**An Application to Recognize National Landmarks Using Deep Learning**

Sourav Kanta, SkyBits Technologies Prvt. Ltd

Contents

[Acknowledgement 3](#_Toc486862107)

[Background and Description of Problem 4](#_Toc486862108)

[Approach to solve problem 5](#_Toc486862109)

[Theoretical background to application 6](#_Toc486862110)

[Convolution 9](#_Toc486862111)

[Convolution Neural Networks 10](#_Toc486862112)

[Pooling Layers 12](#_Toc486862113)

[Technical Details 14](#_Toc486862114)

[Model Used 14](#_Toc486862115)

[Other models tried 17](#_Toc486862116)

[Server End Details 18](#_Toc486862117)

[Screenshots 20](#_Toc486862118)

[Future Prospects 21](#_Toc486862119)

[References 22](#_Toc486862120)

# Acknowledgement

This project would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Mr. Tirthankar Banerjee for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

I would like to express my gratitude towards my college IIEST Shibpur and particularly Professor Ajay Kumar Roy for their kind co-operation and encouragement which helped me in completion of this project.

# Background and Description of Problem

Nowadays tourism is a big source of income for any given local population as well as the state. Therefore attracting more tourists would help the local as well as the state economy flourish.

One approach to achieve this would be to keep tourists constantly well informed of the various interesting details about the places they visit. Not only does this increase a tourist’s travel experience it helps them acknowledge and appreciate the history and background of the place they are visiting. This document outlines a project that proposes a technical solution to this very requirement.

We apply state of the art deep learning techniques that can come up with an intelligent solution.

# Approach to solve problem

One thing that we can all agree upon is that the first thing each of us does while visiting a place is to take a few pictures of it, be it to capture the insane beauty of the place or as a nice addition to our own timelines. But what if one click of that camera can do more than that?

Our solution proposes to fetch the photo captured by the user and run it against a backend that applies various machine learning techniques upon this image and finally predicts the place the user is visiting and comes up with a nice little summary of the description as well as the history of the place.

We use deep learning as the basis to predict the place that the user is visiting and return an appropriate response to the user.

# Theoretical background to application

For recognizing a landmark image, i.e. image classification we are applying deep learning on the image captured by user. **Deep learning**, as the name suggests, is a sub sect of machine learning.

Deep Learning mostly involves using deep artificial neural networks (algorithms/computational models loosely inspired by the human brain) to tackle machine learning problems. State-of-the-art image classification solutions today use deep learning.

So, what is a neural network? Here’s an analogy: imagine a neural network as a series of doors one after another and think of yourself as the ‘input’ to the neural network. Every time you open a door, you become a different person (i.e. you change in some way). By the time you open the last door, you have become a very different person. When you exit through the last door, you become the ‘output’ of the neural network. Each door, in this case, represents a **layer**.

A neural network, therefore, is a collection of layers that transform the input in some way to produce an output. Each layer in the neural network consists of ‘weights’ and ‘biases’ — these are just numbers that augment the input. The overall idea of a neural network is that it takes in some input (usually a collection of numbers that represent something, e.g. Red-Green-Blue values of pixels in an image), applies some mathematical transformations to the input using the weights and biases in its layers and eventually spits out an output.

You can look at the input, output and weights as matrices. The input matrix gets transformed by a series of matrices (i.e. the weight and bias matrices of the layers) and that becomes your output.

A deep neural network is just a neural network with many layers (as you stack layers on top of another, the neural network keeps getting ‘deeper’). How many is many? Well, there’s a [VGG16](https://arxiv.org/pdf/1409.1556v6.pdf) neural network architecture (used for image classification) that consists of 16 layers and then there’s the [ResNet](https://arxiv.org/pdf/1512.03385.pdf" \t "_blank) architecture (also used for image classification) that consists of 152 layers — so, the range is pretty wide. The basic idea of deep learning is using neural networks with multiple layers.

Neural networks consist of layers that consist of weights and biases (which are just collections of numbers). During the training phase, the neural network tries to find the right weights/biases that lead to the most accurate output. It does so using a method called backpropagation. Before a neural network is trained, the weights/biases are initialized, either randomly or from a previously trained model. Either ways, when training happens, the neural network changes those weights and biases based on what it ‘learns’.

When we build a neural network, we have to decide on (i.e. choose or design) something called a **cost function**. The cost function is basically just a mathematical function that takes in the output from a neural network (for a given input) and the ground truth data (i.e. the expected output from the neural network for that given input) and calculates how off/bad the result from the neural network was.

Using optimization techniques like gradient descent, the computer calculates how to change the weights and biases such that the cost function is minimized. It keeps doing this as it trains on more and more data (get the output from the neural network, calculate cost and back propagate to change weights).

Over time, the weights and biases adjust with the data and (hopefully) you end up with a neural network that has high output accuracy. Remember, the practical effectiveness or accuracy of a neural network is largely dependent on the data used to train it; so it’s very important that the proper dataset is built or chosen. Without good data (and a good amount of data) it can be very hard to train an accurate neural network. The various layers available to us are described in the next section.

## Convolution

A Convolution is a sliding window function applied to a matrix.



Imagine that the matrix on the left represents an black and white image. Each entry corresponds to one pixel, 0 for black and 1 for white (typically it’s between 0 and 255 for grayscale images).

The sliding window is called a kernel, filter, or feature detector. Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up. To get the full convolution we do this for each element by sliding the filter over the whole matrix.

## 

## Convolution Neural Networks

Convolution Neural Networks are basically just several layers of convolutions with nonlinear activation functions like [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) or [tanh](https://reference.wolfram.com/language/ref/Tanh.html) applied to the results. In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That’s also called a fully connected layer, or affine layer.

In CNNs we don’t do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands like the ones showed above, and combines their results.

During the training phase, **a CNN** **automatically learns the values of its filters** based on the task you want to perform. For example, in Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.



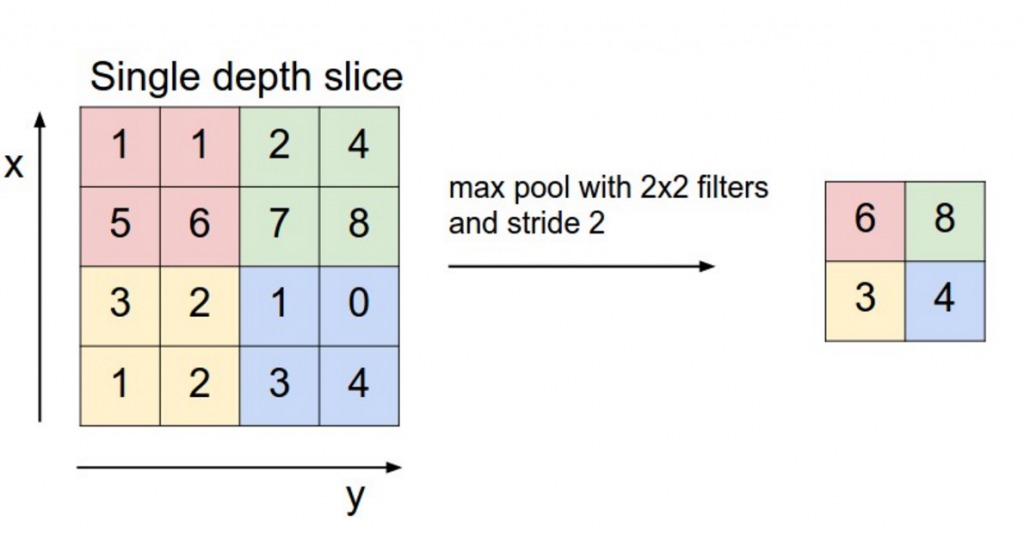
There are two aspects of this computation worth paying attention to: **Location Invariance** and **Compositionality**. Let’s say you want to classify whether or not there’s an elephant in an image. Because you are sliding your filters over the whole image you don’t really care where the elephant occurs. In practice,  pooling also gives you invariance to translation, rotation and scaling, but more on that later.

The second key aspect is (local) compositionality. Each filter composes a local patch of lower-level features into higher-level representation. That’s why CNNs are so powerful in Computer Vision. It makes intuitive sense that you build edges from pixels, shapes from edges, and more complex objects from shapes.

## 

## Pooling Layers

A key aspect of Convolutional Neural Networks are pooling layers, typically applied after the convolutional layers. Pooling layers subsample their input. The most common way to do pooling it to apply a max operation to the result of each filter. You don’t necessarily need to pool over the complete matrix, you could also pool over a window. For example, the following shows max pooling for a 2×2:



Why pooling? There are a couple of reasons. One property of pooling is that it provides a fixed size output matrix, which typically is required for classification.

For example, if you have 1,000 filters and you apply max pooling to each, you will get a 1000-dimensional output, regardless of the size of your filters, or the size of your input. In imagine recognition, pooling also provides basic invariance to translating (shifting) and rotation.

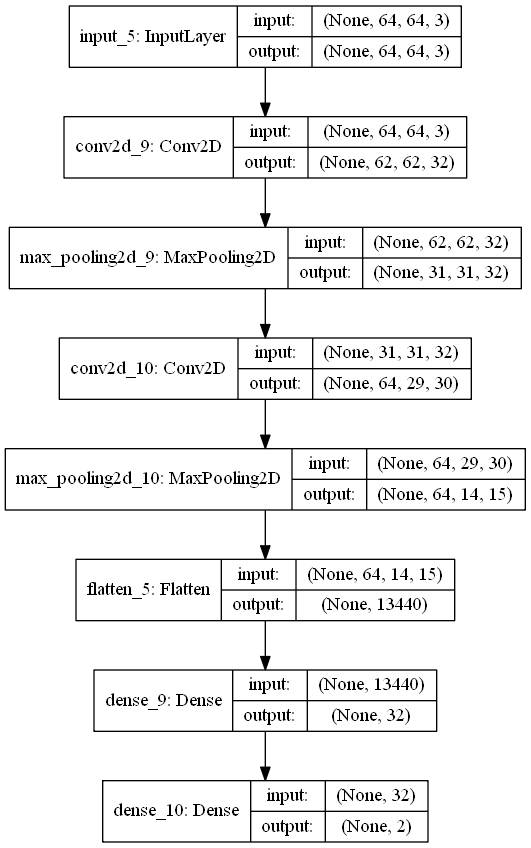
When you are pooling over a region, the output will stay approximately the same even if you shift/rotate the image by a few pixels, because the max operations will pick out the same value regardless.

# Technical Details

This section deals with the model that is used as well as the server end of the application.

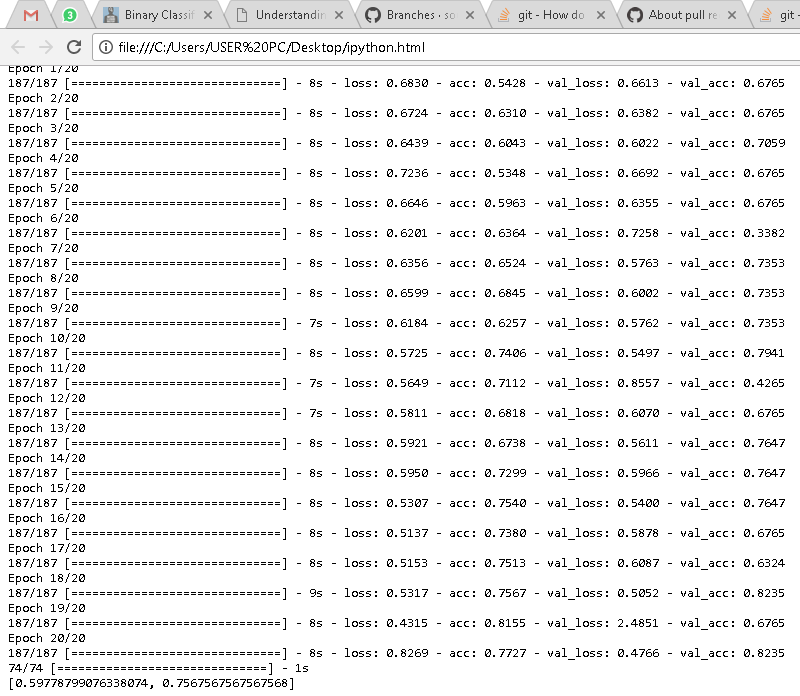
## Model Used

In our project we apply the following model for our neural network.



The model shown above contains the following structure

* The first layer is a 64 X 64 X 3 layer that accepts the numpy array with shape 64 X 64 X 3 ( Image is of size 64 X 64 and for each pixel there are three fields)
* The second layer is a Convolution layer of size 64X 64 followed by a Max Pooling layer.
* This is followed by another Convolution layer of size 32 X 32 and a Maxpooling Layer.
* Finally there is a Flatten layer and then it’s followed by two dense layers of sizes 32 and 2 respectively.
* The batch size at present is kept at 32 while the number of epochs is 20.



Output from training the model stated above

## Other models tried

Aside from this model used we have tried altering the various activations of the models and the results that were obtained were as follows:

* Categorical Crossentropy : The training set was split into an 80% and 20% half and the smaller portion was used to test the model that was obtained. We recorded a 63% accuracy for this model and hence it wasn’t used for this project.
* Our initial model had a single convolution layer which resulted in overfitting and our predictions were suffering in accuracy. Hence we have finally used another additional convolution layer to our model.
* SGD : This activation was used in the same manner as the previous one and the accuracy obtained was 67% for the same dataset. This model was also discarded for our project.
* Besides these we have also tried altering the activations for the convolution layers but the results obtained were not satisfactory.

## Server End Details

We are using a Django backend that accepts a base64 encoded image and the Keras module in Python to implement our deep learning.

Initially the neural network has been trained with about 200 images of each particular landmark and gives an accuracy of 80.3% on a stratified portion of the training dataset.

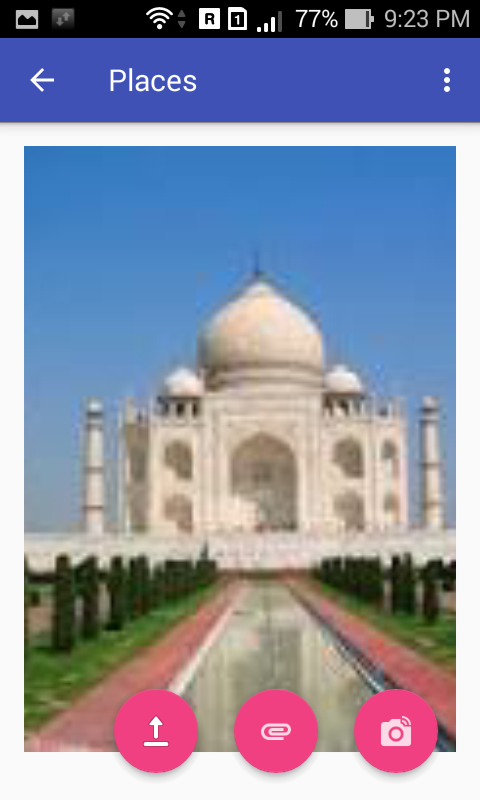
We fetch the photo the user has taken through the camera and convert it into a base64 encoded string and transmit it to the server.

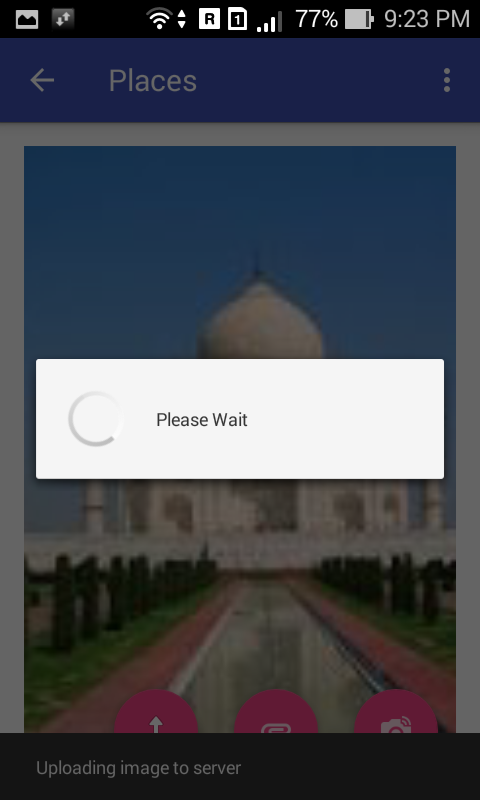
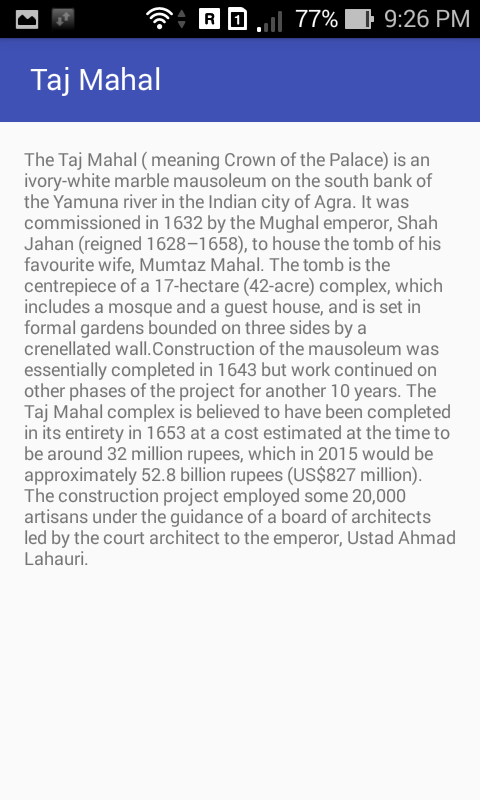
The server decodes the image and resizes it to a 64 X 64px image finally feeding it into the above neural network as a numpy array. We set our prediction by selecting the landmark which has the highest confidence value after the predict function call. We return a JSON string from the backend to the mobile application which contains the following fields:

* Status – Indicates success or failure of query
* Description – This field contains a brief summary of the landmark that will be displayed and read out to the user.
* Title – Name of the landmark
* Img – Url for a picture of the image

This API has been tested and deployed on a Heroku account the link to which can be found in the references section.

## Screenshots

# Future Prospects

Our model has been trained on two landmarks namely the Vidhan Soudha and Taj Mahal. Therefore the classification here is binary. However as the supported landmarks grow the classification will change to categorical and the model used may have to be altered.

In addition to this the training dataset has to be updated and increased for better performance and accuracy. Various data augmentation techniques can be used on the available dataset to increase the accuracy of the model.

One noteworthy thing is that the present application only works for one user at a time and multiple users causes an internal error on the server. This has to be fixed in the future.

Another feature that can be added to improve the accuracy of prediction is to get the users location and cross reference it with the prediction obtained from the server. Moreover the application has been developed for Android mobiles at present but this can be made platform independent by porting it to Ionic2.

# References

<https://github.com/sourav-kanta/server>

<https://github.com/sourav-kanta/LandmarkRecogniser>

<https://github.com/sourav-kanta/PlacerecogniserApp/>

<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/#more-348>

<https://skytourist.herokuapp.com/polls/>

<https://docs.djangoproject.com/en/1.11/intro/tutorial01/>

<http://www.iiests.ac.in/>

<http://sky-bits.com/>