

CAPSTONE PROJECT REPORT

INDIAN LANDMARK DETECTION

DEFINITION

Project Overview:

The rich heritage and deep rooted emotions for the tourists sites of India is something hardly anyone is unaware of.

But it so happens that with time, even for an Indian like me, we just remember that it is a famous place but forget exactly which one! Hence it takes a lot of research to find out which landmark is it from our bunch of pictures.

Here in this project, I'll try to identify the various Heritage Landmark sites of India using deep neural networks, so as to make things a little easy for someone.

Problem Statement:

The goal of this project, is to train a deep learning model that will be able to recognize a tourist site located in India. The model should be able to recognize arbitrary photos with a high degree of accuracy. It should also be able to warn if it is not confident enough in its prediction, for example when it is presented with an image class that it was not trained on.

Metrics:

The problem statement translates to a **multi-class classification, supervised learning problem**. Accuracy is usually a popular metric for such problems, as long as the dataset is not too skewed. The dataset compiled for this problem was not perfectly balanced but not too skewed either. Hence overall **accuracy score was chosen to be the main performance metric** for evaluating the model.

It can be calculated as follows:

$$\text{Accuracy \%} = (\text{Total number of correct predictions} / \text{Total number of images}) \times 100$$

Content Information:

- **1_train_test_analyze.ipynb** : Experiments and training results
- **2_deploy.ipynb** : Experiments for deployment code
- **requirements.txt** : Required to recreate the training / deployment environment
- **data** : Directory containing all the images used to train and test
- **deploy** : Directory containing all files necessary to deploy and test the model
- **results** : Directory of screenshots having training performance, evaluation and test results

ANAYSIS

Data Exploration:

The project is based on a list of **25 popular landmarks**, from across India. Since no relevant dataset was found, all the images for each landmark was downloaded manually and then irrelevant / corrupt images were manually filtered. This project uses images from the publicly available platforms and all images have permissive license.

Landmark	train	train-%	test	test-%
auroville	58	2.21	18	1.90
buddhist_sanchi	38	1.45	9	0.95
charminar	158	6.01	31	3.27
chhatrapati_shivaji_terminus	55	2.09	17	1.80
dakshineshwar	77	2.93	22	2.32
gateway_of_india	96	3.65	30	3.17
golden_temple	141	5.36	45	4.75
hampi	165	6.28	44	4.65
hawa_mahal	155	5.90	44	4.65
howrah_bridge	120	4.56	54	5.70
humayun_tomb	45	1.71	110	11.62
india_gate	67	2.55	22	2.32
jagannath_puri	53	2.02	21	2.22
jantar_mantar	143	5.44	32	3.38
jog_falls	85	3.23	25	2.64
kanchenjunga	120	4.56	53	5.60
lotus_temple	137	5.21	40	4.22
meenakshi_temple	88	3.35	42	4.44
mysore_palace	124	4.72	53	5.60
qutub_minar	121	4.60	36	3.80
red_fort	88	3.35	35	3.70
sun_temple	42	1.60	11	1.16
taj_mahal	240	9.13	82	8.66
victoria_memorial	166	6.31	62	6.55
wagah_border	47	1.79	9	0.95
TOTAL	2629	100.00	947	100.00

Total images = 3576

Train set % = 73.5%

Test set % = 26.5%

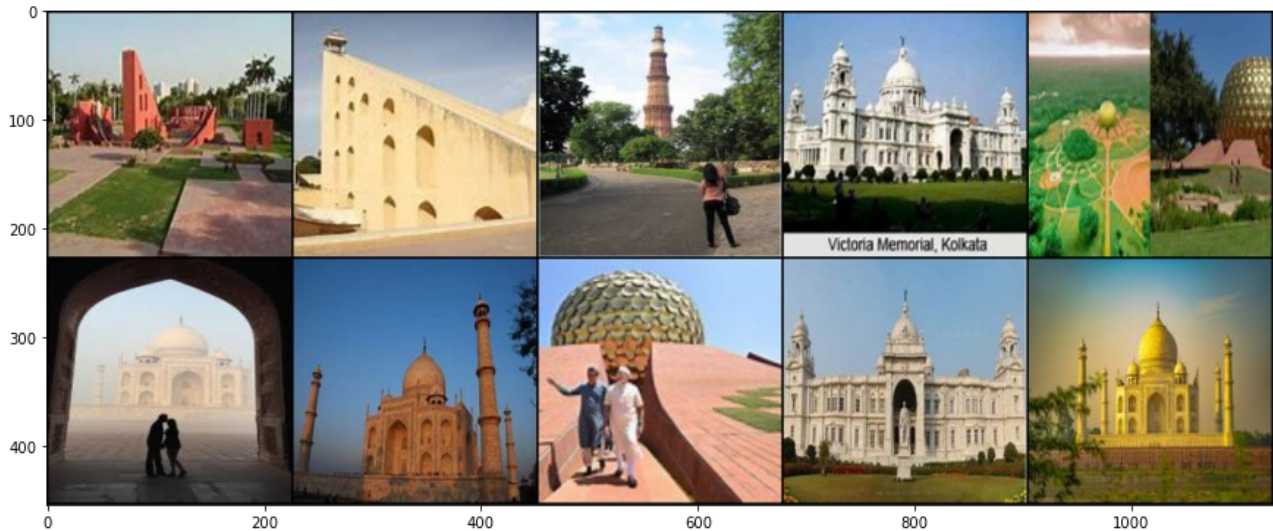
As is evident from the above table, the **images were not balanced** between the 25 classes.

The browser plugin which helped in saving time by batch downloading images is:

[Image Downloader for Chrome](#)

Exploratory Visualization:

Few examples selected from the dataset:



Algorithms and Techniques:

A [Convolutional Neural Network](#) (CNN) is a common deep learning algorithm, that is particularly well suited for image data based problems. Specifically in this case, a **multi-class classification** problem, that is to be able to classify a given image and tell it apart from other classes. CNNs take in an input image, assigns importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

In deep learning, [Transfer Learning](#) is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved then partially retrained on the specific problem's dataset. This significantly improves the convergence and overall performance of the network.

For the classification problem, transfer learning was used by utilizing a [ResNet-34](#) CNN architecture that was pre-trained on ImageNet dataset.

Specifically, the following steps were performed:

- The model was loaded with pre-trained weights for all layers
- Gradient tracking was disabled for all layers to “freeze” them
- The last “fc” (fully-connected) layer was replaced by a new layer with 25 outputs, representing our problem statement
- Only this last layer was retained as trainable
- **Categorical cross-entropy** loss function was chosen to be optimized, since it is well suited for such multi-class classification problem
- **Adam optimizer** was used for optimizing the loss function
- Batch size was chosen based on the GPU memory
- Learning rate was chosen after few trials

After the training was completed and initial evaluation performed, next job was to be able to perform the outlier detection. Since the output was restricted to 25 classes only, we could not have used a separate prediction for outlier images, plus it would not have been feasible to try to classify “everything else”!

So the logic used was as follows:

- For all the test data, prediction probabilities were calculated by using **Softmax activation** on all 25 outputs

- For each true labels, the probability for that predicted class was noted
- The average (P_{mean}) and standard deviation (P_{std}) for all these probabilities were calculated
- We assumed that for a new test input, if the **predicted probability** $< P_{\text{mean}} - P_{\text{std}}$, **then it is not confident enough** in it's prediction, even if it is the highest probability amongst all 25 classes
- This essentially means that the outlier detection depends on the variety in the test set, that was used to calculate P_{mean} and P_{std}

Benchmark:

Since the dataset is made from scratch, there is no benchmark to compare with for an exact evaluation. However, [there is a project with a somewhat similar objective](#) (dataset was not available) that was able to achieve at least **80% overall accuracy**. The objective was to reach achieve at least this my target.

For attempting a solution on a completely new dataset, an 80% overall accuracy should be well justified as target considering that a random guessing classifier for 25 classes should give an accuracy of only 4%

METHODOLOGY

Data Pre-processing:

The images downloaded for creating the dataset had various formats, resolution and aspect ratios. The following pre-processing steps were applied to the training images:

- Resize all images to a uniform resolution: 224 x 224
- Random horizontal flip with 25% probability
- Random rotation of $\pm 15^\circ$
- Transform to floating tensors
- Normalize these images with appropriate mean and standard deviation

The second and third steps were implemented for data augmentation. The last two steps were used for optimizing the convergence of the network.

Following pre-processing steps were applied for the test images:

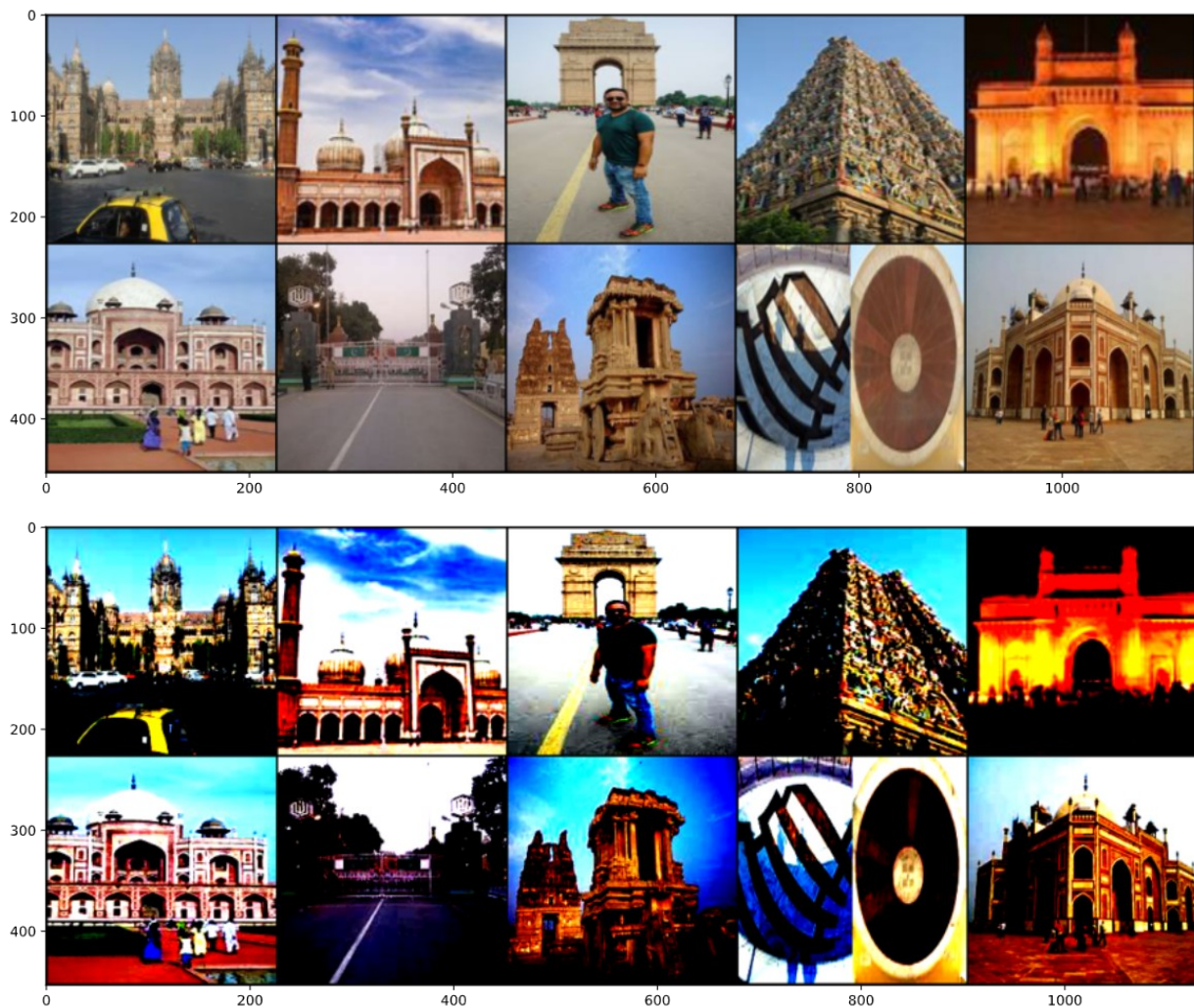
- Resize all images to a uniform resolution: 224 x 224
- Transform to floating tensors
- Normalize these images with appropriate mean and standard deviation

The data augmentation steps were removed for testing.

These are some examples of images before and after the pre-processing steps.



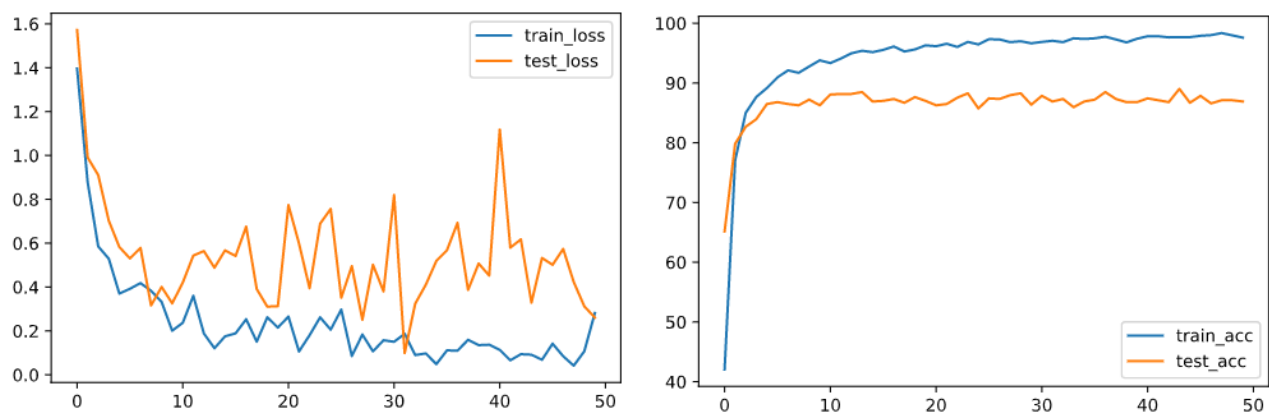
Training sample before (top) and after pre-processing (bottom)



Testing sample before (top) and after pre-processing (bottom)

Implementation:

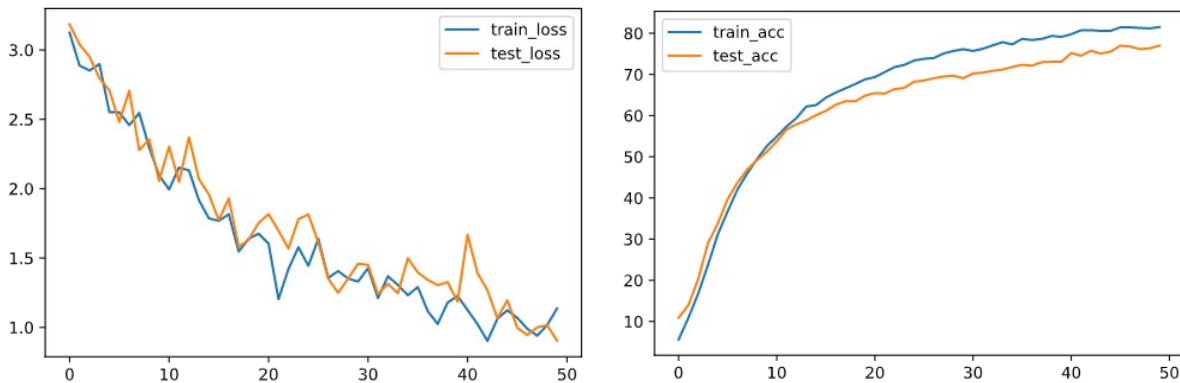
- PyTorch was used as the ML framework
- Simple data augmentation like random rotation was used
- Pre-trained ResNet-34 model was used
- The final layer was customized to have 25 outputs as per the problem statement
- Training was done for 50 epochs
- Details of training code and metrics can be found in **1_train_test_analyze.ipynb**



Refinement:

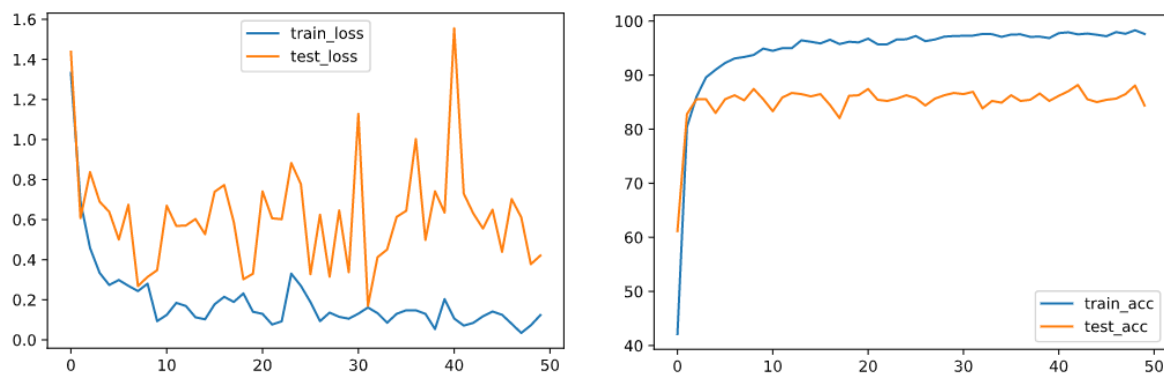
Few experiments that were initially tried are presented with their results:

- Different optimizers:
 - Cross-entropy loss
 - **SGD optimizer**
 - Learning rate = 0.01
 - Batch size = 128
 - Conclusion: Convergence was very slow, best accuracy was around 76%, higher learning rate could have helped but switched to a different optimizer altogether



SGD optimizer with learning rate of $3e-3$. The test loss and accuracy were more unstable

- Adam optimizer with higher learning rate:
 - Cross-entropy loss
 - Adam optimizer
 - **Learning rate = 0.01**
 - Batch size = 128
 - Conclusion: More fluctuations observed, best accuracy was around 86%



Adam optimizer with learning rate of $1e-2$. The test loss and accuracy were more unstable

RESULTS

Model Evaluation and Validation:

Hyper parameters used for final training:

- Cross entropy loss
- Adam optimizer
- Learning rate = 0.003
- Batch size = 128
- Epochs = 50

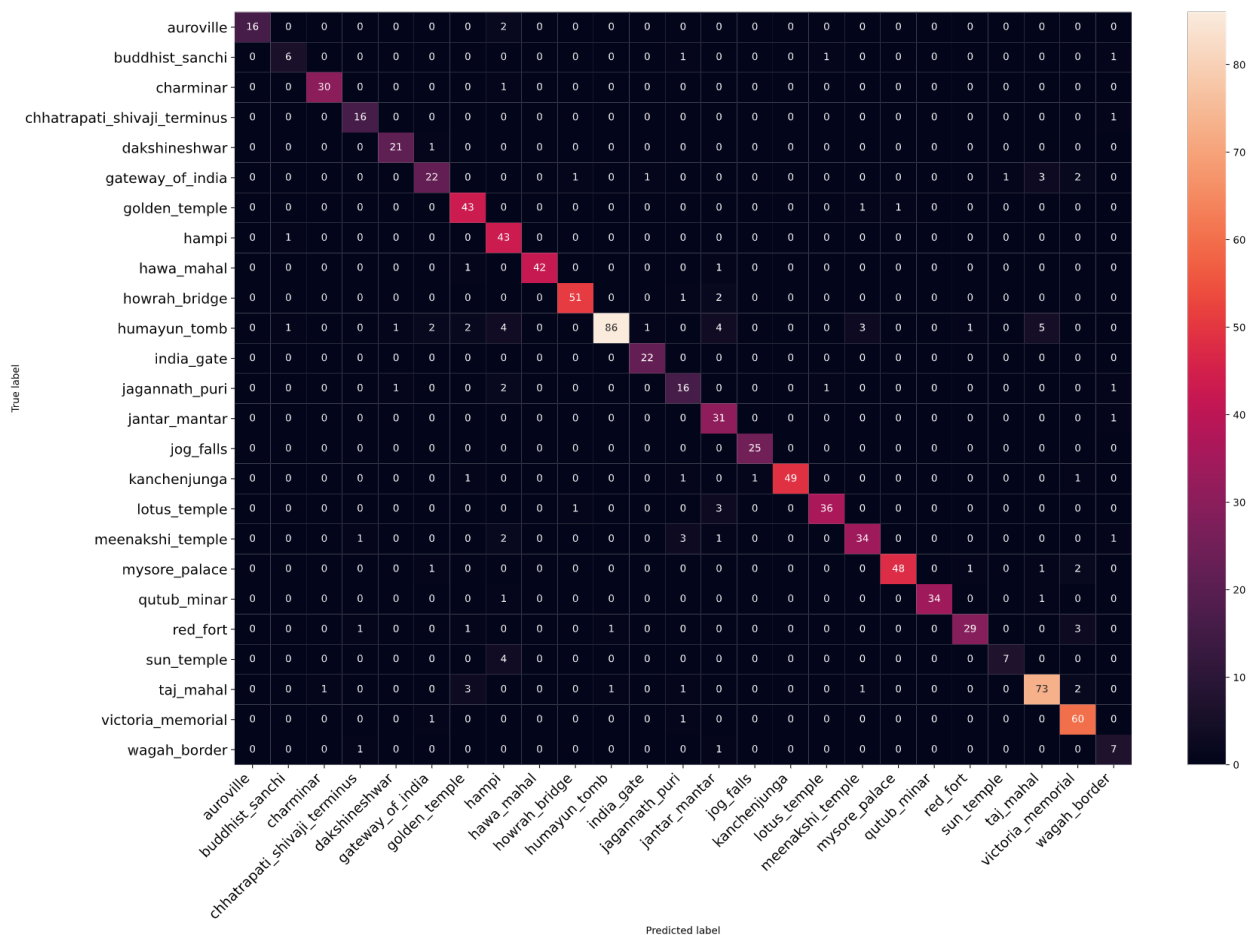
Overall accuracy on test set: **89.44%** (details in training notebook)

Target benchmark $\sim 80\%$

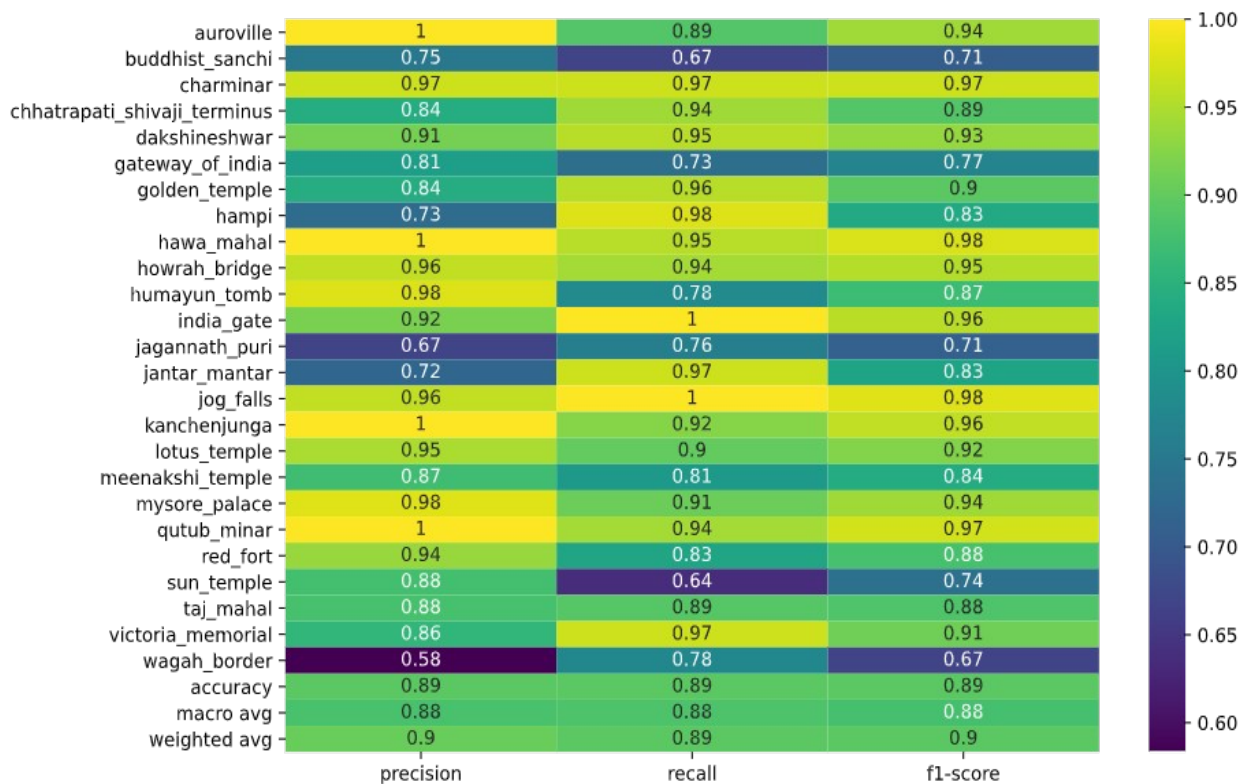
Therefore, the **trained model comfortably beat the assumed benchmark!**

Further details about the training can be inferred from the plots below:

Confusion matrix:



Classification Report:



Justification:

As indicated in the “Benchmark” section, this dataset was created from scratch so no previous work existed to have a one-to-one comparison. The **target of 80% accuracy was exceeded comfortably** with the final model. Given that for 25 classes, a random classifier would only achieve 4% accuracy, the **final result of 89.44% accuracy certainly proves the effectiveness of the learning**. Moreover, the deployed application did perform very well on completely unseen data and is very fun to play around with!

Hence I consider that the results are significant enough to have adequately solved the problem!

CONCLUSION

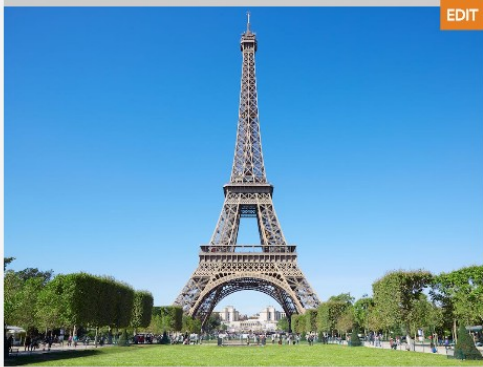
Deployment:

- I used [Gradio](#) Python module to implement a simple web based GUI
- Details of deployment code can be found in **2_deploy.ipynb**
- The notebook is also converted into a standalone Python script and placed with other necessary dependencies within the **deploy** directory
- Some results of using the web-GUI are saved under the **results** directory, e.g.:

Indian Landmark Detection

Capstone Project by Anisha Gupta for Udacity MLE-Nanodegree, October 2020

SELECT IMAGE TO CHECK



EDIT

CLEAR SUBMIT

PREDICTED LANDMARK

Gateway Of India

Gateway Of India	59%
Meenakshi Temple	10%
Hampi	9%

DESCRIPTION

The model is not very confident about this prediction. Maybe this is something unfamiliar or confusing to it!


Latency: 1.24s

gradio

Indian Landmark Detection

Capstone Project by Anisha Gupta for Udacity MLE-Nanodegree, October 2020

SELECT IMAGE TO CHECK



EDIT

CLEAR SUBMIT

PREDICTED LANDMARK

Taj Mahal

Taj Mahal	78%
Charminar	8%
Gateway Of India	5%

DESCRIPTION

The Taj Mahal is an ivory-white marble mausoleum on the southern bank of the river Yamuna in the Indian city of Agra. It was commissioned in 1632 by the Mughal emperor Shah Jahan (reigned from 1628 to 1658) to house the tomb of his favourite wife, Mumtaz Mahal; it also houses the tomb of Shah Jahan himself. The tomb is the centrepiece of a 17-hectare (42-acre) complex, which includes a mosque and a guest house, and is set in formal gardens bounded on three sides by a crenellated wall.

Latency: 1.28s

gradio


Improvements:

- The classes which are underperforming, requires more training data.
- The model might have been over-fitting.
- Training all layers might give better response
- Outliers detection can be better.

Indian Landmark Detection

Capstone Project by Anisha Gupta for Udacity MLE-Nanodegree, October 2020

SELECT IMAGE TO CHECK



EDIT

CLEAR

SUBMIT

PREDICTED LANDMARK


Qutub Minar

Qutub Minar	55%
Sun Temple	31%
Jantar Manta	4%

DESCRIPTION

The model is not very confident about this prediction. Maybe this is something unfamiliar or confusing to it!


Latency: 1.27s



Indian Landmark Detection

Capstone Project by Anisha Gupta for Udacity MLE-Nanodegree, October 2020

SELECT IMAGE TO CHECK



EDIT

CLEAR

SUBMIT

PREDICTED LANDMARK


Chhatrapati Shivaji Terminus

Chhatrapati :	91%
Sun Temple	4%
Gateway Of I	1%

DESCRIPTION

Chhatrapati Shivaji Terminus, also known by its former name Victoria Terminus, is a historic terminal train station and UNESCO World Heritage Site in Mumbai, Maharashtra, India.

Latency: 1.21s



How to test:

- In order to use the model, a virtual environment using Python 3.7 has to be created.
- Activate the virtual environment
- Run "`pip install -r requirements.txt -f https://download.pytorch.org/whl/torch_stable.html`"
- Run "**deploy/deploy.py**"
- Open any browser and go to <http://localhost:7860>
- Drag and drop any image on the left side and click "Submit"

Submitted By:

Anisha Gupta

Udacity Machine Learning Engineering Nanodegree Program 2020