***MINOR PROJECT***

***Image Captioning***

*Submitted in partial fulfillment of the requirements*

*for the award of the degree of*

**Bachelor of Technology**

**Computer Science and Engineering**

Guide:

**Ms. Teena Verma** Submitted by:

**Team** -17



**HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT**

**HAMIDPUR, DELHI 110 036**

**Affiliated to**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY**

**Sector – 16C Dwarka, Delhi – 110075, India**

**2017-2021**

**DECLARATION**

We, students of B.Tech(Computer Science and Engineering) hereby declare that the major project entitled “**Image Captioning**” which is submitted to Department of Computer Science and Engineering, HMR Institute of Technology and Management, New Delhi, affiliated to Guru Gobind Singh Indraprastha University,Dwarka(New Delhi) in partial fulfillment of requirement for the award of the degree of Bachelor of technology in Computer Science and Engineering, has not been previously formed the basis for the award of any degre, diploma or other imilar title or recognition.The list of members involved in the project is listed below:-

S.No. Student Name Enrollment Number Student Signature

01. Deepak Kumar 00696502717

02. Deepak Kumar Rai 00796502717

03. Jnaneshwar 01096502717

04. Manthan Singh 01396502717

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date : \_\_\_\_\_\_\_\_\_\_\_\_

Place : New Delhi

Ms. Teena Verma

Head of the Department Assistant Professor

Computer Science and Engineering Computer Science and Engineering

HMRITM Hamidpur, New Delhi HMRITM Hamidpur, New Delhi

**Acknowledgement**

First of all, We would like to express my gratitude to our Parents and Almighty to enabling us to complete this report on **“Image Captioning”** Successfully completion of any type of project requires helps from a number of persons. We have also taken help from different pepole for the preparation of this project. Now there is a little effort to show my deep gratitude to that helpful person.

We convey our sincere gratitude to **Guide** **Ms. Teena Verma Ma’am**, and our **Proctor Mr. Ravi Arora Sir**, Associate Professor of Department of Computer Science and Engineering, HMR Institute of Technology and Management , New Delhi. Without his kind direction and proper guidance this study would have been a little success. In every phase of the project his supervision and guidance shaped this report to be completed perfectly.

Last but not the least We would like to thanks my friends and classmates to give a clear idea of our project and helping us to implementing.

**Abstract**

Artificial Intelligence(AI) has emerged as one of the decisive expertise with applications across various industry domains. Machine Learning(ML), a subset of AI, is an important set of algorithms used for solving several business and social problems. The objective of this project is to discuss machine learning development using Python. AI and ML are important skills that every graduate in engineering and management discipline will require to advance in their career. Companies across the globe and across various different industrial sectors such as retail, finance, insurance, manufacturing, healthcare are planning to buid and offer AI-based solutions to stay competitive in the market. Companies across the different domains will need a large pool of experts in these fields to build products and solutions for them, An estimate by IDC(International Data Corporation) states that spending by organizations on AI and ML will grow from $12B in 2017 to $57.6B in 2021. Almost 6% of organizations have reported that Machine Learning and Artificial Inteligence will be among their top data initiatives in 2019. Machine Learning is going to be most in demand AI skill for data scientists by 400%, but in contrast the supply has only increased by 19%.

**CONTENTS**

1 INTRODUCTION

                           1.1 Overview

  1.2 Purpose

2 LITERATURE SURVEY

  2.1 Existing problem

  2.2 proposed solution

3 THEORITICAL ANALYSIS

  3.3 Block diagram

  Hardware/Software designing

4 EXPERIMENTAL INVESTIGATIONS

5 FLOWCHART

6 RESULT

7 ADVANTAGES & DISADVANTAGES

8 APPLICATIONS

9 CONCLUSIONS

10 FUTURE SCOPE

11 BIBLIOGRAPHY

APPENDIX

1. Source code
2. Github Link

**OBJECTIVE**

* The objective of our project is to learn the concepts of a CNN and LSTM model and build a working model of Image caption generator by implementing CNN with LSTM.
* In this Python project, we will be implementing the caption generator using ***CNN (Convolutional Neural Networks)***and LSTM (Long short term memory). The image features will be extracted from Xception which is a CNN model trained on the imagenet dataset and then we feed the features into the LSTM model which will be responsible for generating the image captions.

**THE DATASET**

* For the image caption generator, we will be using the Flickr\_8K dataset. There are also other big datasets like Flickr\_30K and MSCOCO dataset but it can take weeks just to train the network so we will be using a small Flickr8k dataset. The advantage of a huge dataset is that we can build better models.
* Thanks to Jason Brownlee for providing a direct link to download the dataset (Size: 1GB).
* Flicker8k\_Dataset
* Flickr\_8k\_text

The Flickr\_8k\_text folder contains file Flickr8k.token which is the main file of our dataset that contains image name and their respective captions separated by newline(“\n”).

**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.

It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. It can handle the images that have been translated, rotated, scaled and changes in perspective.

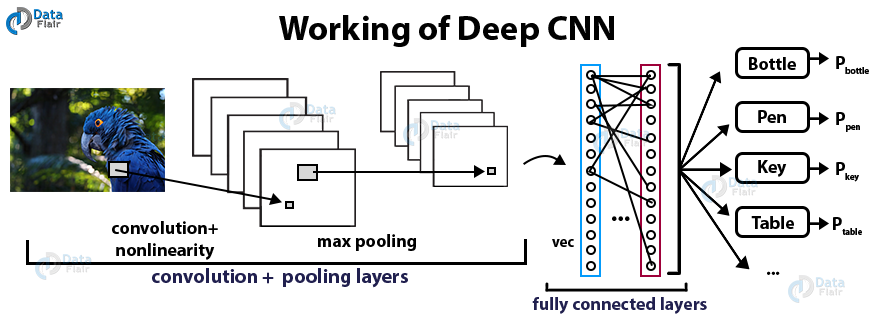


Fig: Architecture of CNN

**What is Image Classification ?**

Image classification takes an image and predicts the object in an image.

For example, when we built a cat-dog classifier, we took images of cat

or dog and predicted their class:

**Image Classification**

What do you do if both cat and dog are present in the image ?

**Image Classification**

What would our model predict? To solve this problem we can train a

multi-label classifier which will predict both the classes(dog as well as

cat). However, we still won’t know the location of cat or dog. The

problem of identifying the location of an object(given the class) in an

image is called **localization**. However, if the object class is not known,

we have to not only determine the location but also predict the class of

each object.

**Image Classification**

**Predicting the location of the object along with the class is called object**

**Detection.** In place of predicting the class of object from an image, we now have to

predict the class as well as a rectangle**(called bounding box)** containing that object.

It takes 4 variables to uniquely identify a rectangle. So, for each instance of the

object in the image, we shall predict following variables:

*class\_name,*

*bounding\_box\_top\_left\_x\_coordinate,*

*bounding\_box\_top\_left\_y\_coordinate,*

*bounding\_box\_width,*

*bounding\_box\_height*

Just like multi-label image classification problems, we can have multi-class object

detection problem where we detect multiple kinds of objects in a single image:

**Image Classification**

**Object Detection is modeled as a classification problem** where we

take windows of fixed sizes from input image at all the possible

locations feed these patches to an image classifier.

Each window is fed to the classifier which predicts the class of the

object in the window( or background if none is present).

**Image Classification**

The object can be of varying sizes. To solve this problem an image

pyramid is created by scaling the image.

**Idea is that we resize the image at multiple scales and we count on**

**the fact that our chosen window size will completely contain the**

**object in one of these resized images**.

Most commonly, the image is downsampled (size is reduced) until

certain condition typically a minimum size is reached. On each of these

images, a fixed size window detector is run. It’s common to have as

many as 64 levels on such pyramids. Now, all these windows are fed to

a classifier to detect the object of interest. This will help us solve the

problem of size and location.

**Image Classification**

RCNN

**Region Based Convolutional Neural Network**

Since we had modeled object detection into a classification problem,

success depends on the accuracy of classification. After the rise of deep

learning, the obvious idea was to replace HOG based classifiers with a

more accurate convolutional neural network based classifier.

However, there was one problem. CNNs were too slow and

computationally very expensive. It was impossible to run CNNs on so

many patches generated by sliding window detector.

RCNN

R-CNN solves this problem by using an object proposal algorithm

called **Selective Search** which reduces the number of bounding boxes

that are fed to the classifier to close to **2000 region proposals**.

Selective search uses local cues like texture, intensity, color and/or a

measure of insideness etc to generate all the possible locations of the

object. Now, we can feed these boxes to our CNN based classifier.

Fully connected part of CNN takes a fixed sized input so, we

resize(without preserving aspect ratio) all the generated boxes to a

fixed size (224×224 for VGG) and feed to the CNN part.

RCNN

There are 3 important parts of R-CNN:

• Run Selective Search to generate probable objects.

• Feed these patches to CNN, followed by SVM to predict the class of each

patch.

• Optimize patches by training bounding box regression separately.

**Selective Search:**

• Generate initial sub-segmentation, we generate many candidate regions.

• Use greedy algorithm to recursively combine similar regions into larger

ones.

• Use the generated regions to produce the final candidate region proposals.

RCNN

• These 2000 candidate region proposals are warped into a square and

fed into a convolutional neural network that produces a 4096-

dimensional feature vector as output.

• The CNN acts as a feature extractor and the output dense layer

consists of the features extracted from the image and the extracted

features are fed into an SVM to classify the presence of the object

within that candidate region proposal.

• In addition to predicting the presence of an object within the region

proposals, the algorithm also predicts four values which are offset

values to increase the precision of the bounding box.

RCNN

• For example, given a region

proposal, the algorithm would

have predicted the presence of

a person but the face of that

person within that region

proposal could’ve been cut in

half.

• Therefore, the offset values

help in adjusting the bounding

box of the region proposal.

Using Selective Search for Region Proposals

In selective search, we start with many tiny initial regions. We use a

greedy algorithm to grow a region. First we locate two most similar

regions and merge them together. Similarity S b/w region a and b is

defined as :

Where Stexture (a,b) measures the visual similarity, and Ssize prefers

merging smaller regions together to avoid a single region from gobbling

up all others one by one.

Using Selective Search for Region Proposals

We continue merging

regions until everything is

combined together. In the

first row, we show how we

grow the regions, and the

blue rectangles in the second

rows show all possible region

proposals we made during

the merging. The green

rectangle are the target

objects that we want to

detect.

Using Selective Search for Region Proposals

Warping

• For every region proposal, we use a CNN to extract the features. Since

a CNN takes a fixed-size image, we wrap a proposed region into a 227

x 227 RGB images.

Using Selective Search for Region Proposals

Extracting features with a CNN

• This will then process by a CNN to extract a 4096-dimensional

feature:

Using Selective Search for Region Proposals

Classification

• We then apply a SVM classifier to identify the object:

Using Selective Search for Region Proposals

**Bounding box regressor**

The original boundary box proposal

may need further refinement. We

apply a regressor to calculate the

final red box from the initial blue

proposal region.

Example of RCNN

**Step 1**

Train (or download) a classification model (like AlexNet).

Example of RCNN

**Step 2 - Fine-tune the model**

• Change the number of categories from 1000 to 20.

• Remove the last full-connection layer.

Example of RCNN

**Step 3 - Feature extraction:**

• Extract all candidate boxes from the image (selective search).

• For each region: Correct the size of the region to fit the CNN input, do

a forward operation, and output the fifth pooled layer (that is, the

candidate box extracted features) to the hard disk.

Example of RCNN

**Step 4**

• Train an SVM classifier (2 categories) to determine the category of objects in this

candidate box.

• Determine whether the SVM belongs to each of the categories. If so the result is

positive, and negative if not. For example, the following figure is the SVM for a

dog classification.

Example of RCNN

**Step 5**

Use regression to fine-tune the position of the candidate boxes. For

each class, train a linear regression model to determine if the box is

optimized.

Problems with RCNN

• It still takes a huge amount of time to train the network as you would

have to classify 2000 region proposals per image.

• It cannot be implemented real time as it takes around 47 seconds for

each test image.

• The selective search algorithm is a fixed algorithm. Therefore, no

learning is happening at that stage. This could lead to the generation

of bad candidate region proposals.

SPPnet & Fast R-CNN

**R-CNN is slow in**

**training & inference.** We

have 2,000 proposals

which each of them

needed to be processed

by a CNN to extract

features, and therefore,

R-CNN will repeat the

ConvNet 2,000 times to

extract features.

SPPnet & Fast R-CNN

Hence, instead of converting 2,000 regions into the corresponding

features maps, we convert the whole image once.

SPPNet

Visualization of the

features maps in a

CNN.

SPPnet

SPPnet uses a regional proposal method to generate region of interests

(**RoIs**). The blue rectangular here shows one possible region of interest:

SPPnet

• Here we warp region of interests (RoIs) into spatial pyramid pooling

(SPP) layers.

Each spatial pyramid layer is in a different scale, and we use maximum

pooling to warp the original ROI to the target map.

We pass it to a fully-connected network, and use a SVM for

classification and a linear regressor for the bounding box.

Fast RCNN

Instead of generating a pyramid of layers, Fast R-CNN warps ROIs into

one single layer using the ROI pooling.

Fast RCNN

R-CNN needs many proposals to be accurate and many regions overlap

with each other. **R-CNN is slow in training & inference.** If we have

2,000 proposals, each of them is processed by a CNN separately, i.e. we

repeat feature extractions 2,000 times for different ROIs.

Instead of extracting features for each image patch from scratch, we

use a **feature extractor** (a CNN) to extract features for the whole image

first.

We also use an external region proposal method, like the selective

search, to create ROIs which later combine with the corresponding

feature maps to form patches for object detection.

Fast RCNN

We warp the patches to a fixed size using **ROI pooling** and feed them to

fully connected layers for classification and **localization** (detecting the

location of the object).

By not repeating the feature extractions, **Fast R-CNN** cuts down the

process time significantly.

Here is the network flow:

Fast RCNN

In the code, the expensive feature extraction is moving out of the forloop,

a significant speed improvement since it was executed for all

2000 ROIs. Fast R-CNN is 10x faster than R-CNN in training and 150x

faster in inferencing.

Fast RCNN

One major takeaway for Fast R-CNN is that the whole network (the

feature extractor, the classifier, and the boundary box regressor) are

trained end-to-end with **multi-task losses** (classification loss and

localization loss). This improves accuracy.

**ROI Pooling**

Because Fast R-CNN uses fully connected layers, we apply **ROI**

**pooling** to warp the variable size ROIs into in a predefined size shape.

The RoI pooling layer uses max pooling to convert the features in a

region of interest into a small feature map of H × W. Both H & W (e.g., 7

× 7) are tunable hyper-parameters.

Fast RCNN

Let’s simplify the discussion by transforming 8 × 8 feature maps into a

predefined 2 × 2 shape.

• Top left below: our feature maps.

• Top right: we overlap the ROI (blue) with the feature maps.

• Bottom left: we split ROIs into the target dimension. For example,

with our 2×2 target, we split the ROIs into 4 sections with similar or

equal sizes.

• Bottom right: find the maximum for each section and the result is our

warped feature maps.

You can consider Fast R-CNN is a special case of SPPNet. Instead of multiple

layers, Fast R-CNN only use one layer.

It is feed into a fully-connected network for classification using linear

regression and softmax. The bounding box is further refined with a

linear regression.

The key difference

between SPPnet and Fast

R-CNN is that SPPnet

cannot update

parameters below SPP

layer during training:

**Faster R-CNN**

Fast R-CNN depends on an external region proposal method like

selective search. However, those algorithms run on CPU and they are

slow. In testing, Fast R-CNN takes 2.3 seconds to make a prediction in

which 2 seconds are for generating 2000 ROIs.

**Faster R-CNN**

• Faster R-CNN adopts similar design as the Fast R-CNN except it replaces the region proposal

method by an internal deep network and the ROIs are derived from the feature maps instead.

• The new region proposal network (**RPN**) is more efficient and run at 10 ms per image in

generating ROIs.

The network flow is similar but the region proposal is now replaced by

a convolutional network (RPN).

**Faster R-CNN**

**Region proposal network**

The region proposal network (**RPN**) takes the output feature maps from

the first convolutional network as input.

It slides 3 × 3 filters over the feature maps to make class-agnostic

region proposals using a convolutional network like ZF network.

Other deep network likes VGG or ResNet can be used for more

comprehensive feature extraction at the cost of speed.

The ZF network outputs 256 values, which is feed into 2 separate fully

connected layers to predict a boundary box and 2 objectness scores.

**Faster R-CNN**

The **objectness** measures whether the box contains an object. We can

use a regressor to compute a single objectness score but for simplicity,

Faster R-CNN uses a classifier with 2 possible classes: one for the “have

an object” category and one without (i.e. the background class).

**Faster R-CNN**

For each location in the feature

maps, RPN makes **k** guesses.

Therefore RPN outputs 4×k

coordinates and 2×k scores per

location.

The diagram shows the 8 × 8

feature maps with a 3× 3 filter,

and it outputs a total of 8 × 8 × 3

ROIs (for k = 3). The right side

diagram demonstrates the 3

proposals made by a single

location.

**Faster R-CNN**

Here, we get 3 guesses and we will refine our guesses later. Since we

just need one to be correct, we will be better off if our initial guesses

have different shapes and size.

Therefore, Faster R-CNN does not make random boundary box

proposals.

Instead, it predicts offsets like x, y that are relative to the top left

corner of some reference boxes called **anchors**. We constraints the

value of those offsets so our guesses still resemble the anchors.

**Faster R-CNN**

To make k predictions per location, we

need k anchors centered at each

location. Each prediction is associated

with a specific anchor but different

locations share the same anchor shapes.

**Faster R-CNN**

**Faster R-CNN**

• Those anchors are carefully pre-selected so they are diverse and cover reallife

objects at different scales and aspect ratios reasonable well.

• This guides the initial training with better guesses and allows each

prediction to specialize in a certain shape. This strategy makes early

training more stable and easier.

• Faster R-CNN uses far more anchors. It deploys 9 anchor boxes: 3 different

scales at 3 different aspect ratio. Using 9 anchors per location, it generates

2 × 9 objectness scores and 4 × 9 coordinates per location.

***Anchors*** *are also called* ***priors*** *or* ***default boundary boxes*** *in different papers.*

Both SPPnet and Fast R-CNN requires a region proposal method.

The difference between Fast R-CNN and Faster R-CNN is that we do not

use a special region proposal method to create region proposals.

Instead, we train a region proposal network that takes the feature

maps as input and outputs region proposals. These proposals are then

feed into the RoI pooling layer in the Fast R-CNN.

**Region-based Fully Convolutional Networks (R-FCN)**

Let’s assume we only have a feature map detecting the right eye of a

face. Can we use it to locate a face? It should. Since the right eye

should be on the top-left corner of a facial picture, we can use that to

locate the face.

**Region-based Fully Convolutional Networks (R-FCN)**

• If we have other feature maps specialized in detecting the left eye,

the nose or the mouth, we can combine the results together to locate

the face better.

• So why we go through all the trouble. In Faster R-CNN,

the *detector* applies multiple fully connected layers to make

predictions. With 2,000 ROIs, it can be expensive.

**Region-based Fully Convolutional Networks (R-FCN)**

• R-FCN improves speed by reducing the amount of work needed for

each ROI.

• The region-based feature maps are independent of ROIs and can be

computed outside each ROI.

• The remaining work is much simpler and therefore R-FCN is faster

than Faster R-CNN.

**Region-based Fully Convolutional Networks (R-FCN)**

Let’s consider a 5 × 5 feature map **M** with a blue square object inside. We divide

the square object equally into 3 × 3 regions. Now, we create a new feature map

from M to detect the top left (TL) corner of the square only. The new feature map

looks like the one on the right below. Only the yellow **grid cell** [2, 2] is activated.

**Region-based Fully Convolutional Networks (R-FCN)**

• Since we divide the square into 9 parts, we can create 9 feature maps

each detecting the corresponding region of the object. These feature

maps are called **position-sensitive score maps** because each map

detects (scores) a sub-region of the object.

**Region-based Fully Convolutional Networks (R-FCN)**

• Let’s say the dotted red rectangle below is the ROI proposed. We

divide it into 3 × 3 regions and ask how likely each region contains the

corresponding part of the object. For example, how likely the top-left

ROI region contains the left eye.

• We store the results into a 3 × 3 vote array in the right diagram below.

For example, vote\_array[0][0] contains the score on whether we find

the top-left region of the square object.

This process to map score maps and ROIs to the vote array is

called **position-sensitive ROI-pool**.

After calculating all the values for the position-sensitive ROI pool, the

class score is the average of all its elements.

**Region-based Fully Convolutional Networks (R-FCN)**

• Let’s say we have **C** classes to detect.

• We expand it to C + 1 classes so we include a new class for the

background (non-object).

• Each class will have its own 3 × 3 score maps and therefore a total of

(C+1) × 3 × 3 score maps.

• Using its own set of score maps, we predict a class score for each

class.

• Then we apply a softmax on those scores to compute the probability

for each class.

Mask RCNN

**Extending Faster R-CNN for Pixel Level Segmentation**

Now can we extend such techniques to go one step further and locate

exact pixels of each object instead of just bounding boxes? This

problem, known as image segmentation, is what Kaiming He and a

team of researchers, including Girshick, explored at Facebook AI using

an architecture known as **Mask R-CNN.**

Mask R-CNN does this by adding a branch to Faster R-CNN that outputs

a binary mask that says whether or not a given pixel is part of an

object.

Mask RCNN

Here are its inputs and outputs:

• **Inputs**: CNN Feature Map.

• **Outputs**: Matrix with 1s on all locations where the pixel belongs to

the object and 0s elsewhere (this is known as a binary mask).

But the Mask R-CNN authors had to make one small adjustment to

make this pipeline work as expected.

Mask RCNN

When run without modifications on the original Faster R-CNN

architecture, the Mask R-CNN authors realized that the regions of the

feature map selected by RoIPool were slightly misaligned from the

regions of the original image.

Since image segmentation requires pixel level specificity, unlike

bounding boxes, this naturally led to inaccuracies.

This problem by cleverly adjusting RoIPool to be more precisely aligned

using RoIAlign.

Mask RCNN

• In RoIPool, we would round this down and select 2 pixels causing a

slight misalignment. However, in RoIAlign, **we avoid such rounding.**

• Instead, we use bilinear interpolation to get a precise idea of what

would be at pixel 2.93. This, at a high level, is what allows us to avoid

the misalignments caused by RoIPool.

Once these masks are generated, Mask R-CNN combines them with the

classifications and bounding boxes from Faster R-CNN to generate such

wonderfully precise segmentation

**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.

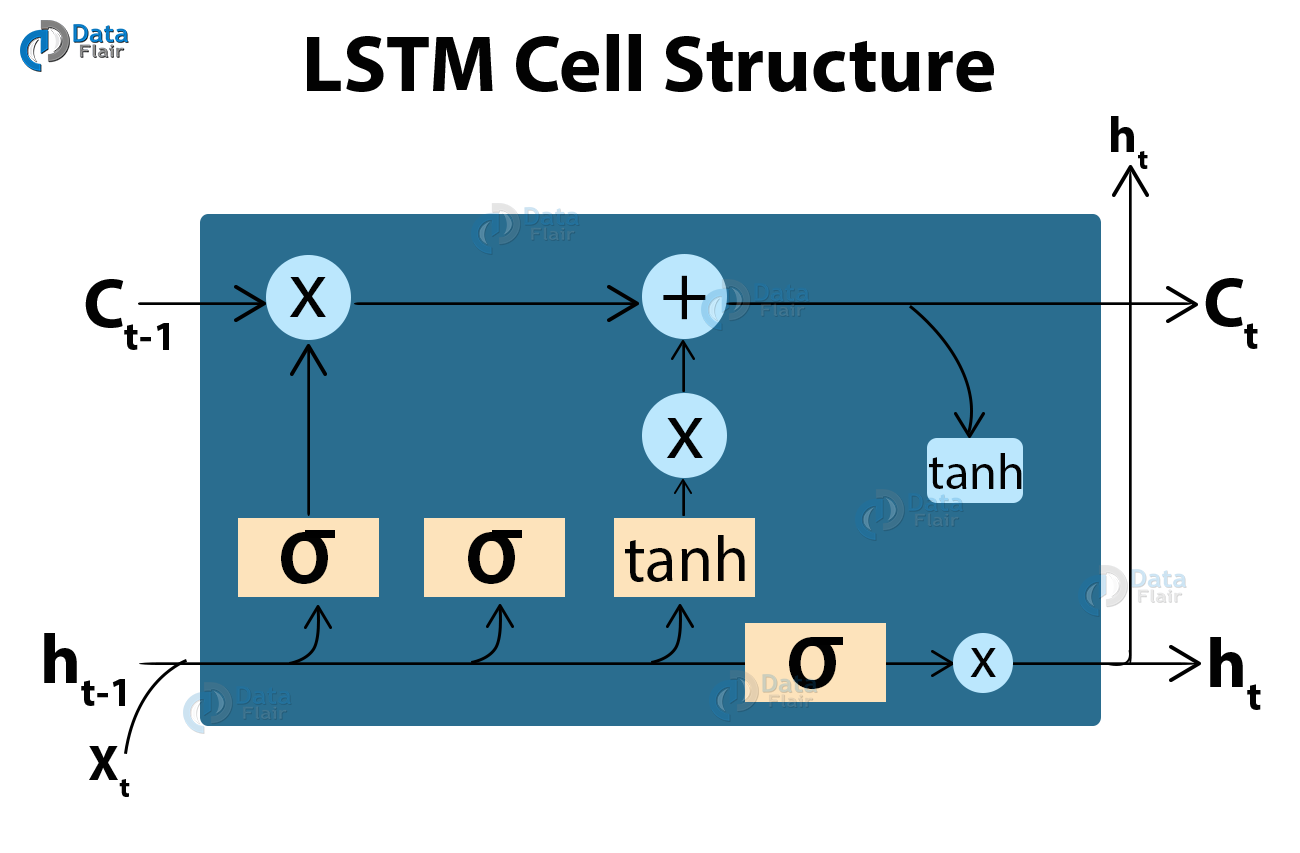


Fig: Architecture of LSTM

**IMAGE CAPTION GENERATOR MODEL**

So, 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. We will use the pre-trained model Xception.

LSTM will use the information from CNN to help generate a description of the image.

**Automatic image annotation** (also known as **automatic image tagging** or **linguistic indexing**) is the process by which a computer system automatically assigns metadata in the form of captioning or keywords to a digital image. This application of computer vision techniques is used in image retrieval systems to organize and locate images of interest from a database.

This method can be regarded as a type of multi-class image classification with a very large number of classes - as large as the vocabulary size. Typically, image analysis in the form of extracted feature vectors and the training annotation words are used by machine learning techniques to attempt to automatically apply annotations to new images. The first methods learned the correlations between image features and training annotations, then techniques were developed using machine translation to try to translate the textual vocabulary with the 'visual vocabulary', or clustered regions known as *blobs*. Work following these efforts have included classification approaches, relevance models and so on.

The advantages of automatic image annotation versus content-based image retrieval (CBIR) are that queries can be more naturally specified by the user. CBIR generally (at present) requires users to search by image concepts such as color and texture, or finding example queries. Certain image features in example images may override the concept that the user is really focusing on. The traditional methods of image retrieval such as those used by libraries have relied on manually annotated images, which is expensive and time-consuming, especially given the large and constantly growing image databases in existence.

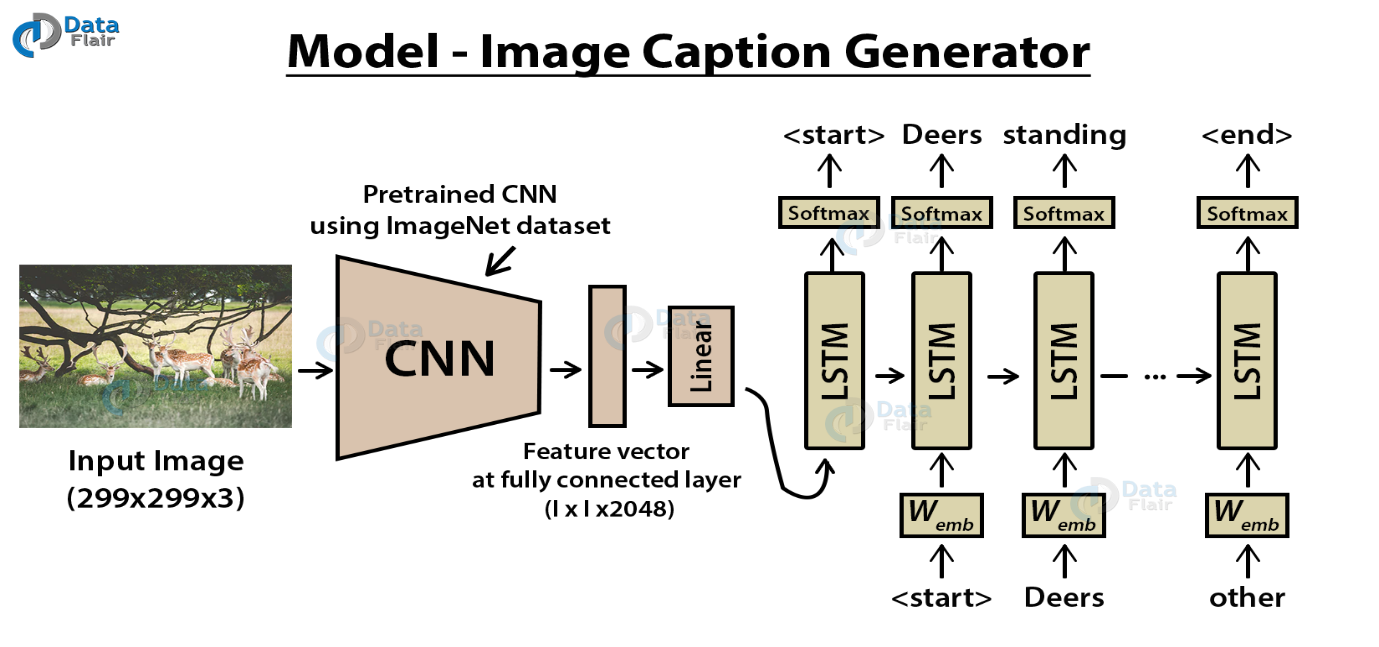
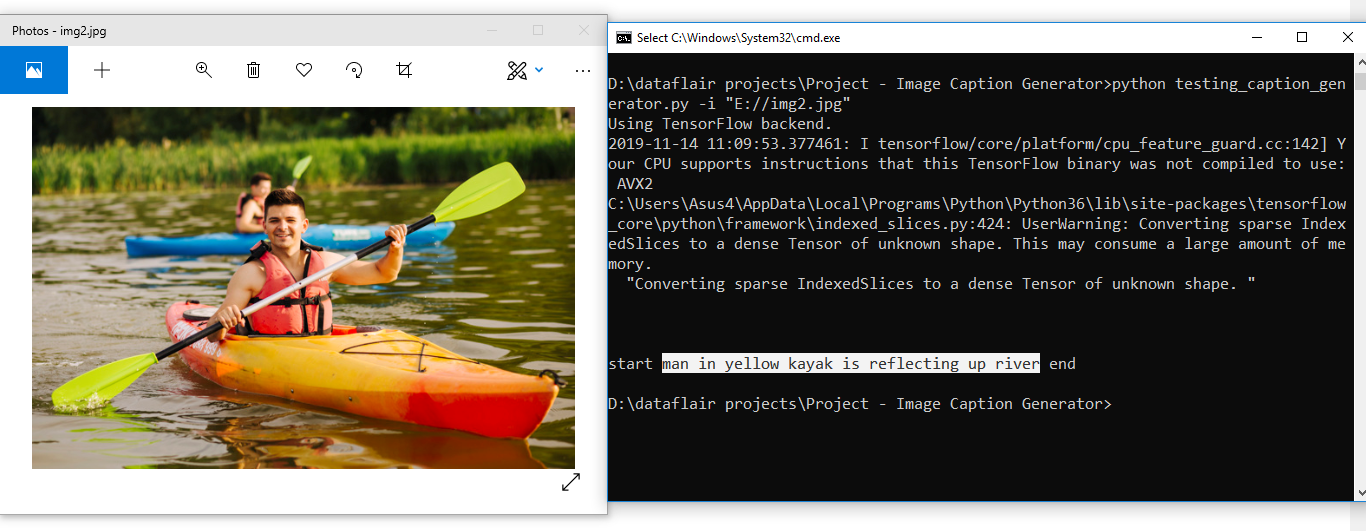
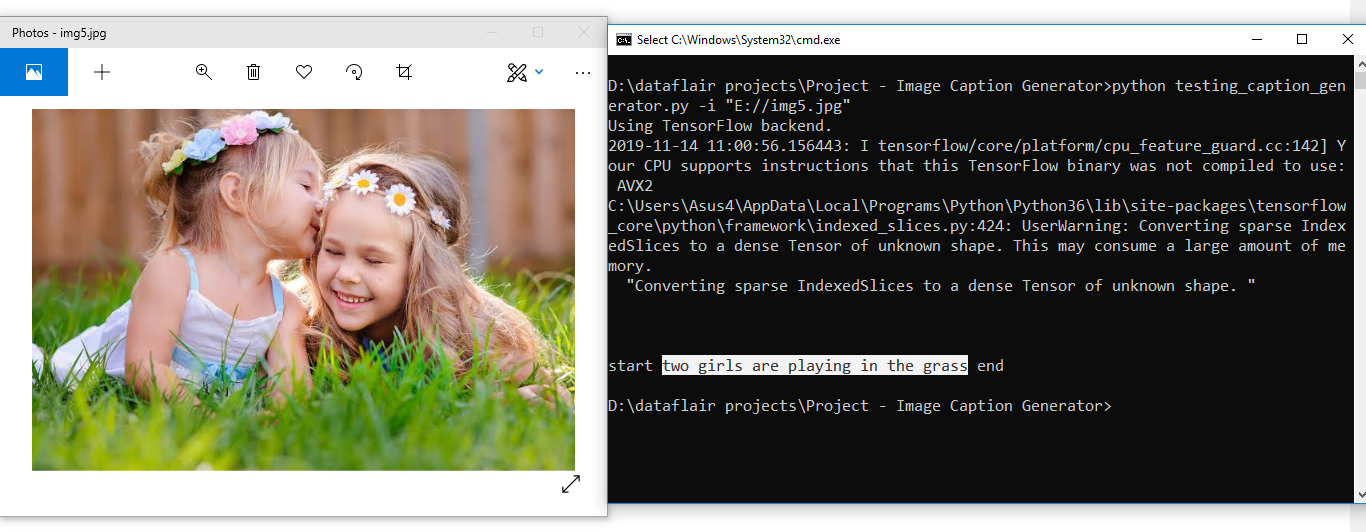


Fig: Architecture of Image Generator Model

**SAMPLE RESULTS**





**SUMMARY**

In this advanced Python project, we have implemented a CNN-RNN model by building an image caption generator. Some key points to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. We used a small dataset consisting of 8000 images. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models.

**BIBLIOGRAPHY**

[https://arxiv.org/pdf/1810.04020.pdf](https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/)

[https://papers.nips.cc/paper/9293-image-captioning-transforming-objects-into-words.pdf](https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/)

<https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/>

<https://machinelearningmastery.com/prepare-photo-caption-dataset-training-deep-learning-model/>

<https://ieeexplore.ieee.org/document/8308186>

<https://arxiv.org/abs/1808.03314>