Visual Question Answering

M.Hariharan 16BCE1293 , J.S.Sandeep Kiran 16BCE1041

School of Computing Sciences and Engineering, VIT Chennai, Tamilnadu, India 600127

1.ABSTRACT

Visual Question Answering (VQA) is a challenging task that has received increasing attention from both the computer vision and the natural language processing communities. Given an image and a question in natural language, it requires reasoning over visual elements of the image and general knowledge to infer the correct answer. The aim of our project is to build a Visual Question Answering system that will be able to take in an image as input and a natural language question from the user. Using the inputs, the system will be able to extract word embeddings from the question using Recurrent Neural Network (RNN), extract features from the image using a VGG convolutional neural network (CNN), and combine both the features together using a fully connected feedforward neural network to give the probabilities of the outputs with the highest probabilities. The end system will be a Flask app that will prompt the users for the inputs and display the prediction.

2.INTRODUCTION

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. The ability of humans to look at an object and derive at some sort of answer regarding the object has been long attempted to be replicated in computer systems, but have failed to do so. Currently, due to the advent of deep learning and natural language processing techniques, building a system that would have capabilities to answer such questions are emerging with promising results. In general, Visual Question Answering system can be defined as an algorithm that takes an image as input and a natural language question about the image and generates a natural language answer as the output. It requires techniques that involve both image recognition/analysis and natural language processing.

In Visual QA, the computer system needs to address issues, such as, a binary classification problem (Is the umbrella upside down?), a counting problem (How many children are in the bed?), or an open-ended question (Who is wearing glasses? Where is the child setting?). Also, if the system is presented with a question, it first needs to break down the question into –

1. Classify the question type – how many, is there….
2. Extract the object from the question
3. Extract the context

After the question has been analyzed, the system needs to build a query mechanism that relies on a knowledge base to get the answer. Hence to answer any question, the system also needs to detect the object being referenced in the question. We use image enhancement techniques like sharpening and smoothing, perform histogram analysis to isolate the object on the basis of pixels, and use image segmentation techniques to identify the object that is being referenced in the question. The end result of these two steps need to be combined using a neural network to produce a correlation between the two

features so that we can get the outputs with the highest probabilities as the answer to the question that was asked.

2.1 SCOPE OF THE PROJECT

- Potential application for Visual QA system would be as an assistant to help the blind and visually impaired users. This system could be given a steady feed of images from a webcam at the perspective of the blind person
- A Visual QA system could provide information about any image on the internet or any social media. This could help maintain the website by generating useful metadata about images that would be helpful for bots and AI systems to organize the directories in servers.
- This system can also be used for educational and recreational purposes like integrating this system with an education app for children
- It needs to localize the subject being referenced needs to detect objects (the banana), and should also have some common-sense knowledge that the word mustache is often used to refer to markings or objects on the face that are not actually mustaches (like milk mustaches).

2.2 OBJECTIVES

- The Visual QA system needs to be able to segment images and be able to extract features efficiently from the image using the layers of the neural networks like convolutional layer, max pool layer etc.
- It also needs to extract the word embeddings from the questions and be able to represent it in a 300 dimensional array.
- The system needs to be able to map all our input image features and input question features into a single feature vector of fixed length.
- The system needs to be able to display the possible outputs with the highest probability to the user for the given image and question.
- It needs to localize the subject being referenced needs to detect objects (the banana), and should also have some common-sense knowledge that the word mustache is often used to refer to markings or objects on the face that are not actually mustaches (like milk mustaches).

2.3 SOFTWARE REQUIREMENTS

The dependencies for running the system are –

- **Spacy** – spacy is a python package that excels at large-scale information extraction tasks. It's written from the ground up in carefully memory - managed Cython, and is used for natural language processing. It supports various methods like tokenization, lemmatization, part-of-speech tagging, entity recognition, dependency parsing, sentence recognition, word2vec transformations and other methods to clean and normalize text.

- **NumPy** - NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities.

- **Matplotlib** - Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and

interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells,
the Jupyter notebook. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code.

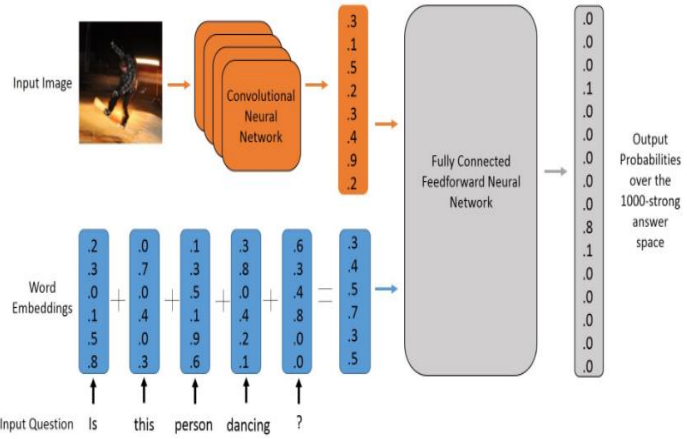
- **PIL** - The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It provides a solid foundation for a general image processing tool.

- **Keras** - Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that allows for easy and fast prototyping (through user friendliness, modularity, and extensibility), supports convolutional networks and recurrent networks, as well as combinations of the two and runs seamlessly on CPU and GPU.

- **TensorFlow** – TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

2.4 MODULES OF THE SYSTEM

1. Region based Image Segmentation using Selective Search
2. Data Augmentation
3. Convolutional Neural Network Design
4. Design of the Recurrent Neural Net for Natural Language processing
5. Fully Connected Layers Design and the Softmax Activation layer



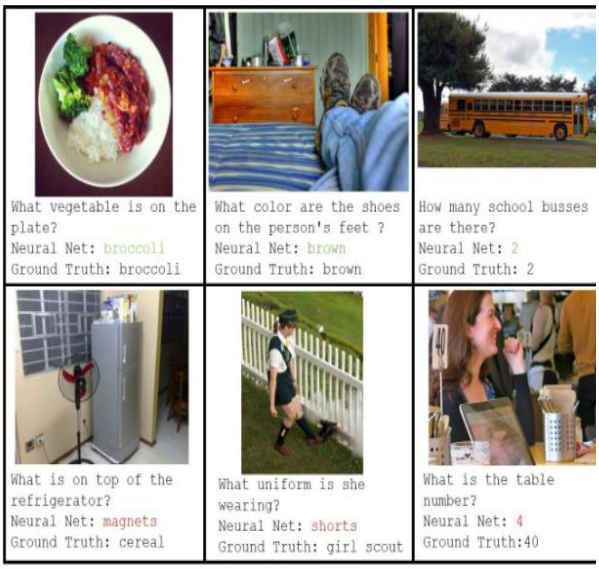
3.RELATED WORK

VQA Efforts: Several recent papers have begun to study visual question answering . However, unlike our work, these are fairly restricted (sometimes synthetic) settings with small datasets. For instance, the Research by M. Malinowski and M. Fritz on ” A Multi-World Approach to Question Answering about Real-World Scenes based on Uncertain Input” only considers questions whose answers come from a predefined closed world of 16 basic colors or 894 object categories. The Research by D. Geman, S. Geman, N. Hallonquist, and L. Younes on “A Visual Turing Test for Computer Vision Systems” also considers questions generated from templates from a fixed vocabulary of objects, attributes, relationships between objects, etc. In contrast, our proposed task involves openended, free-form questions and answers provided by humans. Our goal is to increase the diversity of knowledge and kinds of

reasoning needed to provide correct answers. Text.One key concern in text is the grounding of questions VQA is naturally grounded in images – requiring the understanding of both text (questions) and vision (images). Our questions are generated by humans, making the need for commonsense knowledge and complex reasoning more essential.

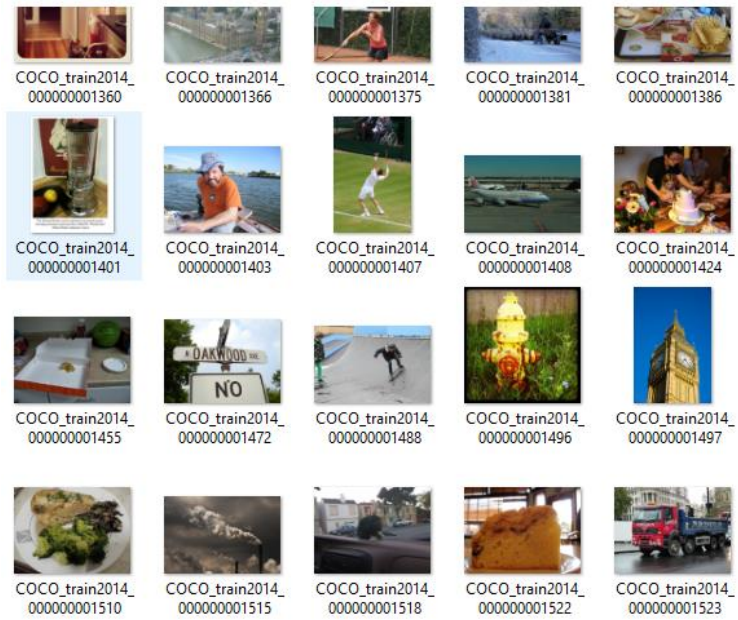
4.VQA DATASET

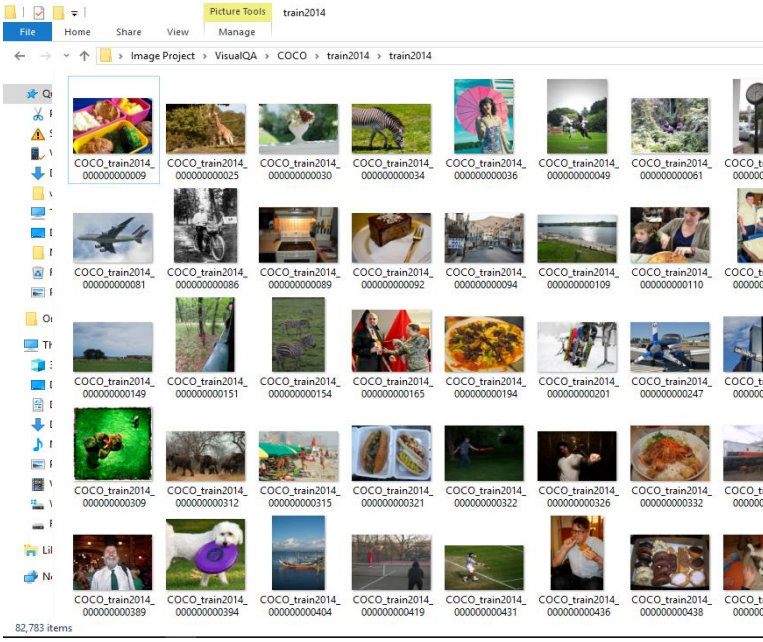
We have utilized the VQA dataset for training our neural networks, which is built on MS COCO dataset of real life pictures and scenarios. VQA is a new dataset containing open-ended questions about images. It contains 2,65,016 images, and has at least 3 questions (5.4 questions on average) per image. There are 10 ground truth answers per question and 3 plausible (but likely incorrect) answers per question. In addition to 204,721 images from the COCO dataset, it includes 50,000 abstract cartoon images. There are three questions per image and ten answers per question, which is over 760K questions with around 10M answers.



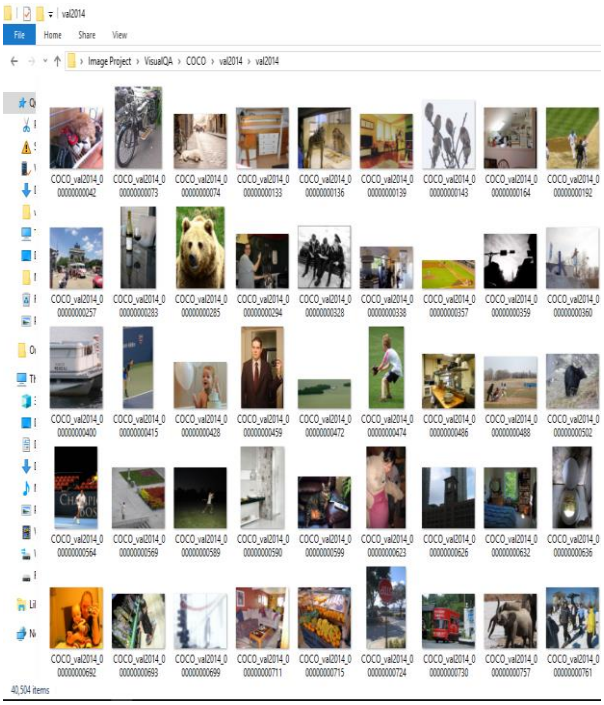
The dataset is has three sections – the images, the questions and the answers. They are further classified into training, validation and testing sections.

Train images – 82,783 images

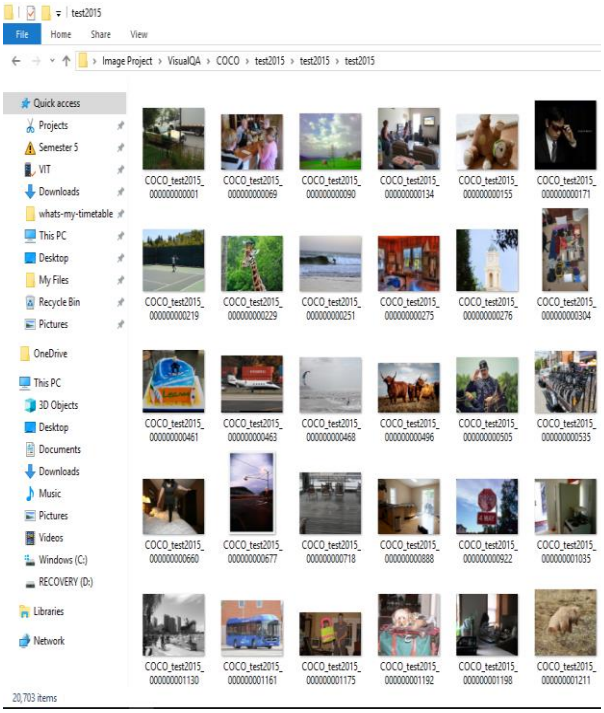




Validation images – 40,504 images



Test images - 40,504 images



The questions are stored in JSON file format and are of the structure -

```
{
"info":info,
"task_type":str,
"data_type":str,
"data_subtype":str,
"questions":[question],
"license":license
}

question{
"question_id":int,
"image_id":int,
"question":str
}
```

The annotations are also stored in JSON file format and are in the format –

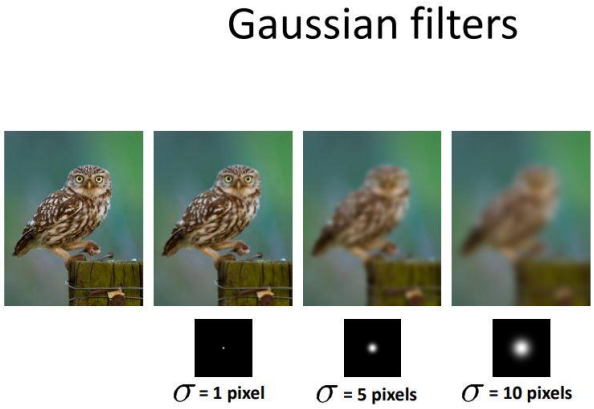
```
{
"info"          :          info,
"data_type":               str,
"data_subtype":            str,
"annotations"   :   [annotation],
"license"       :       license
}

annotation{
"question_id"          :          int,
"image_id"             :          int,
"question_type"        :          str,
"answer_type"          :          str,
"answers"              :     [answer],
"multiple_choice_answer"   :      str
}
```

5.IMPLEMENTATION

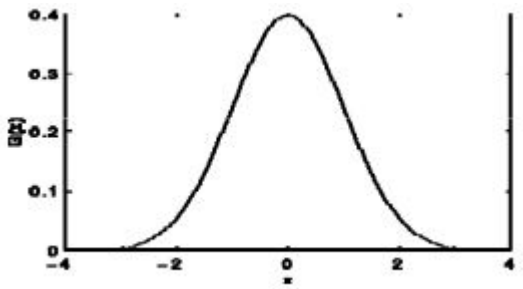
5.1 DATA AUGMENTATION AND SELECTIVE SEARCH

The first step in image processing is image enhancement. So, one major part of the analysis is to identify suitable techniques to cleanse the image and make it suitable for CNN's feature extraction

Blurring: Using Gaussian filters



A Gaussian filter looks like the below.



When working with images we need to use the two dimensional Gaussian function.

This is simply the product of two 1D Gaussian functions (one for each direction) and is given by:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

A Gaussian filter is a linear filter. It's usually used to blur the image or to reduce noise. If you use two of them and subtract, you can use them for "unsharp masking" (edge detection). The Gaussian filter alone will blur edges and reduce contrast.

The Median filter is a non-linear filter that is most commonly used as a simple way to reduce noise in an image. It's claim to fame (over Gaussian for noise reduction) is that it removes noise while keeping edges relatively sharp.

One advantage a Gaussian filter has over a median filter is that it's faster because multiplying and adding is probably faster than sorting.

```
In [28]: blurred_img = ndimage.gaussian_filter(img, sigma=1.5)
         plt.imshow(blurred_img)

Out[28]: <matplotlib.image.AxesImage at 0x22ef85b67f0>
```

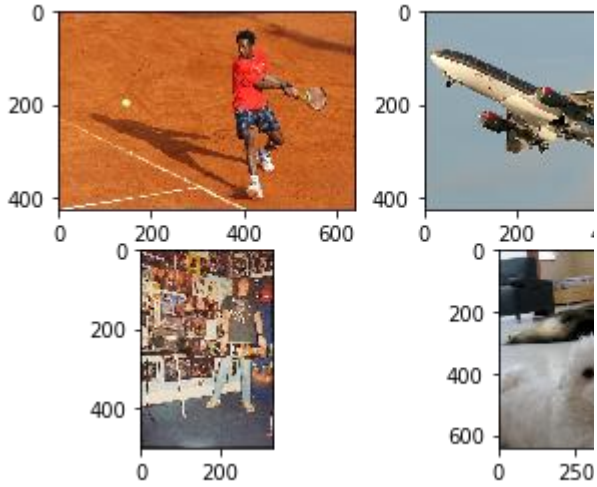
This is followed by **Segmentation. We chose a mixture of Edge based and region based techniques to see what was the learning model's response to each of these segmented <image, question> pairs.**

Thresholding Histogram

Most of our COCO images contain foreground objects, that are quite distinctive against the background. Because, these are real life scenes and every snap of our life

has a backdrop and a background story for coffee :)

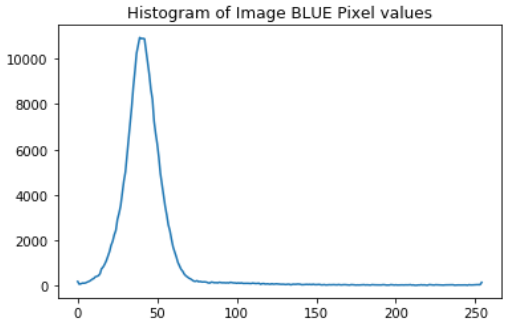
But heres' the glitch. Not all foreground objects have the same color.

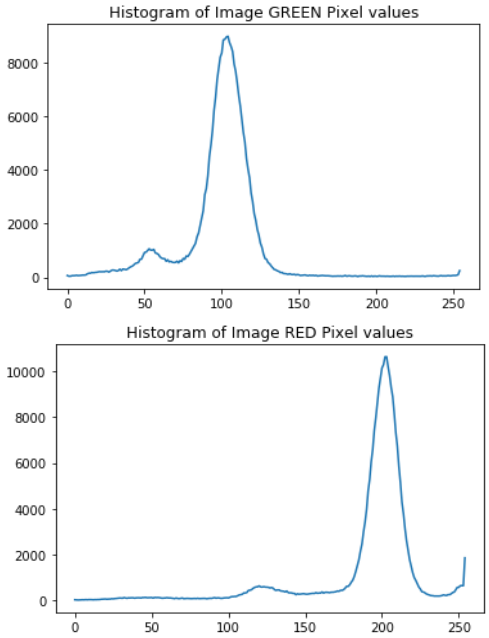


The first 2 images are not that clear. Because, its just one object in the foreground and may give us just 2 uniform regions in the image.

The next 2 images, have a lot of objects in forground which are quite different in color. Even then, proper choice of threshold in a multi-modal histogram can still split apart in binary 0/1 for each of R G B

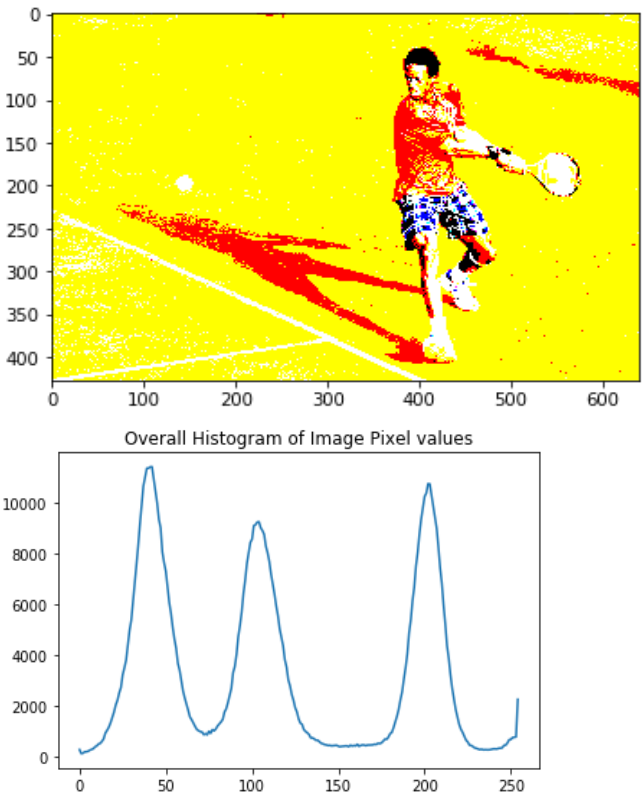
Let’s take the first one of these 4 images and threshold it against the background.





Based on the Thresholding technique, we can now segment the image on each of these local maximas, i.e) on either side of the local maxima, we get 2 regions each of which show different pixel intensities.

Using this idea, after segmenting, we get,



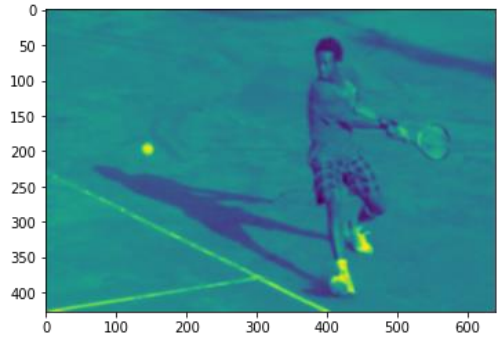
Edge based Segmentation: The Canny Edge Detector

The idea is to take advantage of the local contrast, that is, to use the gradients rather than the RGB values.

The two key parameters of the algorithm are - an upper threshold and a lower threshold. The upper threshold is used to mark edges that are definitely edges. The lower threshold is to find faint pixels that are actually a part of an edge.

The canny edge detector is a multistage edge detection algorithm. The steps are:

(i) Preprocessing: Applying the Gaussian filter. We used a 5*5 Gaussian filter on each one of the input training images.



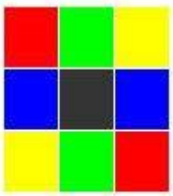
(ii). Calculating gradients: The standard sobel edge detector is used. $m = \sqrt{G_x^2 + G_y^2}$.

The magnitude of gradient is The direction of gradient is $\theta = arctan(\frac{G_y}{G_x})$.

Here, Gx and Gy are the X and Y derivatives at the point being considered.

(iii). Nonmaximum suppression: This step does exactly what it means - if a pixel is not a maximum, it is suppressed. To do this, you iterate over all pixels. The orientation of each pixel is put into one of the four bins. Why? Have a look at this:

The neighbors while computing canny edges



Possible directions of an edge

Let's say you're at the grey pixel. There are only four possible edges possible - it could either go from north to south (the green neighbors), from east to west (the blue ones) or one of the diagonals (the yellow or red neighbors). So using the current pixel's gradient direction, you try and estimate where the edge is going.

(iv). Thresholding with hysterysis

a. If the current pixel is not an edge, check the next one.

b. If it is an edge, check the two pixels in the direction of the edge (ie, perpendicular to the gradient direction).

1. If either of them (or both) have the direction in the same bin as the central pixel

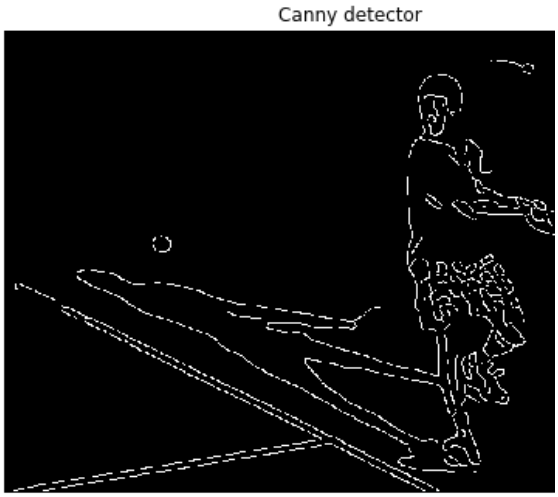
2. And their Gradient magnitude is greater than the lower threshold

3. They are the maximum compared to their neighbors (nonmaximum suppression for these pixels), then

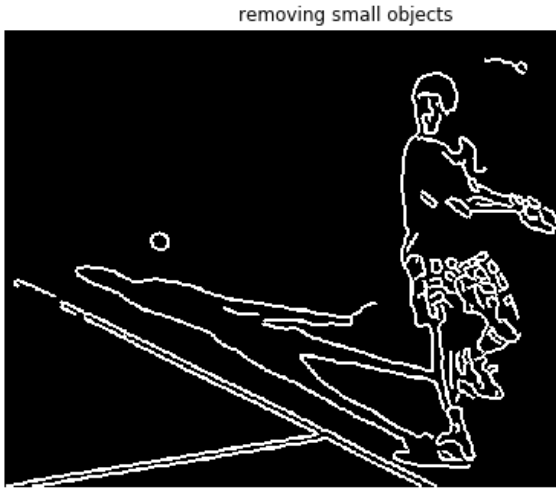
We can mark these pixels as an edge pixel

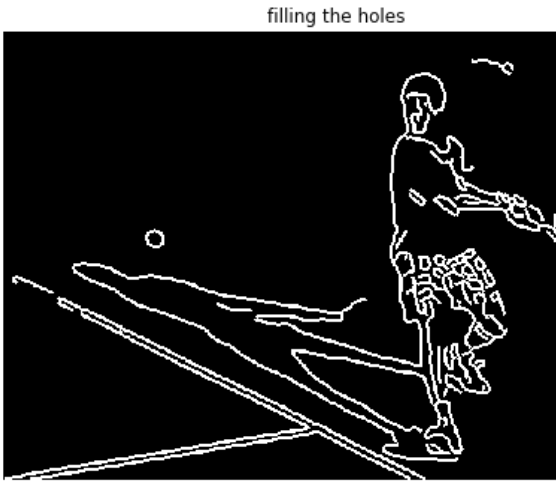
c. Loop until there are no changes in the image

After applying **Canny process**,



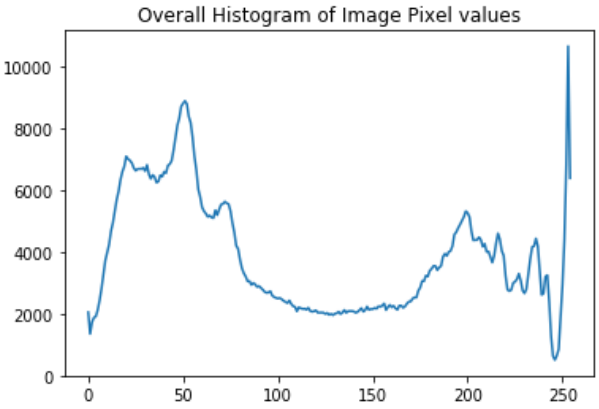
Morphological Operations: Using dilation to fill the holes, followed by erosion





Next up, we use **Watershed algorithm** to combine the results of thresholding along with the detected edges and segment into regions based on that. It means to use the edges as barriers or troughs in the image(to show a separation between 2 objects or regions) and the regions obtained from thresholding as valleys (or regions that represent these boundaries) and then use them as boundaries to contain the regions.

Here's an illustration of this flow.

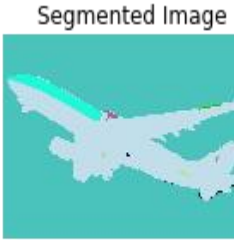


The final segmented image:

Selective Search: Another clustering based region segmentation we used to quickly identify only the regions of interest (ROI).

It uses Felsenszwalb's efficient graph based image segmentation technique and assigns weights to the connectivity(edges) of each of the pixels(represented as vertices in a graph). Closer pixels have small weights connecting them and fall in the same region.

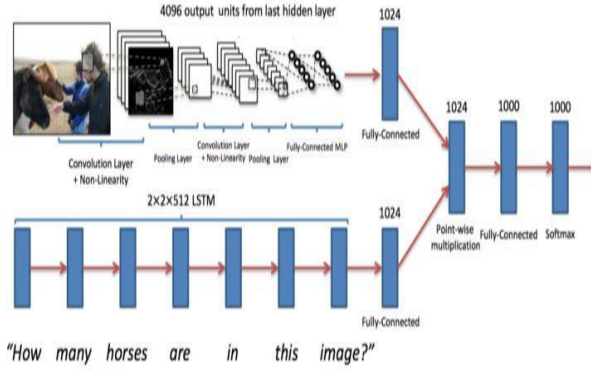
Here's a sample of images from MS COCO we segmented using Selective Search.



Data Augmentation: Is a simple trick used by most researchers to make the CNN learn that, different orientations of the same image doesn't change the classifier output. So different translational, rotational, cropping tricks are applied on the image and the output of these transformations are included in the training set. Here's an example.

5.2 NEURAL NETWORK ARCHITECTURE AT THE INTERSECTION OF NLP AND COMPUTER VISION

Because Visual Ques Answering requires techniques involving image recognition and natural language processing, one major direction in research is on Deep Learning: using Convolutional Neural Network (CNN) for image recognition, using Recurrent Neural Network (RNN) for natural language processing, and then combining the results to deliver the final answer as shown here.



We used the ResNet50 architecture to train extract the image features. Many people at the VQA competition so far, have used VGGnet successfully. We also present their results here. For reference, here is the VGG 16 performance on ILSVRC-2012

Model	top-5 classification error on ILSVRC-201	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

Extracting image features involves, taking a raw image, and running it through the model, until we reach the last layer.

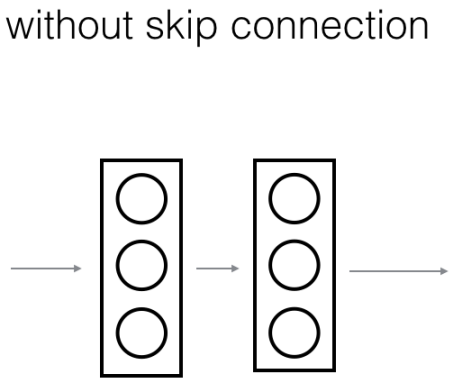
VGG Uniqueness: It has 138 million parameters. So it can fit any complex mapping from the images onto the classifier target. Also, there is a uniformity in the pattern of stacking layers. Some CONV layers, followed immediately by POOL layers (max or avg). The pattern where, the depth of the channels increases and the x-y dims decrease as we add layers, was instrumental in ensuring proper knowledge transfer across layers and learning robust weights. This inspired data scientists to do lot more research in the then left-off CNN. In this case our model is not the VGG Net, but the ResNet50. We present the reasons on the adoption of resnet arch (by Google) in the next few sections.

5.3 IMAGE FEATURE EXTRACTION USING OUR OWN RESNET ARCHITECTURE

The main benefit of a very deep network is that it can represent very complex functions. It can also learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers). However, using a deeper network doesn't always help. A huge barrier to training them is vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow. More specifically, during gradient descent, as you backprop from the final layer back to the first layer, you are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and "explode" to take very large values).

5.4 BUILDING OUR OWN RESIDUAL NETWORK

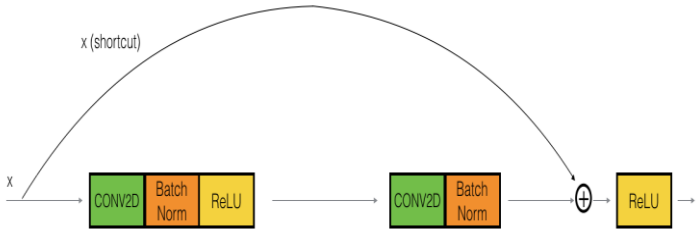
In ResNets, a "shortcut" or a "skip connection" allows the gradient to be directly backpropagated to earlier layers:



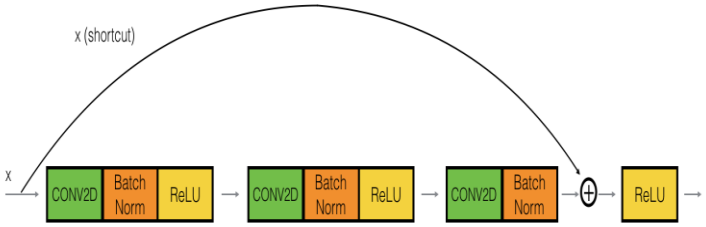
The image on the left shows the "main path" through the network. The image on the right adds a shortcut to the main path. By stacking these ResNet blocks on top of each other, you can form a very deep network.

Having ResNet blocks with the shortcut also makes it very easy for one of the blocks to learn an identity function. This means that you can stack on additional ResNet blocks with little risk of harming training set performance. (There is also some evidence that the ease of learning an identity function--even more than skip connections helping with vanishing gradients--accounts for ResNets' remarkable performance.) Two main types of blocks are used in a ResNet, depending mainly on whether the input/output dimensions are same or different. You are going to implement both of them.

The identity block is the standard block used in ResNets, and corresponds to the case where the input activation (say $a^{[l]}$) has the same dimension as the output activation (say $a^{[l+2]}$). To flesh out the different steps of what happens in a ResNet's identity block, here is an alternative diagram showing the individual steps:



The upper path is the "shortcut path." The lower path is the "main path." In this diagram, we have also made explicit the CONV2D and ReLU steps in each layer. To speed up training we have also added a BatchNorm step. We actually implemented a slightly more powerful version of this identity block, in which the skip connection "skips over" 3 hidden layers rather than 2 layers. It looks like this:



Here're the individual steps.

First component of main path:

- The first CONV2D has F_1 filters of shape (1,1) and a stride of (1,1). Its padding is "valid" and its name should be `conv_name_base + '2a'`. Use 0 as the seed for the random initialization.
- The first BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2a'`.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

Second component of main path:

- The second CONV2D has F_2 filters of shape (f,f) and a stride of (1,1). Its padding is "same" and its name should be `conv_name_base + '2b'`.

Use 0 as the seed for the random initialization.

- The second BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2b'`.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

Third component of main path:

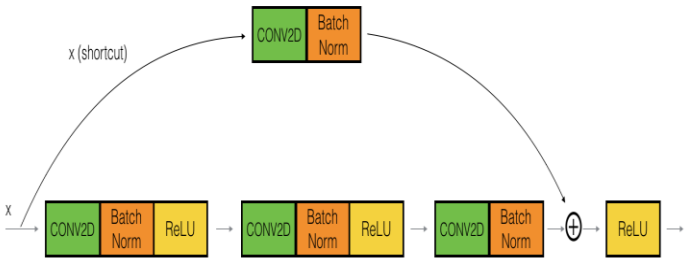
- The third CONV2D has F3 filters of shape (1,1) and a stride of (1,1). Its padding is "valid" and its name should be `conv_name_base + '2c'`. Use 0 as the seed for the random initialization.
- The third BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2c'`. Note that there is no ReLU activation function in this component.

Final step:

- The shortcut and the input are added together.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

5.5 THE CONVOLUTIONAL BLOCK

Now that we have implemented the ResNet identity block. Next, the ResNet "convolutional block" is the other type of block. You can use this type of block when the input and output dimensions don't match up. The difference with the identity block is that there is a CONV2D layer in the shortcut path:



The CONV2D layer in the shortcut path is used to resize the input x to a different dimension, so that the dimensions match up in the final addition needed to add the shortcut value back to the main path. (This plays a similar role as the matrix W_s discussed in lecture.) For example, to reduce the activation dimensions's height and width by a factor of 2, you can use a 1x1 convolution with a stride of 2. The CONV2D layer on the shortcut path does not use any non-linear activation function. Its main role is to just apply a (learned) linear function that reduces the dimension of the input, so that the dimensions match up for the later addition step.

The details of the convolutional block are as follows.

First component of main path:

- The first CONV2D has F1 filters of shape (1,1) and a stride of (s,s). Its padding is "valid" and its name should be `conv_name_base + '2a'`.
- The first BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2a'`.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

Second component of main path:

- The second CONV2D has F2 filters of (f,f) and a stride of (1,1). Its padding is "same" and it's name should be `conv_name_base + '2b'`.

- The second BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2b'`.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

Third component of main path:

- The third CONV2D has F3 filters of (1,1) and a stride of (1,1). Its padding is "valid" and it's name should be `conv_name_base + '2c'`.
- The third BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '2c'`. Note that there is no ReLU activation function in this component.

Shortcut path:

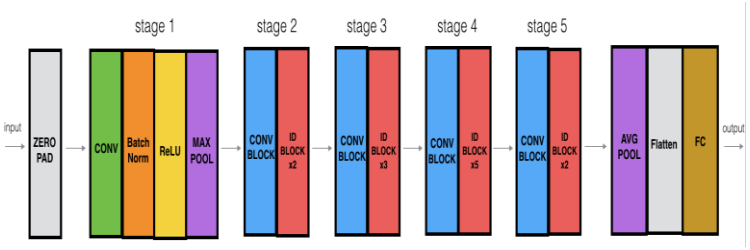
- The CONV2D has F3 filters of shape (1,1) and a stride of (s,s). Its padding is "valid" and its name should be `conv_name_base + '1'`.
- The BatchNorm is normalizing the channels axis. Its name should be `bn_name_base + '1'`.

Final step:

- The shortcut and the main path values are added together.
- Then apply the ReLU activation function. This has no name and no hyperparameters.

F. BUILDING THE RESNET MODEL

We now have the necessary blocks to build a very deep ResNet. The following figure describes in detail the architecture of this neural network. "ID BLOCK" in the diagram stands for "Identity block," and "ID BLOCK x3" means we should stack 3 identity blocks together



The details of this ResNet-50 model are:

- Zero-padding pads the input with a pad of (3,3)

- **Stage 1:**
  - The 2D Convolution has 64 filters of shape (7,7) and uses a stride of (2,2). Its name is "conv1".
  - BatchNorm is applied to the channels axis of the input.
  - MaxPooling uses a (3,3) window and a (2,2) stride.

Stage 2:

- The convolutional block uses three set of filters of size [64,64,256], "f" is 3, "s" is 1 and the block is "a".
- The 2 identity blocks use three set of filters of size [64,64,256], "f" is 3 and the blocks are "b" and "c".

Stage 3:

- The convolutional block uses three set of filters of size [128,128,512], "f" is 3, "s" is 2 and the block is "a".
- The 3 identity blocks use three set of filters of size [128,128,512], "f" is 3 and the blocks are "b", "c" and "d".

Stage 4:

- The convolutional block uses three set of filters of size [256, 256, 1024], "f" is 3, "s" is 2 and the block is "a".

- The 5 identity blocks use three set of filters of size [256, 256, 1024], "f" is 3 and the blocks are "b", "c", "d", "e" and "f".

Stage 5:

- The convolutional block uses three set of filters of size [512, 512, 2048], "f" is 3, "s" is 2 and the block is "a".
- The 2 identity blocks use three set of filters of size [512, 512, 2048], "f" is 3 and the blocks are "b" and "c".

The 2D Average Pooling uses a window of shape (2,2) and its name is "avg_pool".

The flatten doesn't have any hyperparameters or name.

The Fully Connected (Dense) layer reduces its input to the number of classes using a softmax activation. Its name should be `'fc' + classes.

5.6 TRAINING THE MODEL

We ran the following cell to train your model on 500 epochs with a batch size of 1024. On a CPU it should take you around 5min per epoch. But we used the GPU at the lab to train this. ResNet50 is a powerful model for image classification when it is trained for an adequate number of iterations. We can apply it to our own classification problem to perform state-of-the-art accuracy.

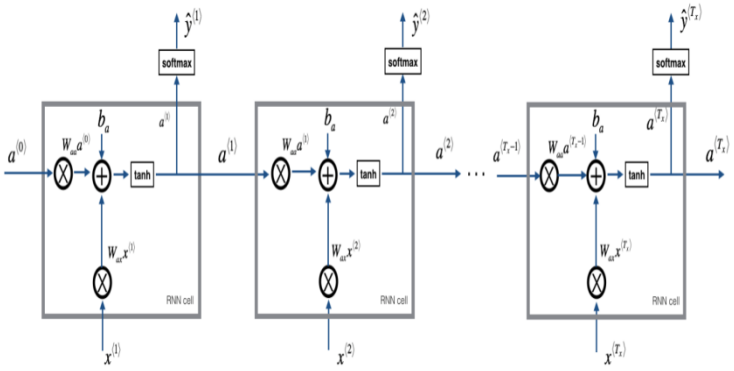
5.7 LONG SHORT TERM MEMORY NETWORKS AND NLP – FORWARD PROPAGATION FOR BASIC RNN AND LTSM

Steps:

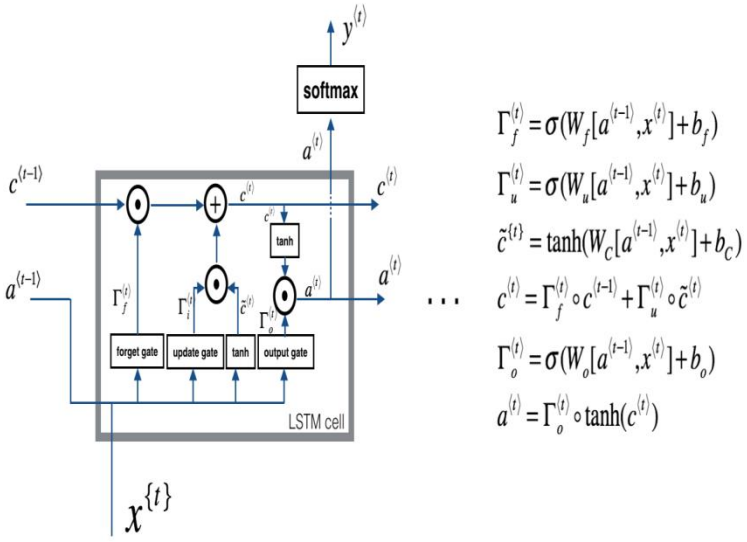
- Implement the calculations needed for one time-step of the RNN.
- Implement a loop over T_x time-steps in order to process all the inputs, one at a time.

RNN CELLS

A Recurrent neural network can be seen as the repetition of a single cell. You are first going to implement the computations for a single time-step. The following figure describes the operations for a single time-step of an RNN cell.



LTSM CELLS



$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$
$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$
$$\tilde{c}^{\{t\}} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$$
$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} \circ \tilde{c}^{\langle t \rangle}$$
$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$
$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} \circ \tanh(c^{\langle t \rangle})$$

About the gates

Forget gate

For the sake of this illustration, lets assume we are reading words in a piece of text, and want use an LSTM to keep track of grammatical structures, such as whether the subject is singular or plural. If the subject changes from a singular word to a plural word, we need to find a way to get rid of our previously stored memory value of the singular/plural state. In an LSTM, the forget gate lets us do this:

$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$

Here, Wf are weight that govern forget gate's behavior. We concatenate [a⟨t−1⟩,x⟨t⟩] and multiply by Wf. The equation above results in a vector Γ⟨t⟩f with values between 0 and 1. This forget gate vector will be multiplied element-wise by the previous cell state c⟨t−1⟩ So if one of the values of Γ⟨t⟩f is 0 (or close to 0) then it means that the LSTM should remove that piece of information in the corresponding component of c⟨t−1⟩ If one of the values is 1, then it will keep the information

Update gate

Once we forget that the subject being discussed is singular, we need to find a way to update it to reflect that the new subject is now plural. Here is the formulat for the update gate:

$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\{t\}}] + b_u)$$

Similar to the forget gate, here Γ⟨t⟩u is again a vector of values between 0 and 1. This will be multiplied element-wise with c̃ ⟨t⟩ in order to compute c⟨t⟩

Updating the cell

To update the new subject we need to create a new vector of numbers that we can add to our previous cell state. The equation we use is:

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$$

Finally, the new cell state is:

$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} * c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} * \tilde{c}^{\langle t \rangle}$$

Output gate

To decide which outputs we will use, we will use the following two formulas

$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o) \quad (5)$$

$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} * \tanh(c^{\langle t \rangle}) \quad (6)$$

Where in equation 5 you decide what to output using a sigmoid function and in equation 6 you multiply that by the tanh of the previous state.

LTSM BACKWARD PASS

One Step backward

The LSTM backward pass is slighltly more complicated than the forward one. We have provided with all the equations for the LSTM backward pass below.

Gate derivatives

$$d\Gamma_o^{\langle t \rangle} = da_{next} * \tanh(c_{next}) * \Gamma_o^{\langle t \rangle} * (1 - \Gamma_o^{\langle t \rangle})$$

$$d\tilde{c}^{\langle t \rangle} = dc_{next} * \Gamma_u^{\langle t \rangle} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * i_t * da_{next} * \tilde{c}^{\langle t \rangle} * (1 - \tanh(\tilde{c})^2)$$

$$d\Gamma_u^{\langle t \rangle} = dc_{next} * \tilde{c}^{\langle t \rangle} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * \tilde{c}^{\langle t \rangle} * da_{next} * \Gamma_u^{\langle t \rangle} * (1 - \Gamma_u^{\langle t \rangle})$$

$$d\Gamma_f^{\langle t \rangle} = dc_{next} * \tilde{c}_{prev} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * c_{prev} * da_{next} * \Gamma_f^{\langle t \rangle} * (1 - \Gamma_f^{\langle t \rangle})$$

Parameter derivatives

$$dW_f = d\Gamma_f^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \tag{11}$$

$$dW_u = d\Gamma_u^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \tag{12}$$

$$dW_c = d\tilde{c}^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \tag{13}$$

$$dW_o = d\Gamma_o^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \tag{14}$$

To calculate dbf,dbu,dbc,dbo you just need to sum across the horizontal (axis= 1) axis on dΓ⟨t⟩f,dΓ⟨t⟩u,dc̃ ⟨t⟩,dΓ⟨t⟩ respectively. Note that you should have the keep_dims = True option.

Finally, you will compute the derivative with respect to the previous hidden state, previous memory state, and input

$$da_{prev} = W_f^T * d\Gamma_f^{\langle t \rangle} + W_u^T * d\Gamma_u^{\langle t \rangle} + W_c^T * d\tilde{c}^{\langle t \rangle} + W_0^T * d\Gamma_0^{\langle t \rangle}$$

Here, the weights for equations 13 are the first n_a, (i.e. Wf=Wf[:na,:] etc...)

$$dc_{prev} = dc_{next}\Gamma_f^{\langle t \rangle} + \Gamma_o^{\langle t \rangle} * (1 - \tanh(c_{next})^2) * \Gamma_f^{\langle t \rangle} * da_{next}$$
$$dx^{\langle t \rangle} = W_f^T * d\Gamma_f^{\langle t \rangle} + W_u^T * d\Gamma_u^{\langle t \rangle} + W_c^T * d\tilde{c}_t + W_o^T * d\Gamma_o^{\langle t \rangle}$$

where the weights for equation 15 are from n_a to the end, (i.e. Wf=Wf[na:,:] etc...)

Training the Model

Training the model with 50 epochs – Just an example screenshot taken on our local sys. The real training process took so much time ~ 2.5 days = 60 hours in the GPU lab system for 5000 epochs.

```
vqa_mod.fit([question_feat, image_feat], answersY, epochs=50, verbose=2)
```

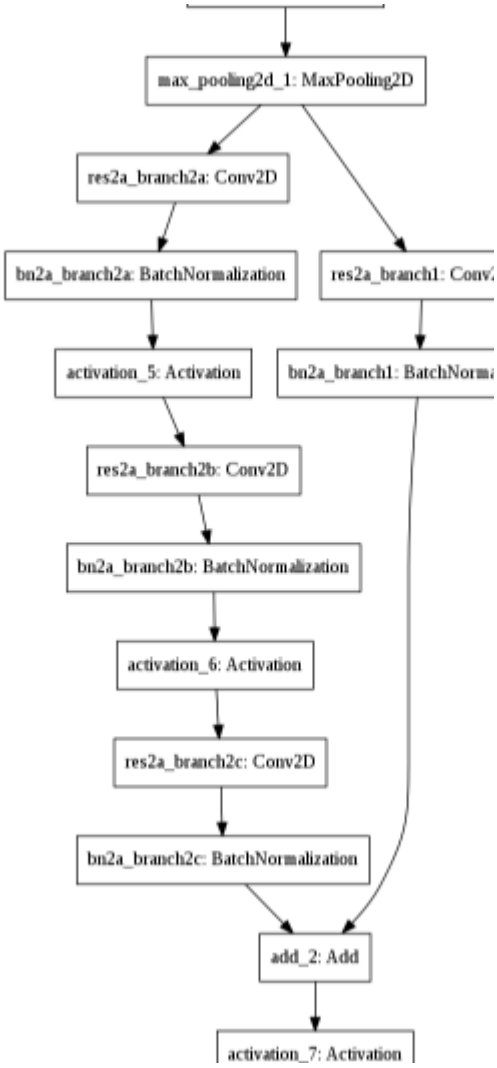
```
Epoch 1/50
 - 8s - loss: 10.0359 - acc: 0.0556
Epoch 2/50
 - 2s - loss: 7.4917 - acc: 0.0556
Epoch 3/50
 - 2s - loss: 5.3651 - acc: 0.2222
Epoch 4/50
 - 2s - loss: 4.3996 - acc: 0.2222
Epoch 5/50
 - 2s - loss: 3.6591 - acc: 0.2222
Epoch 6/50
 - 2s - loss: 3.3157 - acc: 0.1667
Epoch 7/50
 - 2s - loss: 2.7829 - acc: 0.2222
Epoch 8/50
 - 2s - loss: 2.2180 - acc: 0.3889
Epoch 9/50
 - 2s - loss: 1.8937 - acc: 0.3889
Epoch 10/50
 - 2s - loss: 1.5976 - acc: 0.5000
Epoch 11/50
 - 2s - loss: 1.4478 - acc: 0.5000
Epoch 12/50
 - 2s - loss: 1.6968 - acc: 0.6111
Epoch 13/50
 - 2s - loss: 1.4138 - acc: 0.3889
Epoch 14/50
 - 2s - loss: 1.2949 - acc: 0.5000
Epoch 15/50
 - 2s - loss: 1.1700 - acc: 0.6111
```

```
Epoch 38/50
 - 2s - loss: 0.7369 - acc: 0.7222
Epoch 39/50
 - 2s - loss: 0.8375 - acc: 0.7222
Epoch 40/50
 - 2s - loss: 0.8921 - acc: 0.6111
Epoch 41/50
 - 2s - loss: 0.8082 - acc: 0.8333
Epoch 42/50
 - 2s - loss: 0.5294 - acc: 0.7778
Epoch 43/50
 - 2s - loss: 0.7338 - acc: 0.7778
Epoch 44/50
 - 2s - loss: 0.7978 - acc: 0.6111
Epoch 45/50
 - 2s - loss: 0.7268 - acc: 0.6667
Epoch 46/50
 - 2s - loss: 0.5214 - acc: 0.8333
Epoch 47/50
 - 2s - loss: 0.7193 - acc: 0.7222
Epoch 48/50
 - 2s - loss: 0.8188 - acc: 0.7222
Epoch 49/50
 - 2s - loss: 1.0515 - acc: 0.6667
Epoch 50/50
 - 2s - loss: 0.6474 - acc: 0.7778

<keras.callbacks.History at 0x2203ff7c470>
```

We trained the mentioned ResNet50 and the VQA model in the Lab system. The Output weights along with the model were saved into these 2 files.

1. 'resnet_model_final.h5' with the weights for ResNet50
2. 'finally.h5' with the weights for VQA model

The models' tensorboard graphs can be found by clicking on these model names. Unfortunately, the model's are quite complicated, so their tensors' plot is enormous to fit in this doc. Here's a sample conv block drawn from the ResNet.



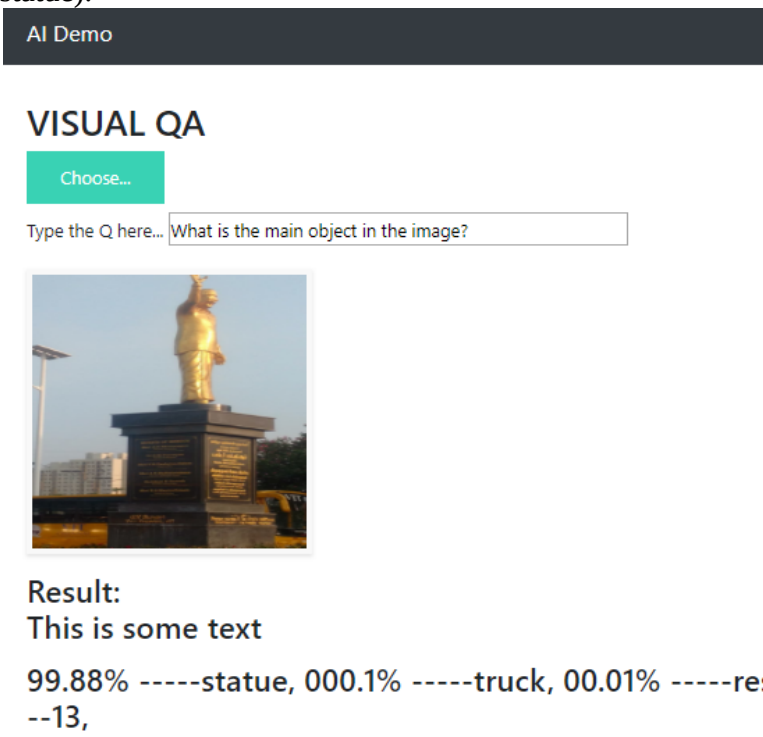
6.RESULTS

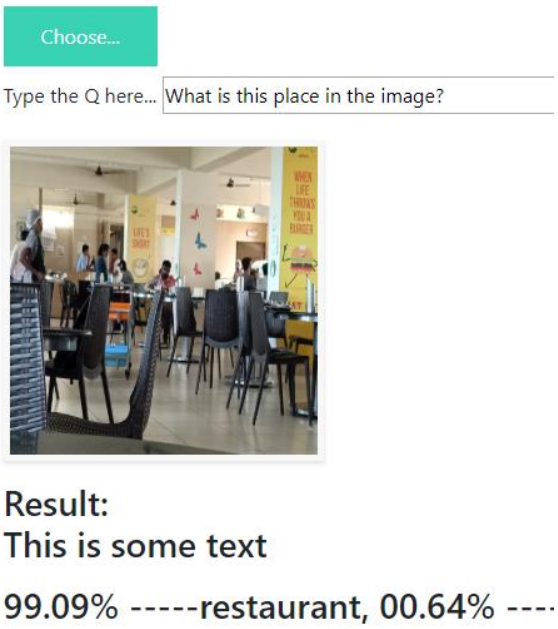
We also built a website to make the end user utilize the system interactively. The images that were tested were taken from real life scenarios in our college and the system was able to give answers to the asked question with a high level of accuracy.

Here are some test images that were provided –

Questions on the main object or places in the image

As we can see, our model is 99% sure that the object is a statue (we know its the MGR statue).

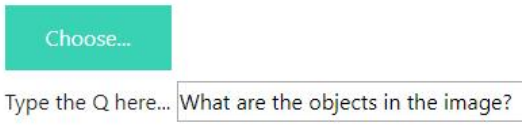




In this image of the Food Park, VITCC, we ask for the kind of place.

And the answer thrown at us, is the restaurant.

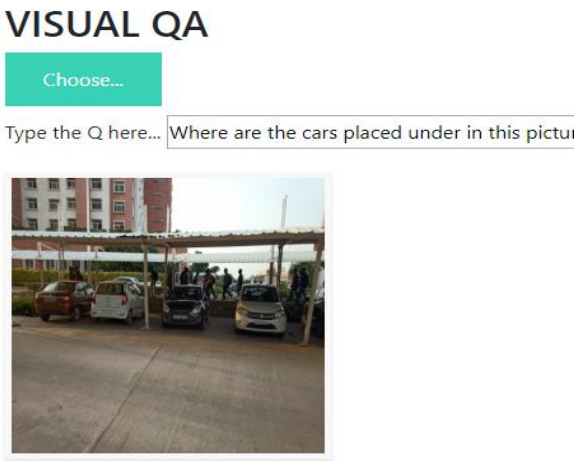
Since our learning model is 99% sure, and the rest of the predictions are just in the of under 1%, we omit those results.



Result:
This is some text

99.65% -----trees, 00.22% -----

Trees are the main objects here. And we are able to detect them correctly.

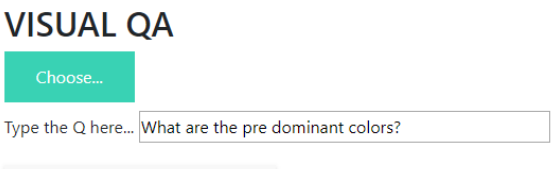


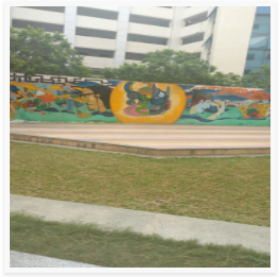
Result:
This is some text

99.91% -----parking lot, 00.03% ----

Again, the results are impressive. It predicts with a 99.9 % confidence, that this arena is a parking lot.

Questions on colors in the image

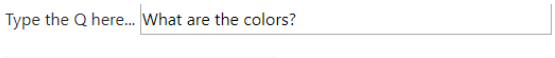




Result:
This is some text

75.68% -----blue and green, 19.82% -----red white and b
00.04% -----parking lot, 00.03% -----13,

As we can see, the green grass, the wall colors of the amphi-theatre, etc clearly show the prominence of the blue and green colors.

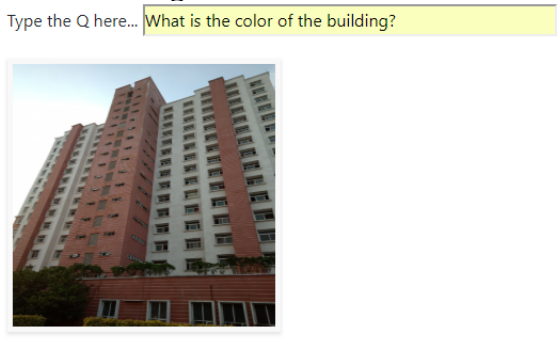


Result:
This is some text

62.94% -----blue and green, 28.32% -----red white and b
00.05% -----13, 00.05% -----28,

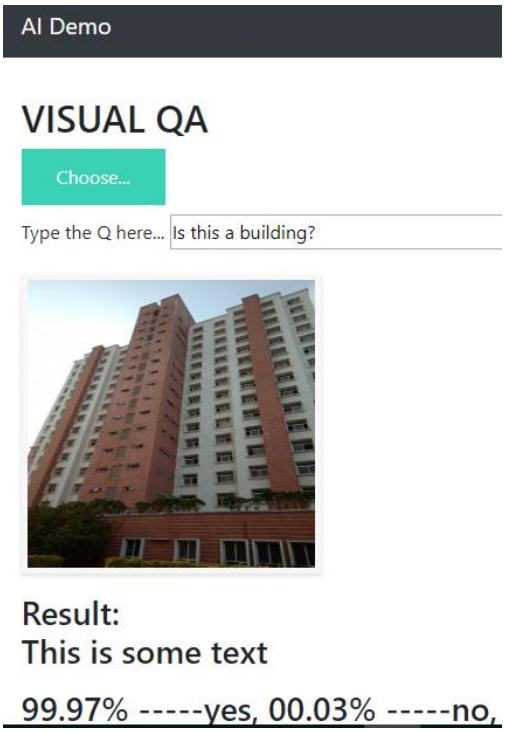
As the building is underlined with the short in-range trees and bounded by the blue sky, the VQA model picks up these 2 as primary colors. Though, the ‘red, white and blue’ combination still holds a 28% probability.

We can ask for more specifically the color of the building.

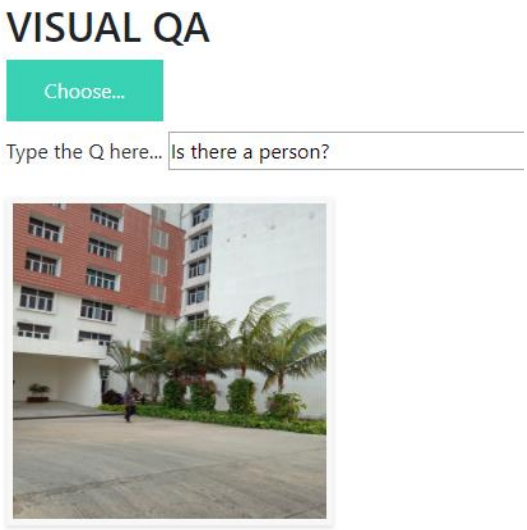


Now, it clearly draws a line between the building and the surroundings. And 'red and white' is shown to be pre dominant.

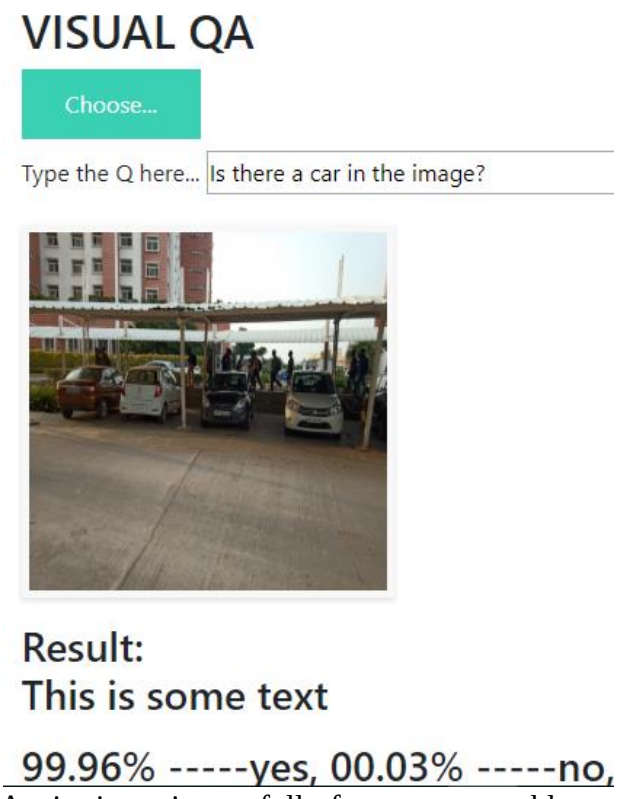
Yes/No questions



The model is almost 100% sure that this is a building.

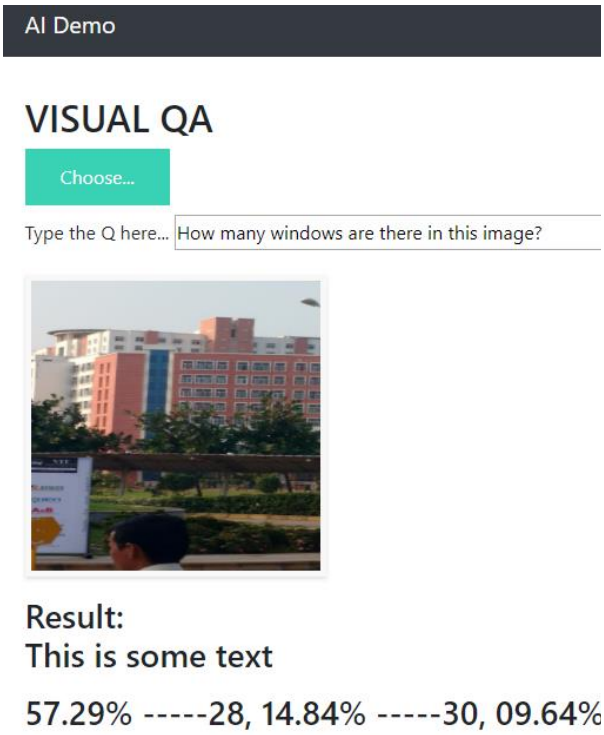


Our CNN model had learnt the features of a human. So, it is able to clearly find out the hidden feature and bring out the man walking down the aisle in the middle of the image.

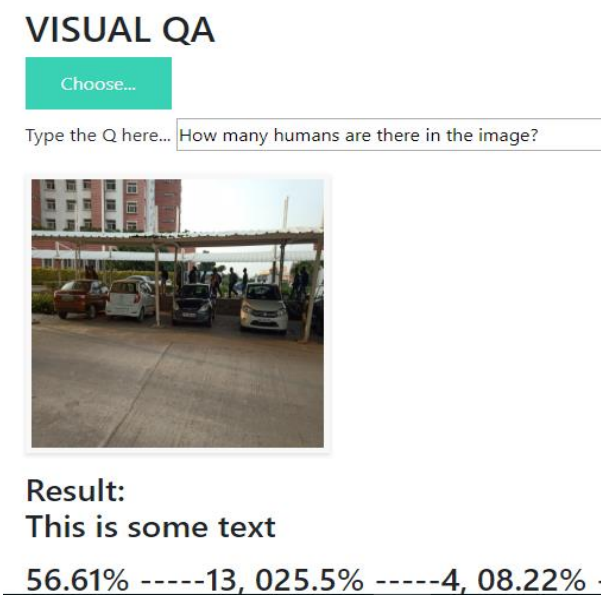


Again, in an image full of cars, we are able to confidently say, there's a car.

Questions on counts/ numbers



So, there is an approximation of 28-30 windows in the image, which is quite reasonable. The actual count is in fact, 33.



For the same image from above (the yes/no part), we see that it has some humans in it. We may also attempt to count each of them. So, there are 13 of them, as per our model. In real, there are 15 of them, as per human eye.

Note: The model from above works exceptionally well even when the same question is rephrased in a different way.

So, instead of 'Where are the cars placed under?' we might ask, 'Where is the car parked in?' Because each one of us, have our own questioning ways.

Again, the result is unchanged.

7.CONCLUSION

We summarise our learning model design phases here:

1. Segmenting images based on similar neighbourhood using Watershed (with thresholding and Canny edge detection) and Selective search.
2. Data augmentation
3. ResNet50 layer CNN for 1024 image features extraction and a RNN for Question features extraction. Followed by fully-connected layers linked onto a Softmax activation.

Overall, our model achieves a 69% accuracy on the official test dataset from VQA

dataset.org. The current VQA challenge winners have a 75% accuracy. The reasons being, the training. The intense training phase of 2.5 days was carried out with a 2.0 lakh dataset samples. The future direction of this research attempts to employ multi-node big data clusters in data prep and powerful CUDA-based *native* Nvidia GPU processing to intensively analyse the 4.5 lakh examples available at ImageNet. Then, we'll be in a position to beat the 75% accuracy and win the challenge.

Further, this model in particular excels at colors and objects detection and works well at answering most other question types. With that, we conclude the project Visual Question Answering.

8.REFERENCES

1. M. Malinowski and M. Fritz. A Multi-World Approach to Question Answering about Real-World Scenes based on Uncertain Input. In NIPS, 2014.

2. S. Antol, C. L. Zitnick, and D. Parikh. Zero-Shot Learning via Visual Abstraction. In ECCV, 2014. 2, 3

3.J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. VizWiz: Nearly Real-time Answers to Visual Questions. In User Interface Software and Technology, 2010.

4. D. Geman, S. Geman, N. Hallonquist, and L. Younes. A Visual Turing Test for Computer Vision Systems. In PNAS, 2014