

A Project Report
On
IMAGE CAPTIONING BOT
Submitted in partial fulfilment of the requirement for the degree of
Bachelor of Technology
In
Electronics and Communication Engineering

Submitted by:

Gurusha Garg (17102241)

Sagar Makhija (17102228)

Under the Supervision of:

Dr. Jasmine Saini



Department of Electronics and Communication Engineering
JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY,
NOIDA

June, 2020

TABLE OF CONTENTS

1. Certificate -----	----- ii
2. Acknowledgement -----	----- iii
3. Chapter 1 – INTRODCUTION -----	-----1
4. Chapter 2 – IMAGE CLASSIFICATION-----	-----3
5. Chapter 3 – IMPLEMENTATION-----	-----15
6. Chapter 4 – SAMPLE OUTPUTS -----	-----21
7. Chapter 5 – CONCLUSION -----	-----23
8. Chapter 6 – FUTURE SCOPE -----	-----24
9. References -----	-----25

CERTIFICATE

It is certified that the work contained in the project report titled **“Image Captioning Bot”** by **“Gurusha Garg (17102241) and Sagar Makhija (17102228)”** has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Dr. Jasmine Saini

ECE

JIIT NOIDA

June, 2020

ACKNOWLEDGEMENT

We would like to acknowledge our parents and all the teachers, who supported us both morally and technically, especially our supervisor Dr. Jasmine Saini who helped us at every step in the making of our project, helped us by clarifying our queries related to our project and technical problems. Also, our special thanks to all class fellows who helped us in clarification of any issue as well as implementation and in documentation.

Signature of Students

Gurusha Garg (17102241)

Sagar Makhija (17102228)

Chapter 1 – INTRODUCTION

1.1 What is AI?

Artificial Intelligence, sometimes called Machine intelligence is intelligence demonstrated by machines. The purpose of AI is to make machines perceive, understand and act on its own, AI can also be considered as giving a brain to a machine so it can think on its own and carry out certain tasks which require a certain ability to think. [2]

1.2 What is Machine Learning and Deep learning?

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

Recently a field of Artificial Intelligence namely Deep Learning is the cause of new revolution in the field of AI in terms of accuracy comparing to the already existing Machine learning algorithms.

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.[6]

1.3 Information about Image Captioning

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given photograph. It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. Recently, deep learning methods have achieved state-of-the-art results on examples of this problem.

Deep learning methods have demonstrated state-of-the-art results on caption generation problems. What is most impressive about these methods is a single end-to-end model can be defined to predict a caption, given a photo, instead of requiring sophisticated data preparation

or a pipeline of specifically designed models. The task of being able to generate a meaningful sentence from an image is a difficult task but can have great impact, for instance helping the visually impaired to have a better understanding of images. The task of image captioning is significantly harder than that of image classification, which has been the main focus in the computer vision community.

A description for an image must capture the relationship between the objects in the image. In addition to the visual understanding of the image, the above semantic knowledge has to be expressed in a natural language like English, which means that a language model is needed.

The attempts made in the past have all been to stitch the two models together. We have tried to combine this into a single model which consists of a Convolutional Neural Network (CNN) encoder which helps in creating image encodings. We could have used some of the recent and advanced classification architectures but that would have increased the training time significantly. These encoded images are then passed to a LSTM network which are a type of Recurrent Neural Network. The network architecture used for the LSTM network work in similar fashion as the ones used in machine translators. The input to the network is an image which is first converted in a 224×224 dimension. We use the Flickr8k dataset to train the model. The model outputs a generated caption based on the dictionary it forms from the tokens of caption in the training set. In the report we first consider the task of image classification separately. We try to classify the images of the Flickr8k dataset using various classifiers. We first try to train the model using a K-Nearest Neighbour classifier. Then we try to apply some linear classifiers. The accuracy with these models was much less than expected since a high loss factor at the time of classification will amplify the loss even further at the time of caption generation. We then try to train a simple Convolutional Neural Network and achieve decent results within few hours of training. Thus, by the end of this section we conclude that CNN are a good fit to be used as the image encoder for the captioning model. In the following sections we discuss briefly about the model used. We then discuss the CNN encoder and the LSTM decoder in detail. The code module of both the architectures is also explained briefly. At the end, we report a few examples tested on the model. [5]

CHAPTER 2- IMAGE CLASSIFICATION

For the task of image captioning we first have to determine a fit model for the task of encoding the image. We discuss three models in the following section.

2.1 K-nearest neighbour classifier

As our first approach, we will explore the concept of a K-nearest neighbour (KNN) classifier. Such classifiers have nothing to do with Convolutional Neural Networks and are very rarely used in practice, but they will allow us to get an idea about the basic approach to an image classification problem.

Suppose we have a training set of 50,000 images divided into 10 different ‘labels’ and we wish to label the remaining 10,000. The k - nearest neighbour classifier will take a test image, compare it to every single one of the training images, and predict the label of the test image based on majority decision of the ‘k’ closest images. A very simple method is used to compare the images - they are compared pixel by pixel and the difference in values is summed up. In other words, given two images and representing them as vectors I_1 and I_2 , the L1 (or Manhattan) distance between them can be calculated as: [1]

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

Test image -				Training image =				Pixel-wise difference			
10	23	45	22	19	17	23	167	9	6	22	145
41	100	34	52	23	56	49	112	18	44	15	60
36	49	90	150	45	34	155	109	9	15	65	41
33	20	200	110	67	22	44	12	34	2	156	98

$$d_1 = 739$$

Alternatively, the L2 (or Euler) distance can be used to compare images:

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Fine-tuning of hyperparameters:

The K-nearest neighbour classifier requires a setting for k. Additionally, there are many different distance functions we could have used (L1 norm/L2 norm etc.). Our goal is to find the best such values of the hyperparameters so as to maximize the accuracy of our classifier.

One would think that an easy way to achieve this would be to try out all possible values of k and pick the one that gives us maximum accuracy on our test data. However, this method should never be used to pick hyperparameters. Using test data to pick hyperparameters results in overfitting i.e our model's results will be too optimistic with respect to what we might actually observe when we deploy your model. In other words, it might fail to generalize to other data.

To overcome this, we used 'cross-validation'. The training data is divided into several 'folds', one fold is used as the 'test fold' and the other folds are used as training folds. Example, in 5-fold cross-validation, we would split the training data into 5 equal folds, use 4 of them for training, and 1 for validation. We would then iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds.

KNN in practice - it's pros and cons:

One advantage of KNN is that it is very easy to implement. However, what we save on implementation time, we lose in computation time later on. KNN classifier takes no time to train, since all that is required is to store and possibly index the training data. However, we pay that computational cost at test time, since classifying a test example requires a comparison to every single training example. This is backwards, since in practice we often care about the test time efficiency much more than the efficiency at training time. Also, the use of L1 or L2 distances on raw pixel values is not adequate since the distances correlate more strongly with backgrounds and color distributions of images than with their semantic content. For example, the L2 distance between the following images is the same:



Fig 2.1: Same L1 distance of three different images (csn231.github.io)

This tells us that same pixel differences don't necessarily translate to same semantic difference. In conclusion, the KNN Classifier may sometimes be a good choice in some

settings (especially if the data is low-dimensional), but it is rarely appropriate for use in practical image classification settings.

2.2 Linear classifiers

In this section, we explore a more powerful approach to image classification that eventually extends to entire Neural Networks and Convolutional Neural Networks. Linear classifiers are an example of parametric classifiers, since we are optimizing some type of numerical values. They have two major components - a score function that maps the raw data to class scores, and a loss function that quantifies the agreement between the predicted scores and the ground truth labels. Then we treat this as an optimization problem in which we minimize the loss function with respect to the parameters of the score function.

Score function

Let's assume a training dataset of images $x_i \in \mathcal{R}^D$, each associated with a label y_i . Here $i=1 \dots N$ and $y_i \in 1 \dots K$. That is, we have N examples (each with a dimensionality D) and K distinct categories. For example, in CIFAR-10 dataset we have a training set of $N = 50,000$ images, each with $D = 32 \times 32 \times 3 = 3072$ pixels, and $K = 10$, since there are 10 distinct labels (dog, cat, car, etc).

We will now define the score function $f: \mathcal{R}^D \rightarrow \mathcal{R}^K$ that maps the raw image pixels to class scores.

$$f(x_i, W, b) = Wx_i + b$$

Where W is the weight matrix and B is the bias vector. W has dimensions 10×3072 , x_i has dimensions 3072×1 and b has dimensions 10×1 .

Essentially, each row of W acts as a classifier for one class of y .

Example- say we have a 4-pixel image that needs to be classified into one of 3 classes. The image will be stretched out into a 12×1 column vector, multiplied with a 3×12 weight matrix and added to a bias vector of dimension 3×1 . The result will be a 3×1 column vector where each row represents the image score for that class.

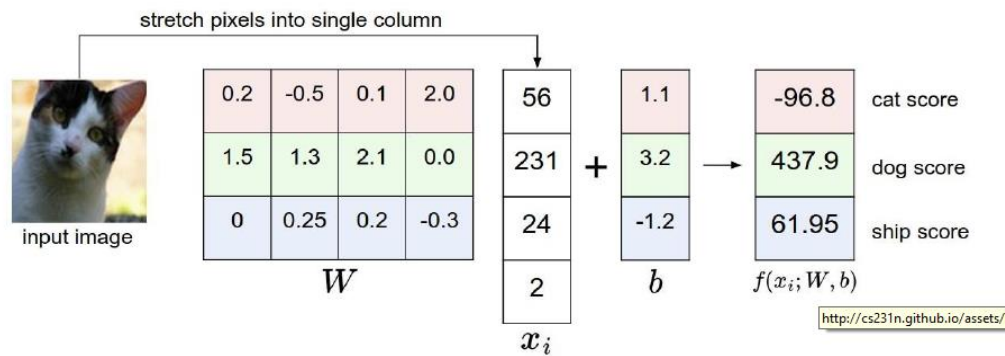


Fig 2.2: Visualisation of score function with 4 pixels (csn231.github.io)

Yet another way to interpret linear classifiers is that each row of the weight matrix W corresponds to a template for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using a dot product one by one to find the one that “fits” best. With this terminology, the linear classifier is doing template matching, where the templates are learned. This is quite similar to nearest neighbour search, except for the fact that one ‘template’ is being constructed per class instead of comparing the test image to thousands of images in each class. [3]



Fig 2.3: Templates of weights generated by linear classifier (ml4a.github.io)

Loss function

Loss functions are used to measure the discrepancy between the model’s prediction and the desired output. In other words, the loss function quantifies our unhappiness with predictions on the training set. Intuitively, the value of the loss function will be high if we’re doing a poor job of classifying the training data, and it will be low if we’re doing well.

2.2.1 Hinge Loss (SVM Classifier)

Hinge loss is set up so that it “wants” the correct class for each image to have a score higher than the incorrect classes by some fixed margin Δ .

The score function takes the pixels and computes the vector $f(xi, W)$ of class scores. For example, the score for the j -th class is the j -th element: $s_j = f(xi, W)_j$. The Multiclass SVM loss for the i -th example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

For example, suppose that we have three classes that receive the scores $s = [13, -7, 11]$, and that the first class is the true class (i.e. $y_i = 0$). Take $\Delta = 10$. The expression above sums over all incorrect classes ($j \neq y_i$), so we get two terms:

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

The first term gives zero since $[-7 - 13 + 10]$ gives a negative number, which is then thresholded to zero with the $\max()$ function. We get zero loss for this pair because the correct class score (13) was greater than the incorrect class score (-7) by at least the margin of 10.

In summary, the SVM loss function wants the score of the correct class y_i to be larger than the incorrect class scores by at least by Δ . If this is not the case, we will accumulate loss.

2.2.2 Cross-entropy loss (SoftMax classifier)

The SoftMax classifier uses a different loss function- namely, the cross-entropy loss. Unlike the SVM which treats the outputs $f(xi, W)$ as (uncalibrated and possibly difficult to interpret) scores for each class, the Softmax classifier gives a slightly more intuitive output (normalized class probabilities) and also has a probabilistic interpretation. Instead of interpreting each row of $f(xi, W)$ as score for that particular class, we interpret it as the probability of the image belonging to that class. The cross-entropy loss has the form –

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j f_j}\right),$$

$$\text{or } L_i = -f_{y_i} + \log \sum_j f_j$$

where we are using the notation f_j to mean the j -th element of the vector of class scores f . As before, the full loss for the dataset is the mean of L_i over all training examples together with a regularization term $R(W)$ (please see next section). The function $f_j(z) = e^{z_j} / \sum_k e^{z_k}$ is called the SoftMax function.

2.2.3 SVM vs SoftMax

Unlike the SVM which computes uncalibrated and not easy to interpret scores for all classes, the SoftMax classifier allows us to compute “probabilities” for all labels. For example, given an image the SVM classifier might give us scores [12.5, 0.6, -23.0] for the classes “cat”, “house” and “dog”. The SoftMax classifier can instead compute the probabilities of the three labels as [0.9, 0.09, 0.01], which allows us to interpret its confidence in each class. In practice, the performance difference between the SVM and SoftMax are usually very small. The SVM does not care about the details of the individual scores: if they were instead [10, -100, -100] or [10, 9, 9] the SVM would be indifferent since the margin of delta = 1 is satisfied and hence the loss is zero. However, these scenarios are not equivalent to a SoftMax classifier, which would accumulate a much higher loss for the scores [10, 9, 9] than for [10, -100, -100]. In other words, the Softmax classifier is never fully happy with the scores it produces: the correct class could always have a higher probability and the incorrect classes always a lower probability and the loss would always get better. However, the SVM is happy once the margins are satisfied and it does not micromanage the exact scores beyond this constraint. This can be thought of as a feature: For example, a ‘dog classifier’ should be spending most of its “effort” on the difficult problem of classifying different breeds of dogs, and should completely ignore the cat examples, which it already assigns very low scores to, and which likely cluster around a completely different side of the data cloud.

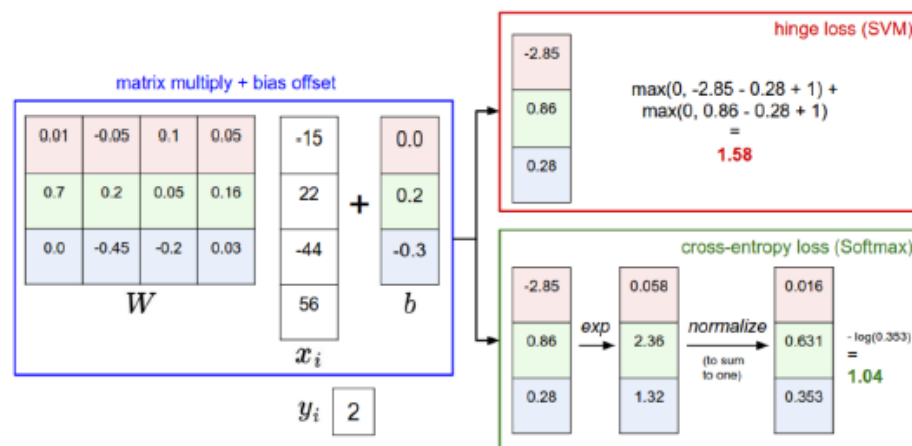


Fig 2.4: Score function comparison of Softmax and SVM (csn231.github.io)

Regularization

Suppose that we have a dataset and a set of parameters W that correctly classify every example. The issue is that this set of W is not necessarily unique: there might be many similar W that correctly classify the examples. Different values of W don't affect the loss function -

the loss function yields a score of zero for all W . In such situations, we must have some criteria to choose from the given set of W . This can be achieved by extending the loss function with a regularization penalty $R(W)$. The most common regularization penalty is the $L2$ norm that discourages large weights through an element-wise quadratic penalty over all parameters:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

In the expression above, we are summing up all the squared elements of W .

For example, suppose that we have some input image vector $x=[1,1,1,1]$ and two weight vectors $w1=[1,0,0,0]$ and $w2=[0.25,0.25,0.25,0.25]$. Let's ignore the bias for the sake of simplicity.

Then $w1^T x = w2^T x = 1$ so both weight vectors lead to the same dot product, but the $L2$ penalty of $w1$ is 1.0 while the $L2$ penalty of $w2$ is only 0.25. Therefore, according to the $L2$ penalty the weight vector $w2$ would be preferred since it achieves a lower regularization loss. Intuitively, this is because the weights in $w2$ are smaller and more diffuse. Since the $L2$ penalty prefers smaller and more diffuse weight vectors, the final classifier is encouraged to take into account all input dimensions to small amounts rather than a few input dimensions and very strongly. As we will see later in the class, this effect can improve the generalization performance of the classifiers on test images and lead to less overfitting.

Optimization (Stochastic Gradient Descent)

SVM loss function for a single data point:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

We can differentiate the function with respect to the weights. For example, taking the gradient with respect to w_{y_i} we obtain:

$$\nabla_{w_{y_i}} L_i = - \left(\sum_{j \neq y_i} (w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

where 1 is the indicator function that is one if the condition inside is true or zero otherwise. This means that we simply count the number of classes that didn't meet the desired margin (and hence contributed to the loss function) and then scale the data vector x_i by this number. This gives us a way to calculate the gradient analytically. But this is the gradient only with

respect to the row of W that corresponds to the correct class. For the other rows where $j \neq y_i$ the gradient is:

$$\nabla w_j L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0))x_i$$

Once we have calculated the gradient, it is straight-forward to implement the expressions and use them to perform the gradient update:

$$W = W - \mu * \text{gradient},$$

where μ is the step size. [7]

2.3 Convolutional Neural Network

Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Artificial Neural Networks which have proven to be very effective in the field of image recognition and classification. They have been used extensively for the task of object detection, self driving cars, image captioning etc. First convnet was discovered in the year 1990 by Yann Lecun and the architecture of the model was called as the LeNet architecture. A basic convnet is shown in the fig. below [1]

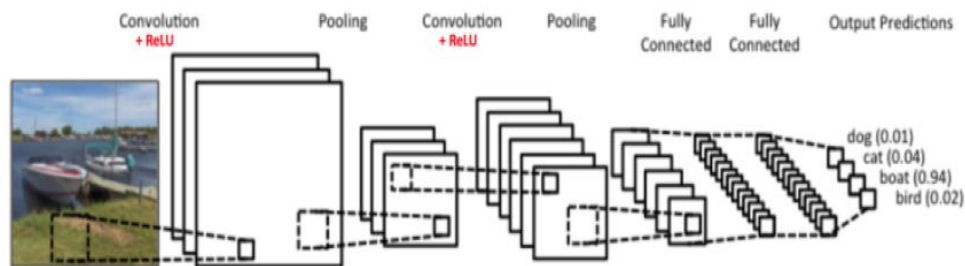


Fig. 2.5: A simple convnet architecture (ujjwalkarn.me)

The entire architecture of a convnet can be explained using four main operations namely,

1. Convolution
2. Non- Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

These operations are the basic building blocks of *every* Convolutional Neural Network, so understanding how these work is an important step to developing a sound understanding of ConvNets. We will discuss each of these operations in detail below.

Essentially, every image can be represented as a matrix of pixel values. An image from a standard digital camera will have three channels – red, green and blue – you can imagine

those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

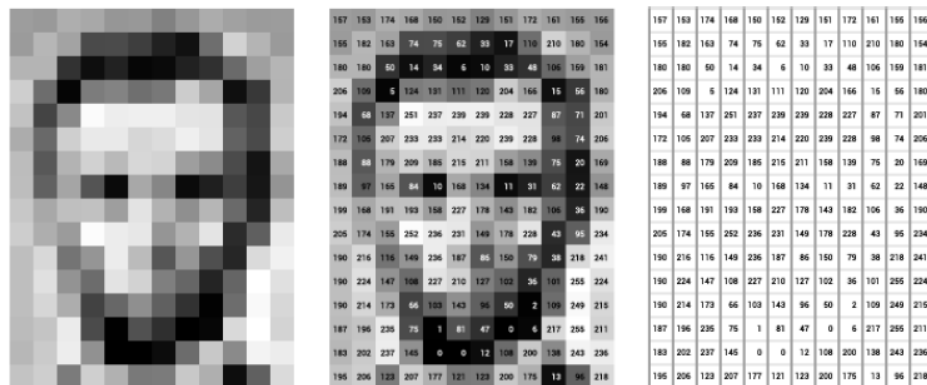


Fig. 2.6: A grayscale image as matrix of numbers Image Captioning (researchgate.net)

Convolution Operator

The purpose of convolution operation is to extract features from an image. We consider filters of size smaller than the dimensions of image. The entire operation of convolution can be understood with the example below.

Consider a small 2-dimensional 5*5 image with binary pixel values. Consider another 3*3 matrix shown in Fig. 2.7.

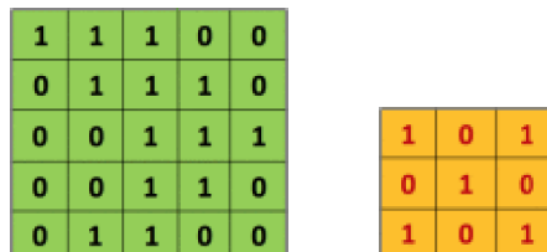


Fig. 2.7: Image (in green) and Filter (in orange) (mc.ai)

We slide this orange 3*3 matrix over the original image by 1 pixel and calculate element-wise multiplication of the orange matrix with the sub-matrix of the original image and add the final multiplication outputs to get the final integer which forms a single element of the output matrix which is shown in the Fig. 2.8 by the pink matrix.

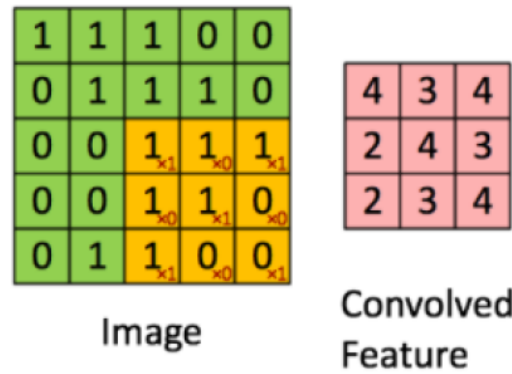


Fig. 2.8: Convolution operation (medium.com)

The 3*3 matrix is called a filter or kernel or feature detector and the matrix formed by sliding the filter over the image and computing the dot product is called the Convolved Feature or Activation Map or the Feature Map. The number of pixels by which we slide the filter over the original image is known as stride.[7]

Introducing Non-Linearity

An additional operation is applied after every convolution operation. The most commonly used non-linear function for images is the ReLU which stands for Rectified Linear Unit. The ReLU operation is an element-wise operation which replaces the negative pixels in the image with a zero. Since most of the operations in real-life relate to non-linear data but the output of convolution operation is linear because the operation applied is elementwise multiplication and addition. The output of the ReLU operation is shown in the figure below.

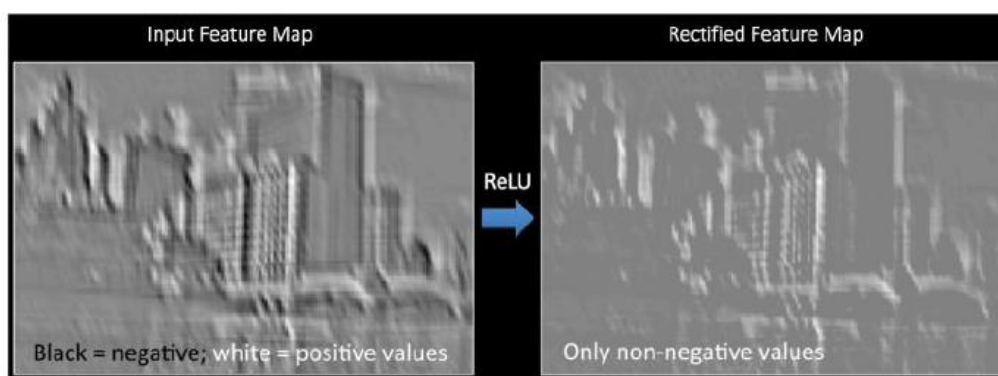


Fig. 2.9: Output after a ReLU operation (kdnuggets.com)

Some other commonly used non-linearity functions are sigmoid and tanh.

Spatial Pooling

The pooling operation reduces the dimensionality of the image but preserves the important features in the image. The most common type of pooling technique used is max pooling. In max pooling you slide a window of $n \times n$ where n is less than the side of the image and determine the maximum in that window and then shift the window with the given stride length. The complete process is specified by the fig.2.10.[1]

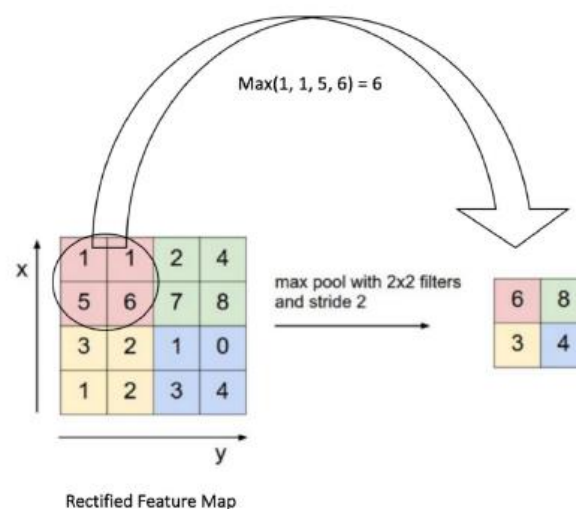


Fig. 2.10: Max pooling operation (ujjwalkarn.me)

Fully-Connected layer

The fully connected layer is the multi-layer perceptron that uses the SoftMax activation function in the output layer. The term “fully-connected” refers to the fact that all the neurons in the previous layer are connected to all the neurons of the next layer. The convolution and pooling operation generate features of an image. The task of the fully connected layer is to map these feature vectors to the classes in the training data. [7]

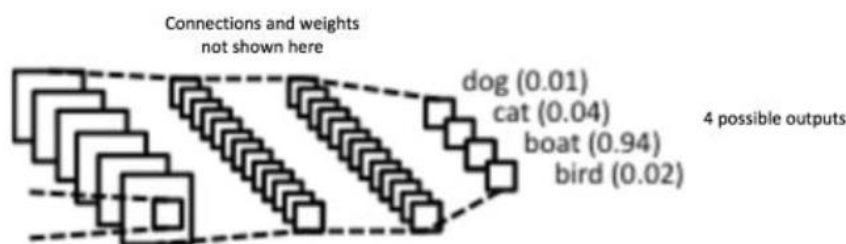


Fig. 2.11: An example of fully connected layer of data with 4 classes (ujjwalkarn.me)

CHAPTER 3- IMPLEMENTATION

3.1 Model Overview

The model proposed takes an image I as input and is trained to maximize the probability of $p(S/I)$ [1] where S is the sequence of words generated from the model and each word S_t is generated from a dictionary built from the training dataset. The input image I is fed into a deep vision Convolutional Neural Network (CNN) which helps in detecting the objects present in the image. The image encodings are passed on to the Language Generating Recurrent Neural Network (RNN) which helps in generating a meaningful sentence for the image as shown in the fig. 3.1. An analogy to the model can be given with a language translation RNN model where we try to maximize the $p(T/S)$ where T is the translation to the sentence S . However, in our model the encoder RNN which helps in transforming an input sentence to a fixed length vector is replaced by a CNN encoder. Recent research has shown that the CNN can easily transform an input image to a vector.

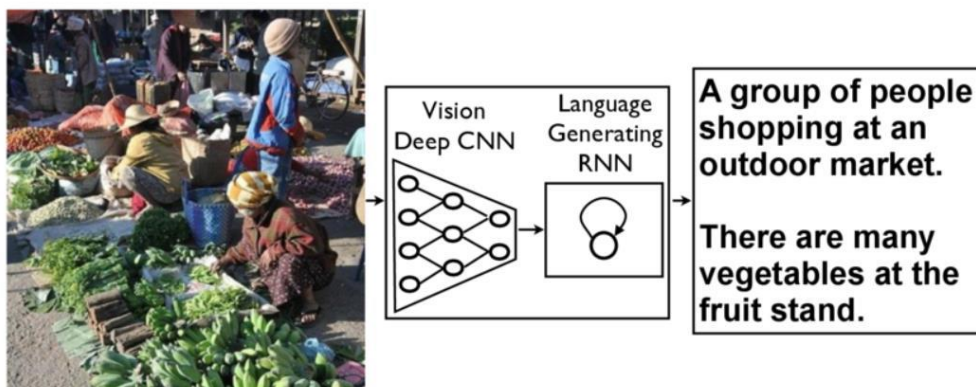


Fig. 3.1: An overview of the image captioning model (medium.com)

For the task of image classification, we use a pretrained model ResNet50. The details of the models are discussed in the following section. A Long Short-Term Memory (LSTM) network follows the pretrained ResNet50 model. The LSTM network is used for language generation. LSTM differs from traditional Neural Networks as a current token is dependent on the previous tokens for a sentence to be meaningful and LSTM networks take this factor into account.

In the following sections we discuss the components of the model i.e. the CNN encoder and the Language generating RNN in details.[1]

3.2 Dataset

The dataset used in this project is the flickr8k dataset, which contains 8000 images with 5 captions per each image, each image has a unique ID and the captions are stored corresponding to the ID. The images were chosen from six different Flickr groups, and tend not to contain any well-known people or locations, but were manually selected to depict a variety of scenes and situations

The dataset contains 6000 images for training of project, 1000 images for development, and 1000 for testing.[4]



Fig. 3.2: An example of a random image in Flickr8k Dataset (Flickr8k dataset from kaggle.com)

3.3 Deep CNN Architecture

The details of the CNN were discussed in section 2.3. Convolutional Neural Network (CNN) have improved the task of image classification significantly. ImageNet Large Scale Visual Recognition competition (ILSVRC) have provided various opensource deep learning frameworks like ZFnet, Alexnet, Vgg16, ResNet etc have shown great potential in the field of image classification. For the task of image encoding in our model we use ResNet50 which is a 150-layered network proposed in ILSVRC 2015. The architecture of ResNet50 has **4 stages** as shown in fig. 3.3. The network can take the input image having height, width as multiples of 32 and 3 as channel width. For the sake of explanation, we will consider the input size as $224 \times 224 \times 3$. Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. Afterward, Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively.

The curved arrows refer to the identity connection. The dashed connected arrow represents that the convolution operation in the Residual Block is performed with stride 2, hence, the size of input will be reduced to half in terms of height and width but the channel width will be doubled. As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half.

For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F , 3 layers are stacked one over the other. The three layers are 1×1 , 3×3 , 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions. Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).[8]

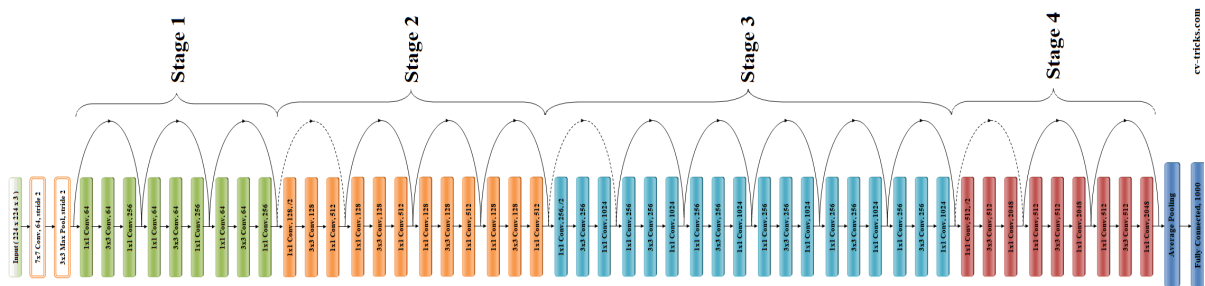


Fig. 3.3: ResNet50 architecture (CV-Tricks.com)

3.4 Recurrent Neural Net (RNN) Decoder Architecture

Recurrent neural nets are a type of artificial neural network in which connection between units form a directed cycle. The advantage of using RNN over conventional feed forward net is that the RNN can process arbitrary set of inputs using its memory. RNNs were discovered in the year 1980 by John Hopfield who gave the famous Hopfield model. Recurrent neural nets in simple terms can be considered as networks with loops which allows the information to persist in the network.

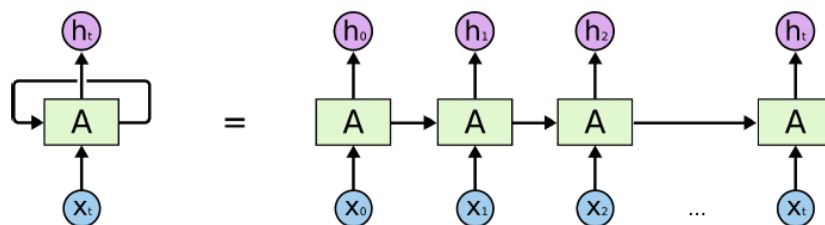


Fig. 3.4: A simple neural network unrolled into simple neural net (colah.github.io)

As shown in the figure above a recurrent neural network can be considered as multiple copies of same network with each network passing the message to its successor.

One of the problems with RNNs is that they do not take long-term dependencies into account. Consider a machine that tries to generate sentences on its own. For instance, the sentence is “I grew up in England, I speak fluent English”, if the machine is trying to predict the last word in the sentence i.e. *English*, the machine needs to know that the language name to be followed by fluent is dependent on the context of the word England. It is possible that the gap between the relevant information and the point where it is needed becomes very large in which case the conventional RNNs fail.

To overcome the above-mentioned problem of “long term dependencies”, Hochreiter and Schmidhuber proposed the Long Short-Term Memory (LSTM) networks in the year 1997. Since then LSTM networks have revolutionized the fields of speech recognition, machine translation etc. Like the conventional RNNs, LSTMs also have a chain like structure, but the repeating modules have a different structure in case of a LSTM network. A simple LSTM network is shown in Fig. 3.5. [7]

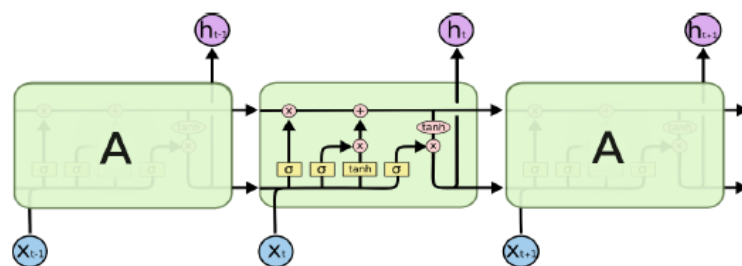


Fig. 3.5: Four interacting layers in a LSTM layer (analyticvidhya.com)

The key behind the LSTM network is the horizontal line running on the top which is known as the cell state. The cell state runs through all the repeating modules and is modified at every module with the help of gates. This causes the information in a LSTM network to persist.

We use this LSTM network with a slight variation. The architecture of the LSTM network used is given below.

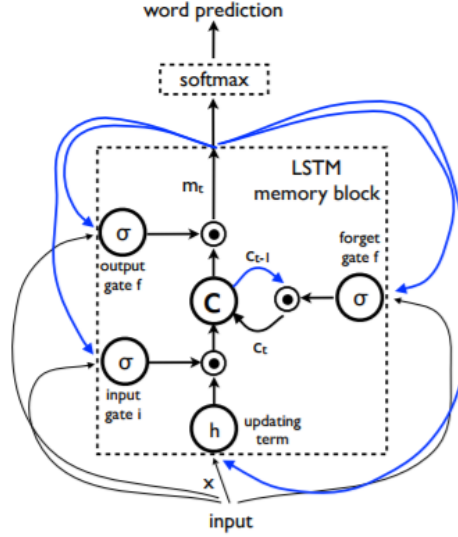


Fig. 3.6: LSTM architecture for language generation (researchgate.net)

The entire network is governed by the following equations

$$it = \sigma(Wixxt + Wimmt - 1),$$

where it is the input gate at time t , W represents the trained parameters. The variable $mt-1$ denotes the output of the module at time $t-1$ and σ represents the sigmoid operation which outputs numbers between zero and one, describing how much of each component should be let through.

$$ft = \sigma(Wfxxt + Wfmmt - 1),$$

where ft represents the forget gate which control whether to forget the current cell value.

$$ot = \sigma(Woxxt + Wommt - 1),$$

where ot represents the output gate which determines whether to output the new cell value or not.

$$ct = ft \odot ct - 1 + it \odot h(Wcixt + Wcmmt - 1),$$

where ct is the cell state that runs through all the modules and \odot represents the tensor product with a gate value.

$$mt = ot \odot ct,$$

where mt is the encoded vector which is then fed into the softmax function.

$$pt+1 = \text{Softmax}(mt)$$

The output $pt+1$ of a module gives the word prediction. The same LSTM network is repeated until an end token (.) is encountered by the network. The series of these word prediction generate the caption for a given image. The complete training process for the

combined model (CNN encoder + RNN language generator) and the LSTM network in unravelled form is given below in Fig. 3.7.

The LSTM model is trained to predict each word of the sentence after it has seen the image as well as all preceding words as defined by $p(S_t|I, S_0, \dots, S_{t-1})$. [1]

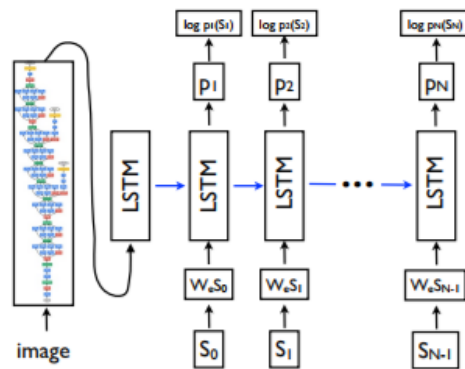

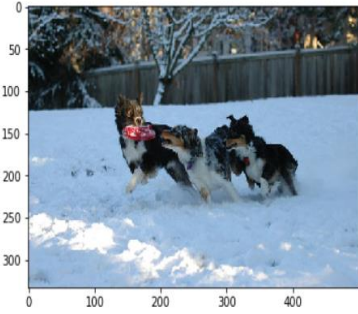


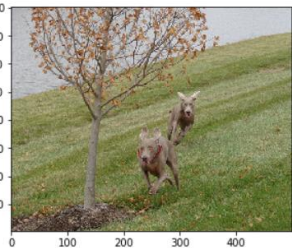



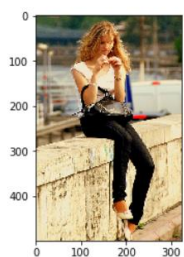
Fig. 3.7: Working of the image captioning model (machinelearningmastery.com)

Chapter 4 – SAMPLE OUTPUTS

The model was able to predict the meaningful captions for an image in most of the cases. However, in some of the cases it got confused due to lack of the tokens in dictionary to define an event. Also, it got confused in the cases where almost the entire image was covered with one colour.

Following are the outputs generated.

 <p>(17102241, 17102228)Output: black dog is carrying ball in its mouth</p>	 <p>(17102241, 17102228)Output: two dogs are playing bend in the snow</p>
 <p>(17102241, 17102228)Output: boy in swimming trunks is swimming in pool</p>	 <p>(17102241, 17102228)Output: two teams are playing soccer</p>
 <p>(17102241, 17102228)Output: two dogs are running through the grass</p>	 <p>(17102241, 17102228)Output: two climbers are scaling up rock face</p>



(17102241, 17102228)Output: the little girl is sitting on the pavement



(17102241, 17102228)Output: group of people are standing in the snow



(17102241, 17102228)Output: group of people are gathered in the street



(17102241, 17102228)Output: little boy in red shirt and jeans is playing on swing



(17102241, 17102228)Output: boy in red shirt is playing soccer



(17102241, 17102228)Output: dog is running through training competition



(17102241, 17102228)Output: two dogs are playing with ball in field



(17102241, 17102228)Output: motorcycle rider is turning corner turn



(17102241, 17102228)Output: black dog is leaping through the water



(17102241, 17102228)Output: man is wakeboarding on his skateboard in the air

Chapter 5 – CONCLUSION

Our end-to-end system neural network system is capable of viewing an image and generating a reasonable description in English depending on the words in its dictionary generated on the basis of tokens in the captions of train images. The model has a convolutional neural network encoder and a LSTM decoder that helps in generation of sentences. The purpose of the model is to maximize the likelihood of the sentence given the image.[5]

Experimenting the model with Flickr8K dataset show decent results. The accuracy can be increased if the same model is worked upon a bigger dataset. Furthermore, it will be interesting to see how one can use unsupervised data, both from images alone and text alone, to improve image description approaches.[3]

Chapter 6 – FUTURE SCOPE

The task of image captioning can be put to great use for the visually impaired. The model proposed can be integrated with an android or ios application to work as a real-time scene descriptor. The accuracy of the model can be improved to achieve state of the art results by hyper tuning the parameters. The model's accuracy can be boosted by deploying it on a larger dataset so that the words in the vocabulary of the model increase significantly. The use of relatively newer architecture, like VGG and GoogleNet can also increase the accuracy in the classification task thus reducing the error rate in the language generation. Apart from that the use of bidirectional LSTM network and Gated Recurrent Unit may help in improving the accuracy of the model.[6]

REFERENCES

1. S. Smys, T. Senjyu, P. Lafata, *Second International Conference on Computer Networks and Communication Technologies: ICCNCT 2019*, Switzerland: Springer Nature, 2020, pp. 327-330.
2. www.keras.com
3. www.github.com
4. www.kaggle.com
5. www.towardsdatascience.com
6. www.machinelearningmastery.com
7. cs231n.github.io
8. CV-Tricks.com