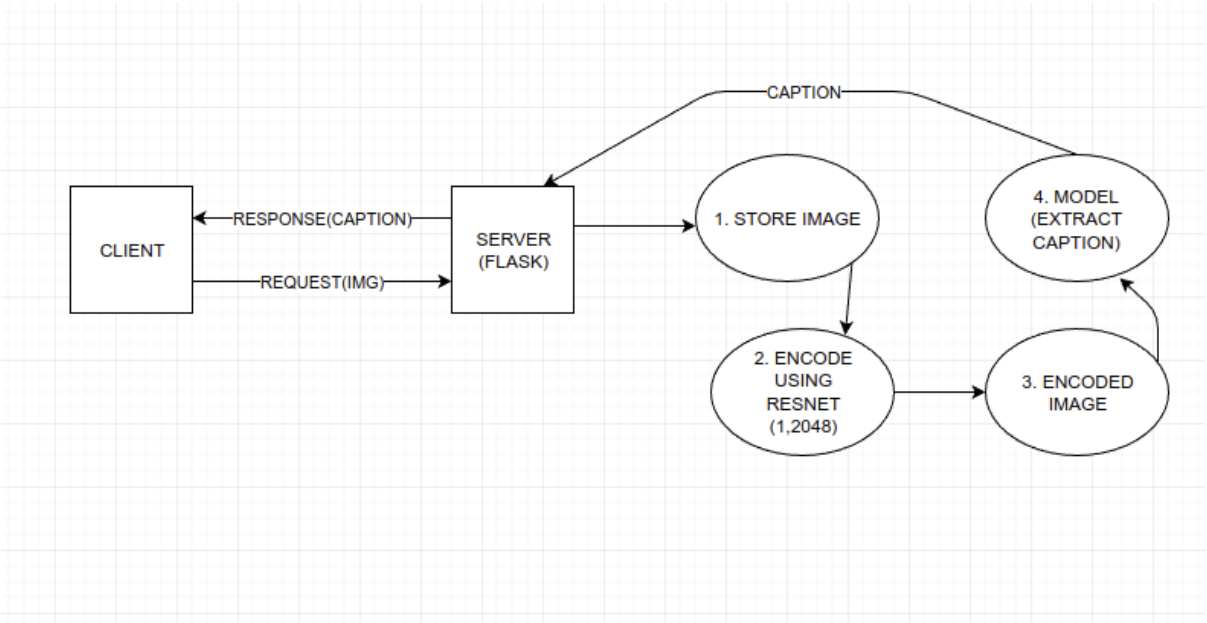
	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	[9]	10
2	Image_1	[9, 10]	1
3	Image_1	[9, 10, 1]	2
4	Image_1	[9, 10, 1, 2]	8
5	Image_1	[9, 10, 1, 2, 8]	6
6	Image_1	[9, 10, 1, 2, 8, 6]	4
7	Image_1	[9, 10, 1, 2, 8, 6, 4]	3
8	Image_2	[9]	10
9	Image_2	[9, 10]	12
10	Image_2	[9, 10, 12]	2
11	Image_2	[9, 10, 12, 2]	5
12	Image_2	[9, 10, 12, 2, 5]	11
13	Image_2	[9, 10, 12, 2, 5, 11]	6
14	Image_2	[9, 10, 12, 2, 5, 11, 6]	7
15	Image_2	[9, 10, 12, 2, 5, 11, 6, 7]	3

Data matrix after replacing the words by their indices

2.7 Prediction Our model will accept the photograph vector along with some partial sequences. We will give some input_text to our model, which is initially "start_sequence". In each iteration, we will give the sequence which was previously generated by the model along with the current word predicted by it and we will break from the loop when the model predicts the end sequence as next word.[3]

III. Pipeline For Deployment On web

In this project, the pipeline that I used is depicted via the above figure. So, first when the client opens the website and uploads the image for which he wants the caption that request goes to the server which is managed by flask. At the server side the uploaded image first gets saved and since we do not give directly the uploaded image to our saved model first we compute the encoding of the image from resnet model which gives us (1,2048) vector of encoding and then this encoding we pass to our saved model which then extracts caption from the image and then this extracted caption is sent to the client in the form of response and shown on the screen[4]



IV. Conclusion

The main part of this research paper is the deep learning which I have to use to implement this model, at certain instances I got very lost while building this model since there are a lot of things which I did use in this model to make it work like RNN, LSTM, resnet50, word embeddings, transfer learning, language models, etc. Yes, I am no expert in those technologies and I am also not able to understand them completely. I took help from various sources, which helped me a lot to build this model and I am really thankful to creators of these sources as the content put me on right track whenever I got lost.

The model which I created is not up to the mark I know there are still some small flaws in it and it can be improved to predict the captions of images more accurately. So, in future I will definitely try to remove those small flaws and improve the accuracy of the model.

References

[1]. Deep Learning and Convolutional Neural Networks for Medical Imaging and Clinical Informatics (Advances in Computer Vision and Pattern Recognition) By Le Lu (Editor), Xiaosong Wang (Editor), Gustavo Carneiro (Editor), Lin Yang (Editor)
[2]. Perceptrons Author, Marvin Minsky., Author, Seymour Papert
[3]. Author Harshall Lamba https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8
[4]. Coding Blocks Data Science Course
[5]. Author Muhammad Abdelhadie Al-Malla https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00571-w
[6]. Flanagan, David. JavaScript - The definitive guide (6 ed.). p. 1. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages and JavaScript to specify the behaviour of web pages.
[7]. "JavaScript data types and data structures - JavaScript | MDN". Developer.mozilla.org. February 16, 2017. Archived from the original on March 14, 2017. Retrieved February 24, 2017.
[8]. Clinick, Andrew (July 14, 2000). "Introducing JScript .NET". Microsoft Developer Network. Microsoft. Archived from the original on November 10, 2017. Retrieved April 10, 2018. [S]ince the 1996 introduction of JScript version 1.0 ... we've been seeing a steady increase in the usage of JScript on the server—particularly in Active Server Pages (ASP).
[9]. "ISO/IEC 9001: Quality management systems -- Requirements," International Organization for Standardization, 1999.
[10]. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26(2):1–26, 2008.