TripRecs: A Landscape Classifier, Recommendation and Search Engine

Dilochan Karki, Yunik Tamrakar

April 27, 2024

1 Problem Formulation

As the Covid crisis had begun spreading around the world in early 2020, the travel industry had just been able to show an impressive steady and sustained growth over many years, reaching as high as 1.5 billion international tourist arrivals annually. With the drastic worldwide mobility restrictions in order to contain the viral spread over the course of the year 2020, the global total passenger number plummeted by 60 percent compared to 2019, leading up to a loss of 1 billion international tourist arrivals. [1] As the world emerged from the pandemic, travel restrictions became relaxed and now the international travel has entered a remarkable rebound and pent-up phase. Many people are traveling to compensate for the lack of outdoor opportunities due to the covid crisis.

Even if we disregard the covid factor, humans have always had the strong, innate need / desire to travel and explore the world. In the early stages of human evolution, humans travelled for survival and better living conditions in the wake of adverse climate. These days our travel purposes are different; we travel for enjoyment, for work or for the intrinsic joy of traveling and exploring itself. We capture pictures of our travels and many of us choose to post it online to platforms such as Instagram and share them with the world. Many of us look at these images of foreign destinations online and become impressed with the landscape enough to plan a trip to the place. As a landscape photographer, the urge to go visit a cool landscape after seeing any picture of the place is even more compelling. Now, many landscape images on the internet may have descriptors associated with them, which helps the viewers identify the place. But, there can be numerous images (of especially the lesser known hidden gems) floating around the Internet with no image descriptors.

Our solution classifies images of landscapes from around the world so that the users do not have to spend time performing reverse lookup on the images. The user can upload the image using our intuitive user interface and the software generates recommendations of landscapes very similar to the one the user just queried for. This will help the user formulate travel plans to all the recommended places or form alternate travel plans to the other recommended locations that fit their budget or schedule.

Traveling to cool destinations is one thing, finding a travel plan that fits the budget is another problem. We also provide a blazing-fast search interface for the users to query for these locations and look up travel fares to the locations so that they can factor this into their planning as well.

This is a very interesting topic relevant on a technical level. The color images these days can be pretty large and thus, training image classification models on these image datasets can involve significant workloads which merit the use of distributed and parallel systems [2]. Our training image datasets was originally well over the 30 GB mark, but after downsampling we were able to reduce the image size to around 5 GBs to account for the storage constraints. Our search engine implementation also stores numerous data points (more than 1.1 million records) and more importantly, provides responses to the user queries in a blazing-fast fashion. Robust Landscape classification models such as the ones we evaluate in our project can be used across a wide variety of disciplines such as agriculture, urban planning, environment, disaster managements. Social media platforms can also use such classification to tag the images stored, uploaded to these sites. Similarly, recommendation engines, search engines dealing with locations can be leveraged by the tourism industry to provide a better, seamless experience to their users.

2 Methodology

2.1 Background / Data Exploration

Image retrieval and instance recognition are fundamental research topics which have been studied for decades. The task of instance recognition [3, 6, 8] is to identify which specific instance of an object class (e.g. the instance "Mona Lisa" of the object class "painting") is shown in a query image. Our solution deals with instance recognition pertaining to landmark images. Instance-level recognition refers to a very fine-grained identification problem, where the goal is to visually recognize a single (or indistinguishable) occurrence of an entity. This problem is typically characterized by a large number of classes, with high imbalance, and small intra-class variation. Numerous Datasets for such problems have been introduced in the community for this task, besides the Google landmark datasets. For example: logos [4, 5, 11] cars [13, 14], products [12, 10] and so on. These datasets however, do not contain specific landmark images necessary for our task. This is where the Google Landmarks v2 dataset comes into play.

The original Google Landmarks Dataset [9] contains 2.3M images from 30k landmarks, but due to copyright related restrictions this dataset is not stable. This dataset shrinks over time as images get deleted by the users who uploaded them. The Google Landmarks v2 dataset surpasses its predecessor as well as all existing datasets in terms of the number of images and landmarks, and uses images only with licenses that allow free reproduction and indefinite retention. Thus, because of the size, diversity of the dataset as well as the creative commons licensing provision, we opt to use the Landmarks v2 dataset for our project.

The Landmarks v2 dataset can be broadly categorized into Train, Index and Test datasets. There are 4,132,914 images in the train set, 761,757 images in the index set and 117,577 images in the test set. We source the dataset for our project using the Train set. There are several metadata fields associated with dataset, namely - 'landmark_id', 'category', 'supercategory', 'hierarchical_label' , 'natural_or_human_made' fields. 'landmark_id' is an integer, 'category' is a Wikimedia URL referring to the class definition, 'supercategory' is a string referring to the type of landmark mined from Wikimedia, 'hierarchical_label' is a string corresponding to the hierarchical label, 'natural_or_human_made' is a string indicating whether the landmark is natural of human-made.

Upon exploring the actual images in the dataset, we discovered that 'photo library', 'botanical garden', 'city of regional significance of Ukraine', 'National Park of the United States', 'archaeological site' appeared to be the top 5 supercategories containing the highest number of landmark images categorized under it.(as depicted in figure 1)

Also, upon exploring the metadata in the dataset, we discovered that church, parks, mountain, castle/fort, museum were the top 5 hierarchial labels with the most occurrences in the dataset.(as depicted in Figure 2). Similarly, chrurch building, mountain, museum, castle, lake were the top 5 supercategories as per the metadata file.(as depicted in figure 3). It is to be noted that the hierarchical_label field has a one to many relationship with the supercategories field. (as depicted in figure 4)

2.2 Dataset

As we started gathering the data, we quickly realized that merely downloading the images from the Google Landmarks v2 repository would not suffice for use in our project. There were several reasons behind this assessment:

- First, the downloaded images belonged to random landscape categories without any given order. That is, say 3 consecutively downloaded images could be of mountains, churches and streets whereas the next 3 subsequently downloaded images could be of rivers, lakes and mountains. The images were not grouped based on any labels.
- The supercategories and the hierarchical label information attached to the metadata files for these landscape ids weren't sufficient enough to achieve the level of granularity we wanted in our project. (E.g: We require very specific labels such as 'Eiffel Tower', whereas the metadata only provides broader labels such as tower, tower in Europe, etc).

To solve these issues we

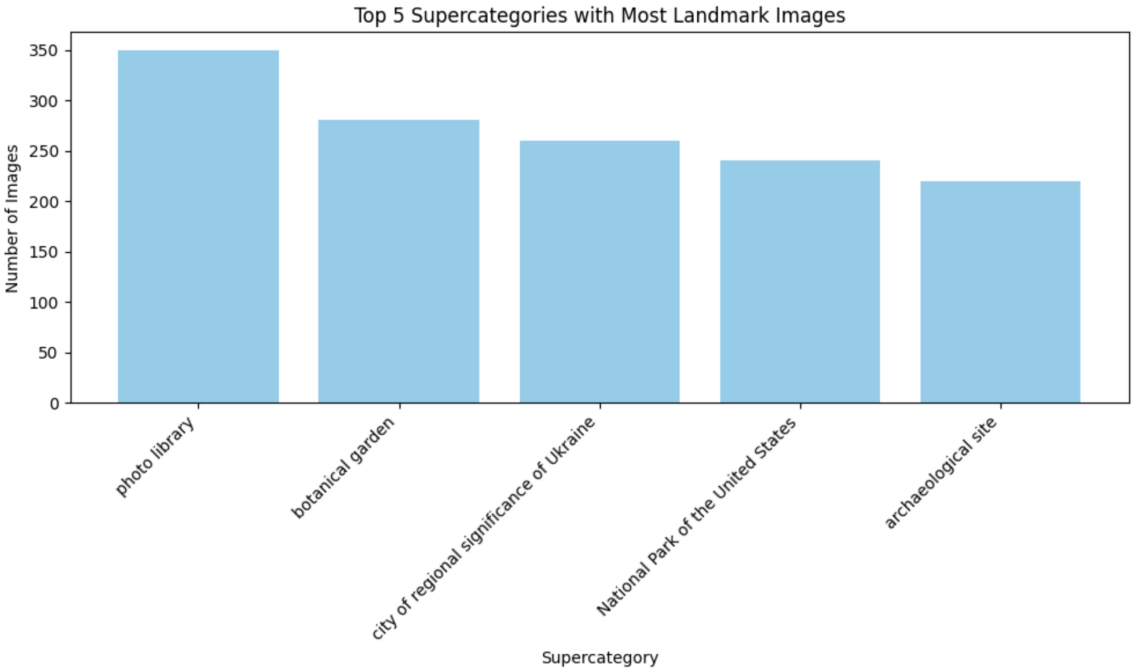

Figure 1: Top 5 SuperCategories containing most images in the train set)

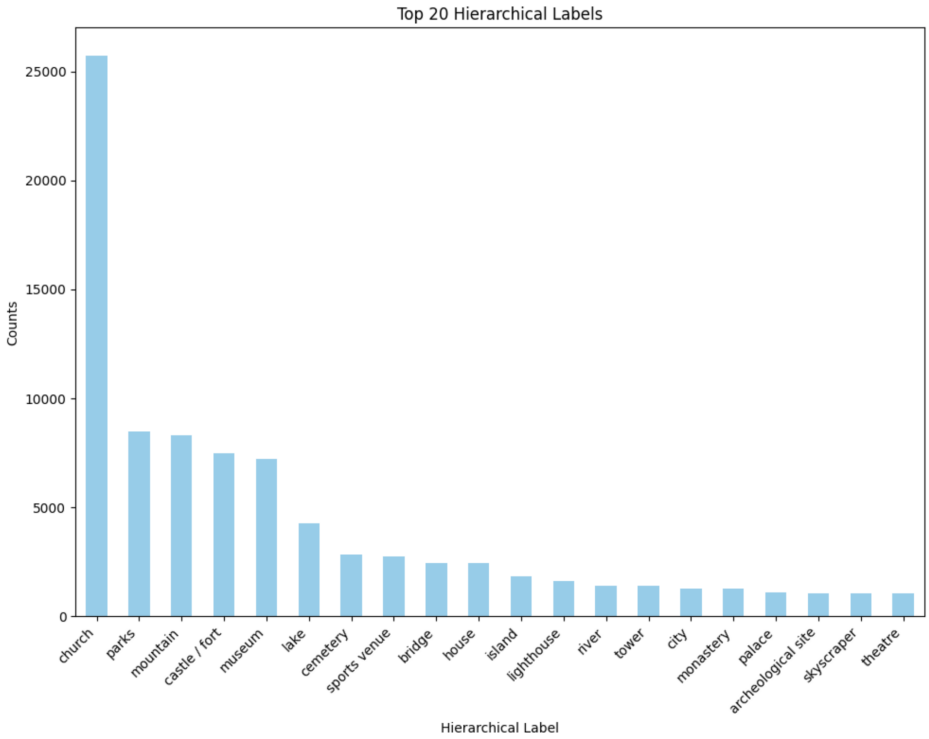

Figure 2: Hierarchial Labels ranked by Count (Metadata)

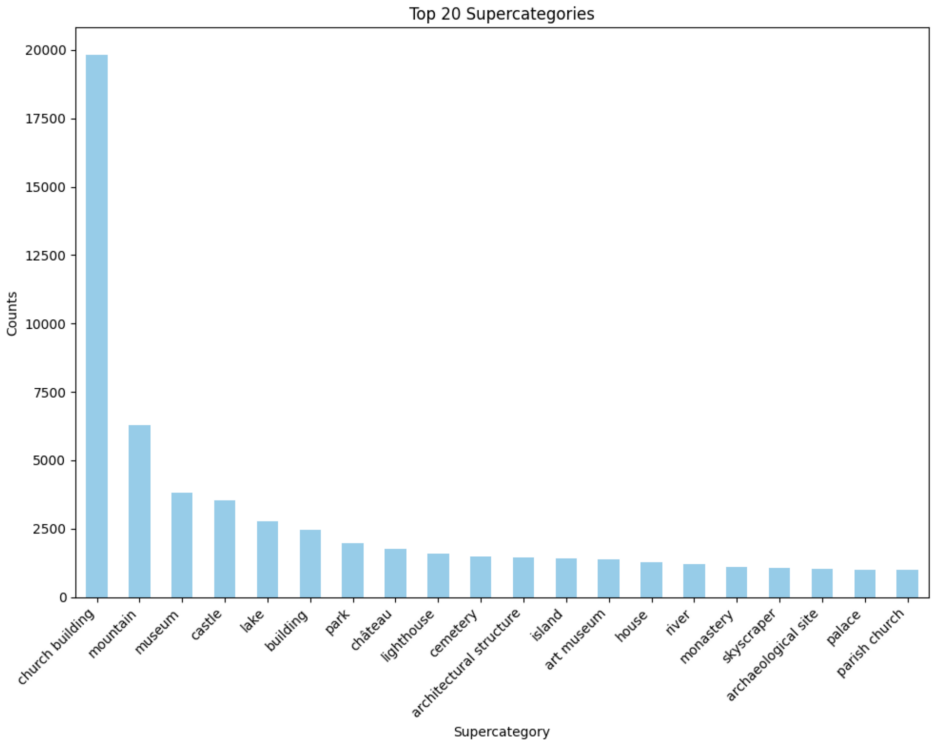

Figure 3: SuperCategories ranked by Count (Metadata)

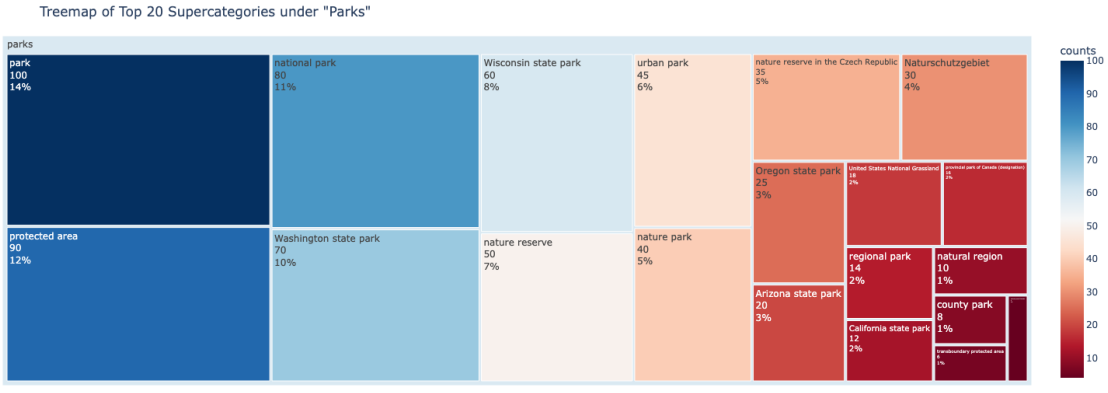

Figure 4: TreeMap depicting the One to many relationship between the Hierarchial label "Parks" and its top 20 supercategories

- programmatically grouped the image URLs by landscape IDs (listed in the train.csv file supplied by Google) and then downloaded the first 100 images to our train dataset and the remaining 25 to our test dataset.
- manually inspected the images for a given landscape ID and then labeled the folder based on previous experience, link tracing to the original Wikimedia image source or via reverse image lookup.
- we also manually sourced Creative commons images from search engines such as Google, Bing and other sources such as the Udacity repository and then manually annotated the collected images.

2.2.1 Explorative Analytics with Apache Spark

We performed further exploratory analytics on the available metadata csv file for the landmark images dataset. Since, the size of the metadata file was too large, we considered using Apache Spark - a distributed computing framework which would be suitable to perform queries and analyze attributes of the dataset. For this, we setup a SparkSQL schema based on the columns of the csv file and loaded it as dataframe. Then we developed queries to rank the number of occurrences of supercategories, hierarchical labels and indicators of whether a landmark was manmade or natural. Plot visualization of these aggregates can be found in 3 and 2. It took 13.141 milliseconds to complete query for counts of manmade/natural indicators, 15.720 milliseconds for the query to rank hierarchical labels and 59.740 milliseconds to query rank of supercategories. The number of distinct values for each attribute being larger for supercategories might have contributed to longer times to perform the aggregation query.

2.2.2 Pre-Processing

The original images from the Google Landmarks v2 dataset come in all sizes. We noticed individual image files ranging anywhere from a few hundred KBs to 10 MB+ in size. Since our project has numerous instance labels (57 instance labels) and numerous images (125 images) belonging to each label, we opted to downsample the images (to 35 percent of original size) to reduce the storage costs for the dataset. We also noticed several files were in the TIFF format and we converted these to jpeg to maintain consistency of the image formats across our dataset.

Once we confirm our dataset, our routines load the images from the local image directories and then, the data is split into train, validation and test sets. One thing to note is that we perform image transformations on the loaded images for all sets. We crop the images to 224 x 224 because it is the recommended image size for using with the pytorch models, after which we convert it to a tensor and then normalize it. For the train set, we also perform data augmentation operations on the images. Data augmentation involves applying a series of random transformations to training images, ensuring that the model generalizes well across varied data and is not overfit to the training dataset. Several researches have pointed out that certain transformations can result in improved performances for the model and also, that data augmentation has the potential to significantly improve the generalization of deep learning models.

Figure 5: Visualization of one batch (size = 5) of the pre-processed dataset

2.3 Deep Learning

After pre-processing our image dataset, these images become ready to be fed into our Deep Learning models as inputs. For this project, our chief contributions pertaining to Deep learning include:

- We have designed and built a custom Convolutional Neural Network model from scratch using Pytorch.
- We have set up several models for transfer learning using the PyTorch deep learning framework. We have trained and evaluated:
  - ResNet-18
  - VGG-16
  - MobileNet-Small-V3

Expectations We expect the transfer learning models to perform significantly better than our custom model, however, with a minimum benchmark accuracy of 50 percent set for our custom model, we still expect our model to be serviceable in production.

2.3.1 Custom CNN Model

After experimenting with several design choices, we chose to implement our CNN model with 5 convolutional layers and 3 Fully connected layers. The first 3 convolutional layers contain ReLU activations, whereas the final two convolutional layers as well as the fully connected layers consist of LeakyReLU activations. We also have max pooling layers for all convolutional layers and also, perform batch normalization for some convolutional layers and all fully connected layers to normalize the intermediate outputs of each layer within a batch during training, making the optimization process more stable and faster. Some layers do not perform batch normalization.

We also have added a few dropout layers to reduce overfitting. The configuration described above has been selected mostly for empirical reasons as that configuration helped us attain our primary goal of achieving a minimum of 50 percent accuracy, which we set out in the original project proposal.

2.3.2 Transfer Learning

For our transfer learning phase, we load the models with the default pre-trained weights. Then, the parameters of the loaded model are frozen. This is a common practice in transfer learning where the pre-trained weights of the model are not trained further. This is done by setting requires_grad to False for all parameters. This prevents gradients from being computed for these parameters during training, thus the weights remain unchanged. A new fully connected layer (nn.Linear) is then created to replace the existing final layer. This new layer is designed to output the desired number of classes (n_classes), effectively tailoring the model to the specific landmark classification task at hand.

2.4 Distributed Training

For the purpose of distributed training, we used Distributed Data-Parallel Training (DDP) which allows to train a deep neural network in a single-program multiple-data training paradigm[7]. DDP creates a replica of the model on every process and feeds it with a sample of input data. The data sample fed as input will be different for each replica and is created using DistributedSampler. First, it checks if the dataset size is divisible by the number of replicas. Shuffle in the DistributedSampler wasn't used for our experiments. In gist, the sampler is fed into the training, validation and test dataloaders then it simply subsamples the subset data from whole dataset for each worker process. The model replicas are synchronized through gradient communication and its also overlapped with gradient computations to speed up the training process[7]. It is done before the optimizer step to make sure that the model replicas will stay consistent across various training iterations.

To use DDP with our custom model and transfer learning model, it required wrapping up the defined neural network with the DDP class. It also requires calling the initialization process group of worker processes that will communicate with each other during training and start the communication backend. The communication backend used for our experiments was Gloo which sets up the distributed environment by establishing connections between the worker processes and allocating resources.

Our training experiments for the custom model and transferred models were run on the Falcon HPC cluster provided by the Department of Computer Science. We defined a slurm job that used two Nvidia A100 GPUs in a single node and had two worker process running. For smoother batch loading, we setup two cores per each worker process and allocated 4GB memory per each core.

2.5 Exporting the models

For the custom pre-trained model as well as each of the three deep learning models used for transfer learning, after distributed training, we export the models to the pt format using the torch.jit.script. This model is then deployed to production and used in our web application.

2.6 Production Web Application

The production web application is built using Streamlit in python. Streamlit allows rapid prototyping of the UI using python code. While not as powerful or flexible as Nodejs powered SPA frameworks such as Angular, React or Vue, StreamLit is perfect for our task given its ease of use and minimality. The Web UI consists of three separate pages.

Inference/Recommendations Page This is where the user can upload the images of the landmark and get the predictions on the spot. Not only that, the user is able to select from 4 of our trained models for the classification task. For a given image, the top k = 5 prediction labels are generated using the softmax probabilities reported by the DNN model. We fetch 4 * 5 = 20 images from each of 2nd to 5th ranked prediction labels and then display these as the recommendations.

Image Search Page The users can type in their query in the search box and then be able to lookup the images pertaining to their query. The images are fetched from the local image repository as all these images are licensed under Wikimedia commons. Using an approach such as scraping could potentially result in licensing issues and thus, we do not pursue those methods.

Trip Fare Search Page The users can type in the location in the search bar and then the keyword is transformed into a Solr query, which ultimately returns a set of Travel fare records pertaining to the keyword, stored in our Solr backend.

2.7 Search Engine - Apache Solr

The Solr core (AirFares) consists of around 1.1 million records corresponding to the airlines fare to the destinations. It is to be noted that the data for the Solr core were synthesized and augmented using the Mockaroo website as well as GPT-4 (3rd party APIs were not feasible due to cost constraints). We use the edismax parser as our default Solr query parser and use the built-in ranking model provided by Solr.

3 Results and Evaluation

3.1 Overview

For all our deep learning models, we use several metrics to evaluate our deep learning model and the distributed training process. Some of these include:

- Prediction Accuracy on the Test Dataset
- Confusion matrices
- Sensitivity Analysis
- Loss function
- Distributed training observations

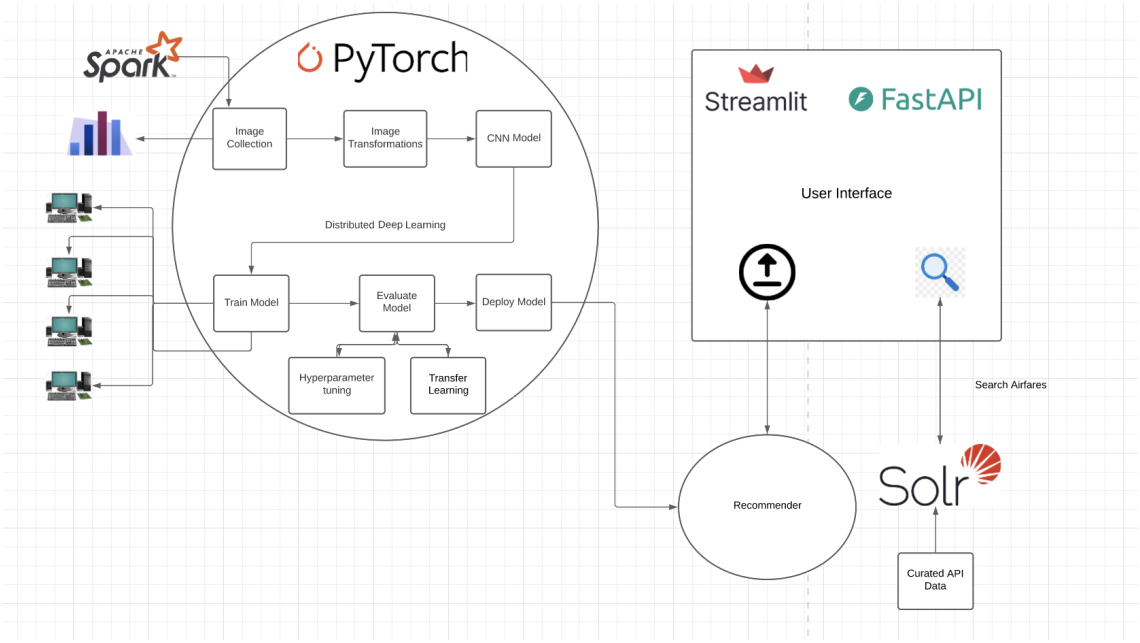

Figure 6: Solution Architecture

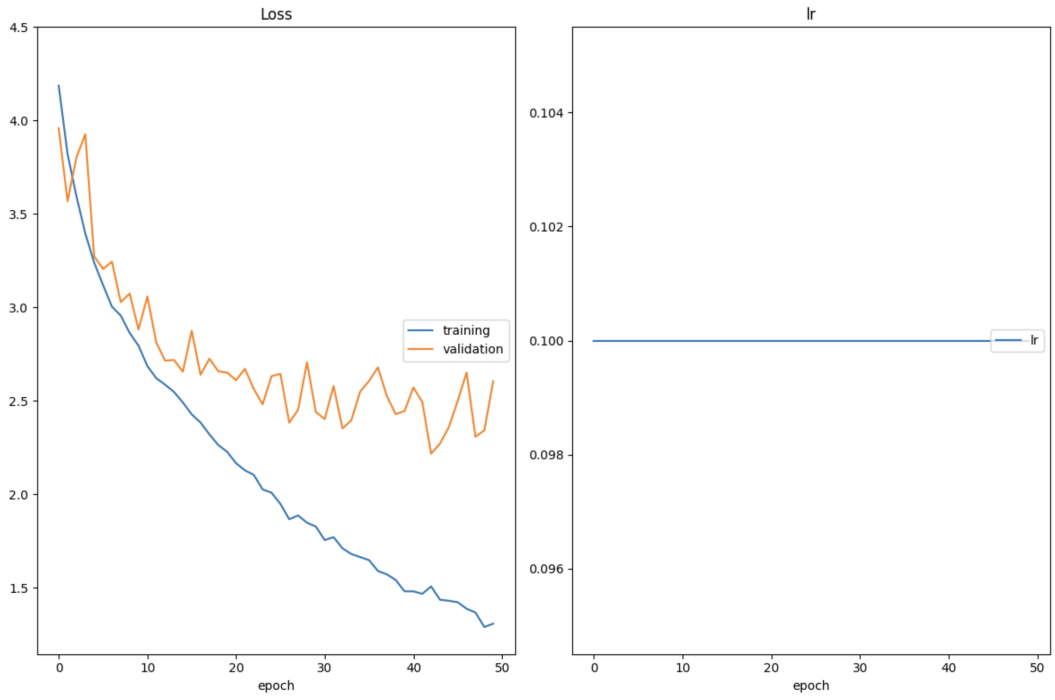

Figure 7: Loss plot of Custom CNN for 50 epochs

3.2 Custom CNN model

For our hand-crafted CNN model, the goal was to attain a classification accuracy of at least 50 percent. This is a respectable benchmark since image classification is a historically difficult task and instance classification is even more difficult.

This model was trained for 50 epochs using the SGD optimizer and a learning rate of 0.1 (full details of the config in the subsequent sections). The model fell just short (49 percent) of our benchmark test accuracy of 50 percent. We decided to then train the model for a bit longer (70 epochs) and then evaluate the results. We were able to attain an accuracy of 51 percent with this hyperparameter configuration and thus, were able to achieve our benchmark accuracy. The full hyperparameter configuration is specified as (batch size = 64, validation size (fraction of the training data to reserve for validation) = 0.2, Number of epochs for training = 70 learning rate = 0.001 optimizer = 'adam' weight decay = 0.001)

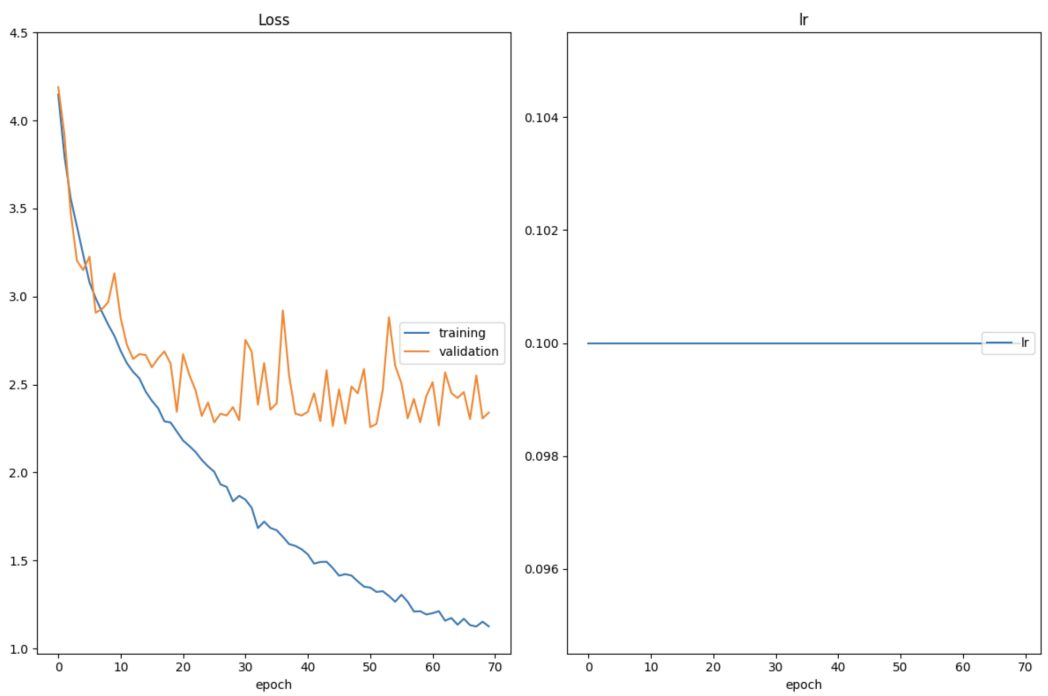

Figure 8: Loss plot of Custom CNN for 70 epochs

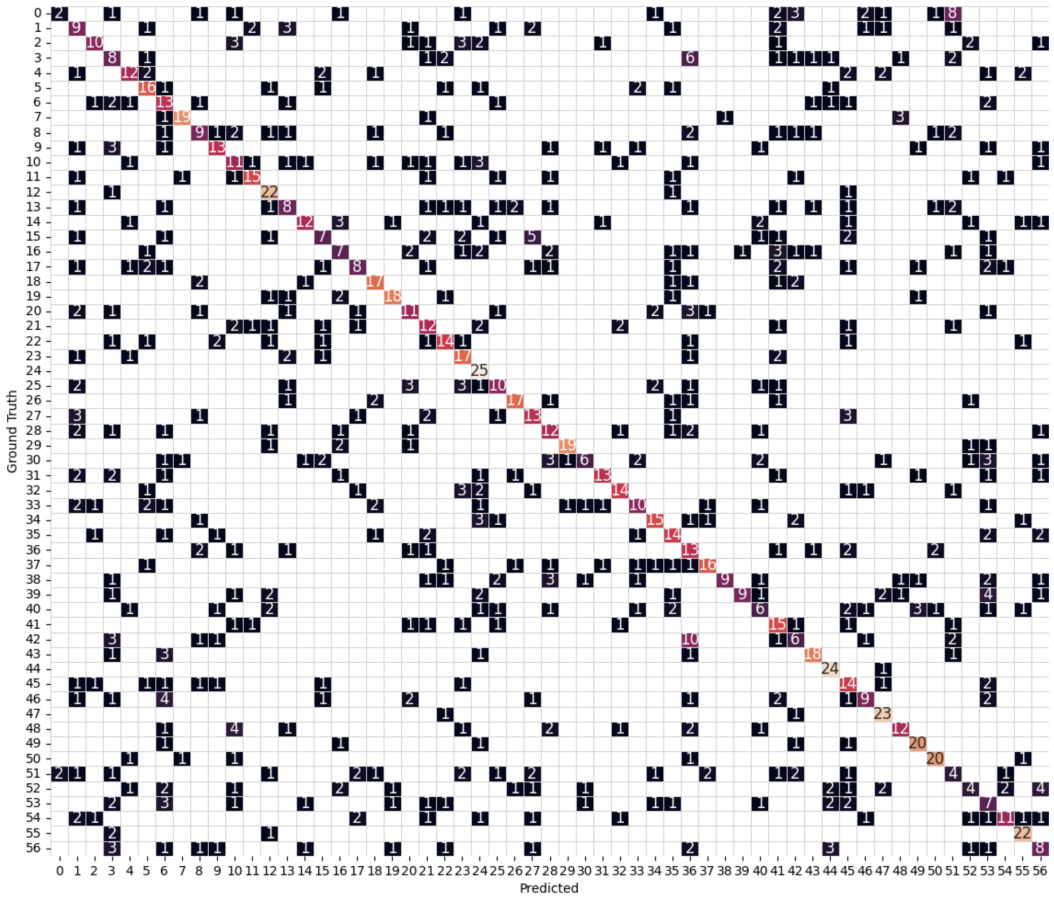

Figure 9: Confusion matrix for our custom CNN

Confusion matrix Evaluation Upon investigating the confusion matrix, we noticed:

- The custom model performed best for labels 24: Soreq Cave (25/25 correct), label 44: Trevi Fountain (24/25 correct), label 47: Prague Astronomical Clock (23/25 correct), label 49: Temple of Olympian Zeus, 50: Noratus Cemetery Armenia (20/25 correct) and label 12: Kantanagar Temple (22/25)
- The custom model performed the worst for label 0: Haleakala National Park (2/25 correct), labels 51: Maui, 52: Lviv (4/25 correct) and label 30: Brooklyn Bridge (6/25 correct)

3.3 Transfer Learning : Resnet-18

We applied transfer learning using the ResNet-18 model from pytorch repository. We froze the parameters of the model so that pre-trained weights of the model are not trained further. We added a new fully connected layer (nn.Linear) to replace the existing final layer. This new layer is designed to output the desired number of classes (n_classes = 57), effectively tailoring the model for our landmark classification task at hand.

The model was trained for the same hyperparameter configurations deemed optimal for our custom model. (batch size = 64, validation size (fraction of the training data to reserve for validation) = 0.2, Number of epochs for training = 70 learning rate = 0.001 optimizer = 'adam' weight decay = 0.001)

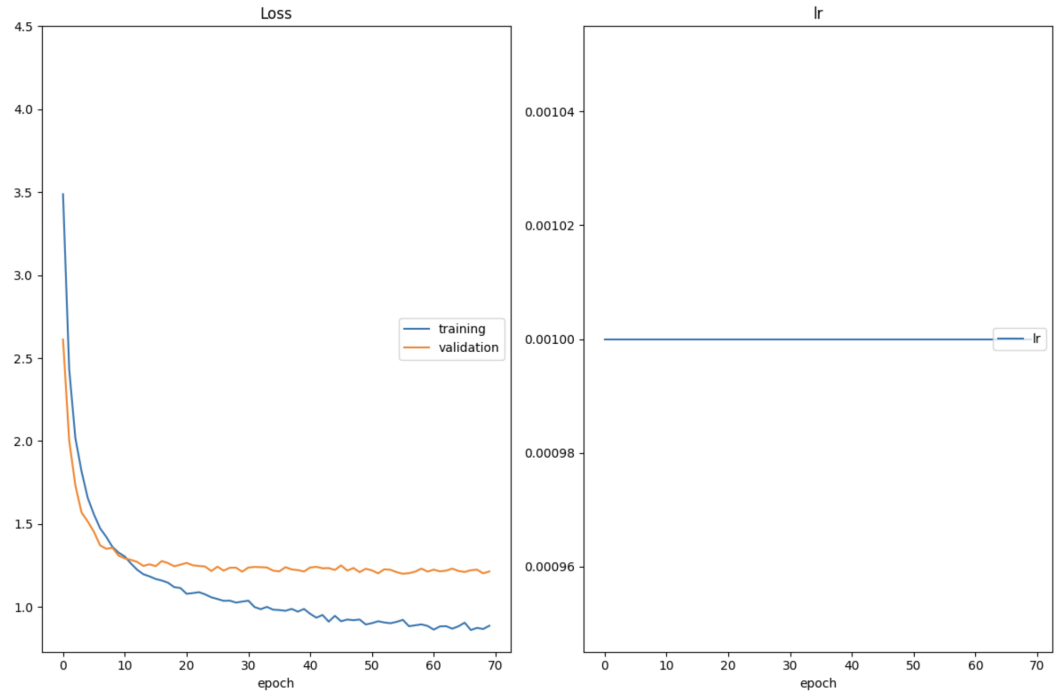

Figure 10: Loss plot of ResNet-18 for 70 epochs

The ResNet-18 transfer model performed significantly better compared to our custom CNN, which wasn't all that surprising. This model was able to achieve a test accuracy of 70 percent on our dataset. Also, a minimum validation loss of 1.17 was observed for this model. Upon investigating the confusion matrix (Figure 10), we noticed:

- The model performed best for Labels 55: Harvard, 47: Prague Astronomical Clock, 24: Soreq Cave, 44: Trevi Fountain (25/25 correct), labels 7: Stonehenge, 43: Gullfoss Falls, 50: Noratus Cemetery Armenia (24/25 correct), labels 37: Atomium, 29: Petronas Towers, 12: Kantangar Temple (23/25 correct)
- The model performed the worst for labels 56: Kazan Tatarstan (4/25 correct), label 0: Haleakala National Park (5/25 correct) and label 3: Dead Sea (3/25 correct)

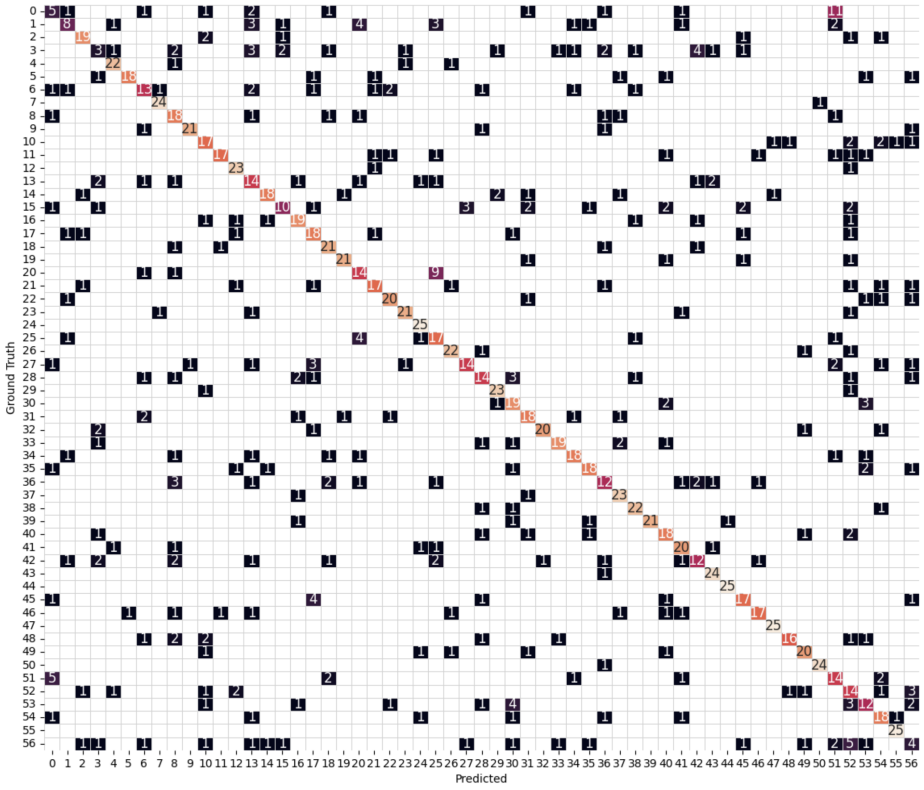

Figure 11: Confusion matrix for ResNet-18 after 70 epochs

- **For labels that the base model did well for**: the ResNet-18 transfer model performed very well . Labels 24: Soreq Cave, 44: Trevi Fountain, 47: Prague Astronomical Clock achieved perfect accuracy (25/25 correct), label 49: Temple of Olympian Zeus (20/25 correct), label 50: Noratus Cemetery (24/25 correct), label 12: Kantanagar Temple (23/25 correct).
- **For labels that the base model struggled with**: the ResNet-18 transfer model achieved mixed results. For label 0: Haleakala National Park (5/25 correct) the model struggled mightily, for label 51: Maui ,52: Lviv (14/25 correct) the model performed better. And for label 30: Brookyln Bridge, the model performed very well (19/25 correct)

3.4 Transfer Learning : VGG-16

Similar to ResNet- 18, we applied transfer learning using the VGG-16 model from pytorch repository. We froze the parameters of the model so that pre-trained weights of the model are not trained further. We added a new fully connected layer (nn.Linear) to replace the existing final layer. This new layer is designed to output the desired number of classes (n_classes = 57), effectively tailoring the model for our landmark classification task at hand.

The model was trained for the same hyperparameter configurations deemed optimal for our custom model. (batch size = 64, validation size (fraction of the training data to reserve for validation) = 0.2, Number of epochs for training = 70 learning rate = 0.001 optimizer = 'adam' weight decay = 0.001)

The VGG-16 transfer model again performed significantly better compared to our custom CNN and performed slightly better than ResNet-18 for this task as well. This model was able to achieve a test accuracy of 73 percent on our dataset. Also, a minimum validation loss of 1.05 was observed for this model.

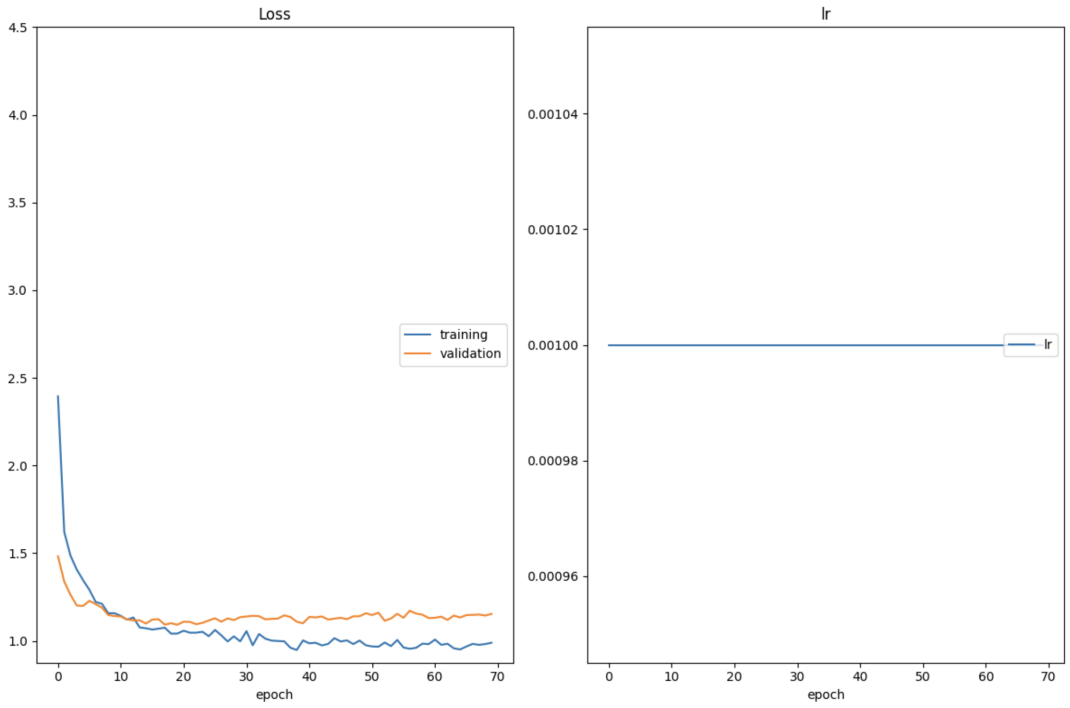

Figure 12: Loss plot of VGG-16 for 70 epochs

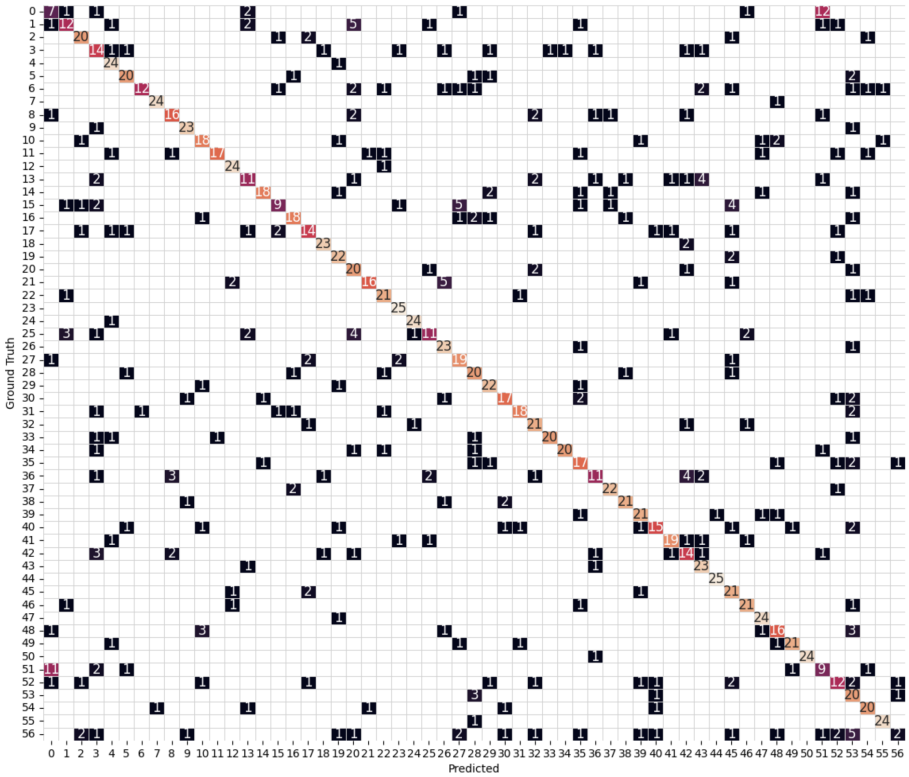

Figure 13: Confusion matrix for VGG-16 after 70 epochs

Confusion matrix Evaluation Upon inspecting the confusion matrix, we noticed:

- The model performed best for labels 44: Trevi Fountain, 23: Externsteiene (25/25 correct), for labels 4: Wroclaws Dwarves, 7: Stonehenge, 12:Kantanagar Temple, 24:Soreq Cave, 47: Prague Astro Clock, 50:Noratus Cemetery Armenia, 55: Harvard (24/25) and labels 43: Gullfoss Falls, 26: Pont du Gard, 18: Delicate Arch, 9: Golden gate bridge (23/25 correct)
- The model performed the worst for label 56: Kazan Tatarstan (2/25), label 0: Haleakala National Park (7/25). However, for label 3: Dead Sea it performed suprisingly well (14/25 correct) compared to ResNet-18 (3/25 correct)
- **For labels that the base model did well for**: the VGG-16 model did excellent for all. For label 44: Trevi Fountain, perfect accuracy was reported (25/25 correct), label 24: Soreq Cave, 47: Prague Astro Clock(24/25 correct), label 12: Kantanagar Temple, 50: Noratus Cemetery Armenia (24/25 correct) and label 49: Temple of Olympian Zeus (21/25 correct).
- **For labels that the base model struggled with**: the VGG-16 model achieved mixed but much better results than base for these labels. For label 0: Haleakala National Park, it performed better than ResNet-18 and base, but the results were still poor (7/25 correct). For label 51: Maui (9/25) it struggled compared to ResNet (14/25). For label 52: Lviv, the result (12/25) was comparable with ResNet (14/25). Similar comparable results for label 30: Brookyln Bridge, was observed (17/25) versus ResNet (19/25 correct). VGG-16 however beats ResNet-18 over other labels not scrutinized here and thus, achieves higher overall test accuracy.

3.5 Transfer Learning : MobileNet-v3-Small

Last but not the least, we applied transfer learning using the MobileNet-v3-Small model from the pytorch repository. Again, we preserved the pre-trained weights and simply added a new fully connected layer (nn.Linear with n _classes = 57) to replace the existing final layer.

The model was trained for the same hyperparameter configurations deemed optimal for our custom model. (batch size = 64, validation size (fraction of the training data to reserve for validation) = 0.2, Number of epochs for training = 70 learning rate = 0.001 optimizer = 'adam' weight decay = 0.001)

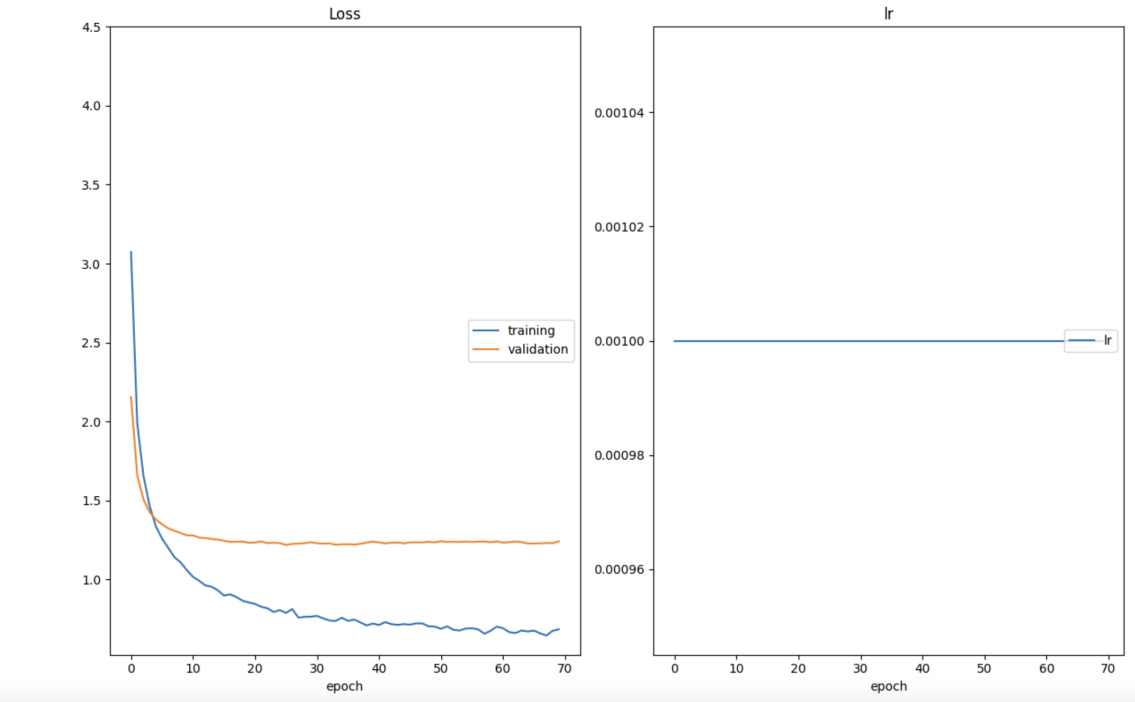

Figure 14: Loss plot of MobileNet-v3-Small model for 70 epochs

The MobileNet-Small-v3 transfer model again performed significantly better compared to our custom CNN and performed almost identically to VGG-16 and slightly better than ResNet-18 for this task. This model was able to achieve a test accuracy of 73 percent on our dataset. Also, a minimum validation loss of 1.11 was observed for this model.

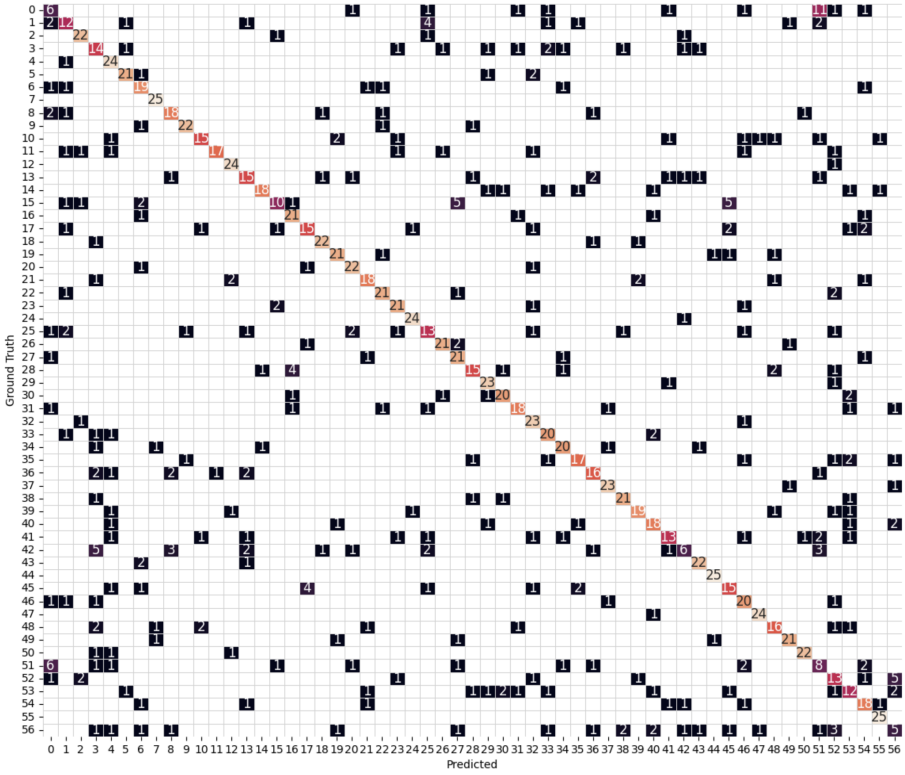

Figure 15: Confusion matrix for MobileNet-Small-v3 after 70 epochs

Confusion matrix Evaluation Upon inspecting the confusion matrix, we observed:

- MobileNet-Small-v3 performed great for Label 55: Harvard, 44: Trevi Fountain, 7: Stonehenge (25/25 correct), label 4: Wroclaw Dwarves, 12: Katanagar Temple, 24: Soreq Cave, 47: Prague Astro Clock (24/25 correct) and labels 29: Petronas Tower, 32: Hanging Temple, 37: Atomium (23/25 correct).
- The model performed worse for label 56: Kazan Tatarstan (5/25), label 42: Death Valley (6/25), label 0: Haleakala National Park (6/25). For label 3: Dead Sea, it performed as well as VGG-16 (14/25); much better than ResNet-18 for this (3/25)
- **For labels that the base model did well for**: MobileNet performed very well for these labels. Label 44: Trevi Fountain (25/25 correct), Label 24: Soreq Cave, 47: Prague Astro Clock, 12: Katanagar Temple (24/25 correct), Label 50: Noratus Cemetry Armenia (22/25 correct) and label 49: Temple of Olympian Zeus (21/25 correct)
- **For labels that the base model struggled with**: Mixed results were observed for these problematic labels. Label 0: Haleakala National Park (6/25 correct), label 51: Maui (8/25 correct), label 52: Lviv (13/25 correct), label 30: Brooklyn Bridge (20/25) correct. Similar performance was observed as compared to VGG-16 for these labels.

3.6 Insights based on the Confusion matrices

Label 56: Kazan Tatarstan , Label 0: Haleakala National Park seemed to be the two labels for which all our models (including the base and transfer learnt) performed terribly. The base model performed

struggled with Labels such as Maui, lviv and Brooklyn Bridge as well, but the transfer learning models performed much better for these labels for the most part.

Also, all models performed great for Labels such as Soreq cave, Trevi Fountain, Prague Astronomical Clock, Temple of Olympian Zeus, Noratan Cemetery Armenia, Kantanagar Temple.

3.7 Insights based on the Loss Function

Model	Test Accuracy	Loss
Resnet-18	70% (1011/1425)	1.177942
VGG-16	73% (1044/1425)	1.051462
MobileNet-Small-v3	73% (1046/1425)	1.1160761040189993
Custom	51% (727/1425)	2.078365

Figure 16: Loss Function / Accuracy comparisons for all models

The base model performed significantly poorly compared to the transfer learning models, but that was to be expected given the resource constraints. The loss function stats for all transfer learning models were comparable to each other. However, we give the edge to MobileNet-Small-v3 for being able to put up excellent loss values while being a lightweight model that takes almost 4x less than VGG-16 to train for almost the same performance.

It is to be noted that our web application allows the user to choose from all 4 of these models while performing the inference task.

3.8 Distributed Training Observations

It was observed that distributed training significantly reduced the training duration by almost 50%. Each of the four models were trained for 70 epochs using two worker processes and the entire training workloads for the four models completed within 4 hours. This is a significant reduction in training duration compared to training in a single node. We tried conducting more experiments in Falcon by increasing the number of worker processes per node but it increased the training duration. For example, training a Resnet18 model for one epoch took about 5 minutes and the synchronization of the worker processes took more time as well. Increasing the number of CPU cores for each worker didn't help with this as well. We also tried using lab machines with GPU support in the department but we weren't able to setup a distributed training environment using them because of backend issues and also taking longer training times. This motivated us to use the Falcon cluster for our distributed training workloads.

3.9 Sensitivity Analysis

For sensitivity analysis, we performed a comparative study for the inference performance between the regular version of the colored landmark images and the noisy versions of these images. We noticed a prediction probability dropoff for all the noisy images versus their non-noisy counterparts.

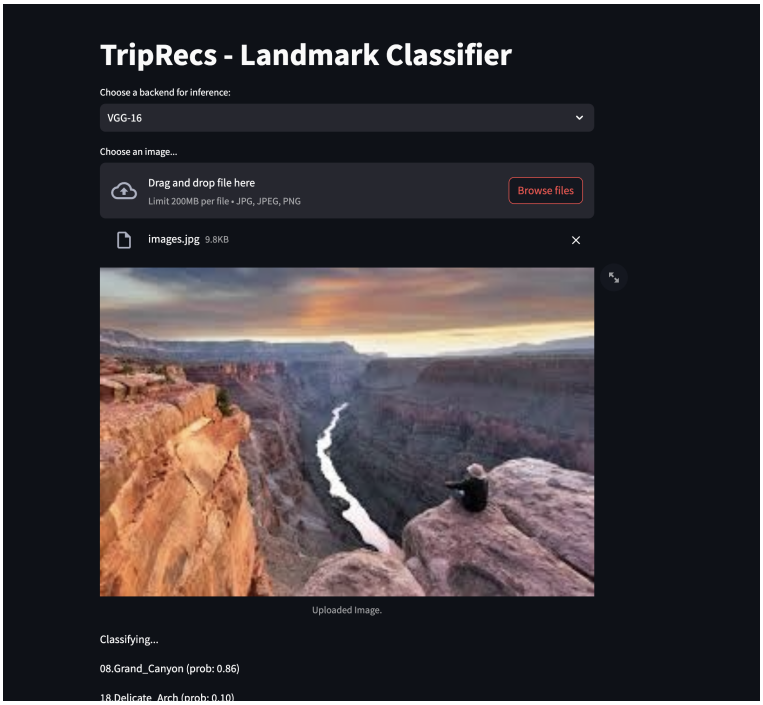

Figure 17: Grand canyon - inference (Regular image)

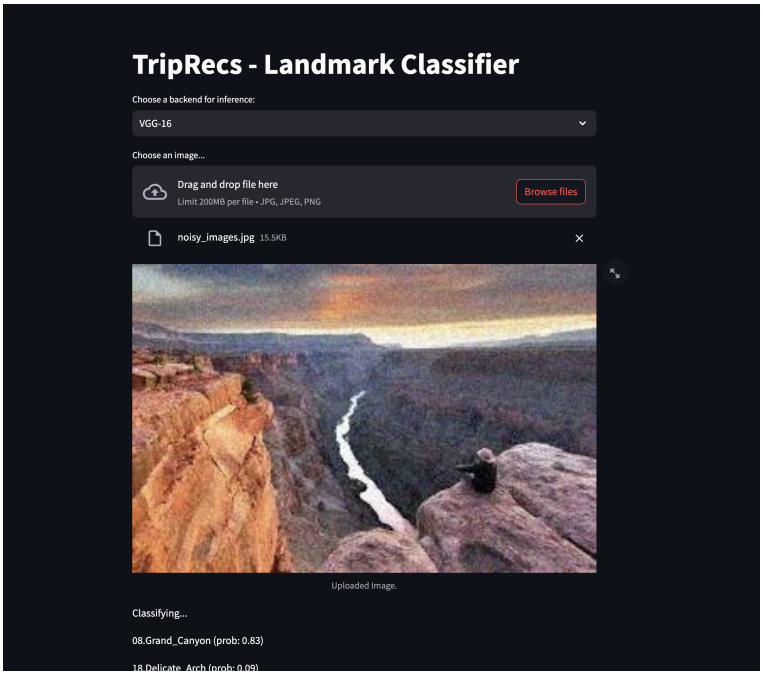

Figure 18: Grand canyon - inference (Noisy image)

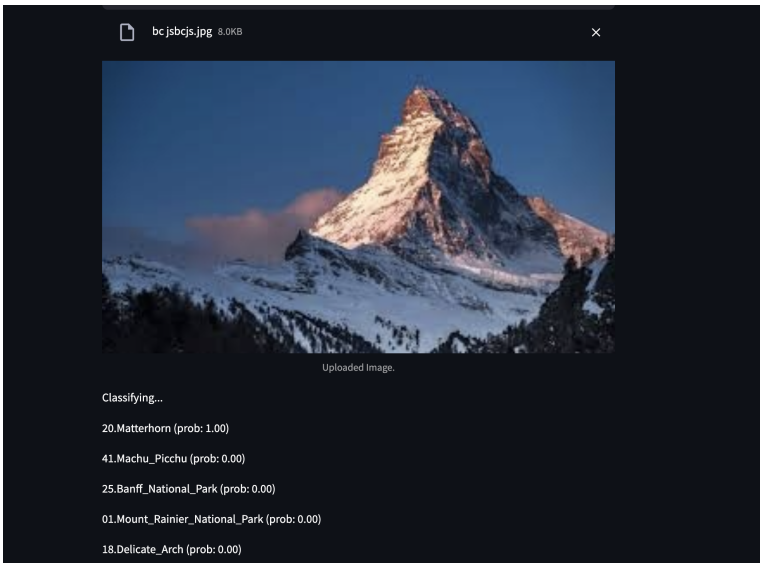

Figure 19: Matterhorn - inference (Regular image)

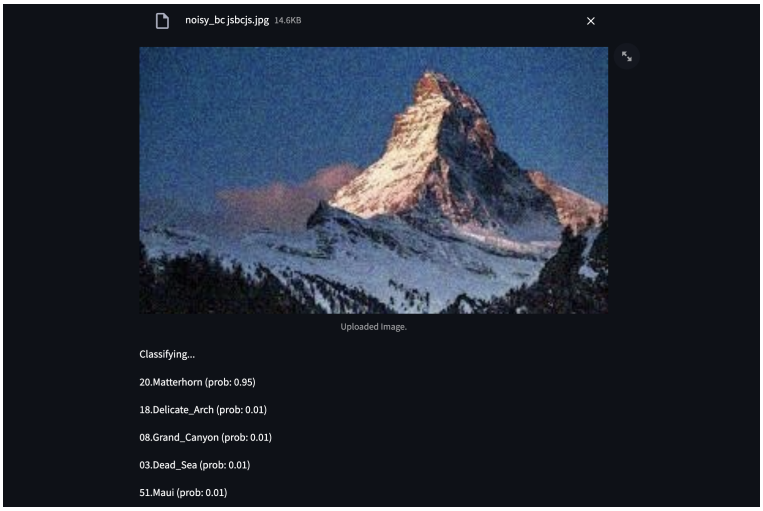

Figure 20: Matterhorn - inference (Noisy image)

3.10 Inference Examples (Base vs Transfer)

Our evaluations and the inference examples from production show that despite the fact that the base model is prone to making significantly more inference errors than the transfer models, it can perform comparable or even better than the transfer models for several image categories. Thus, we include all models we have trained as inference options for our production app.

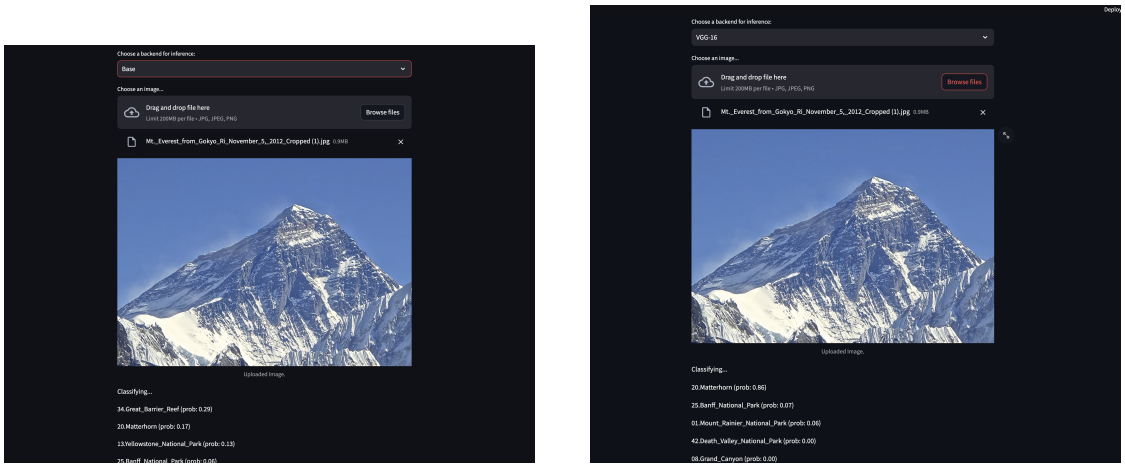

Figure 21: Inferences using base model considers Everest to be the barrier reef. The transfer model at least infers a mountain

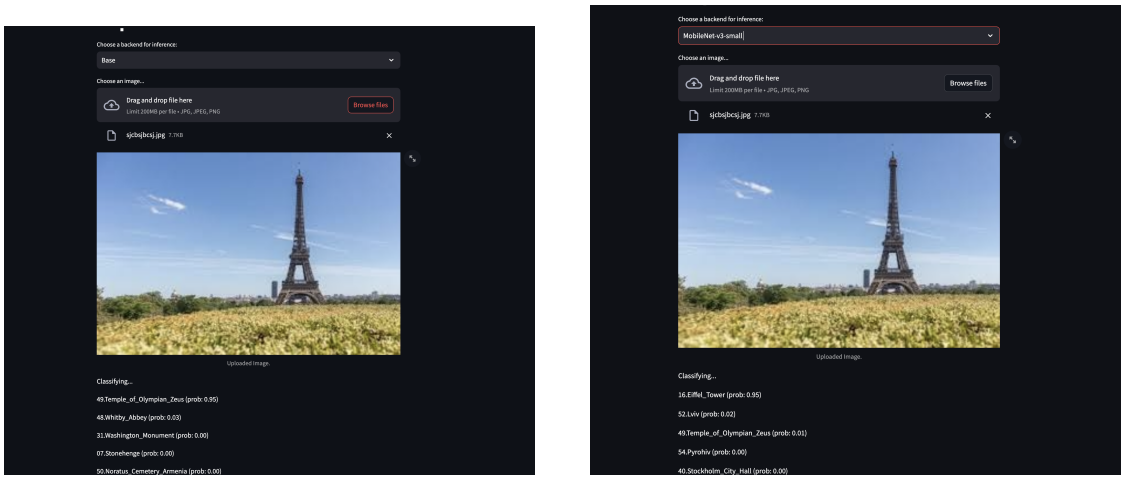

Figure 22: Base model misclassifies Eiffel tower. Transfer model is correct

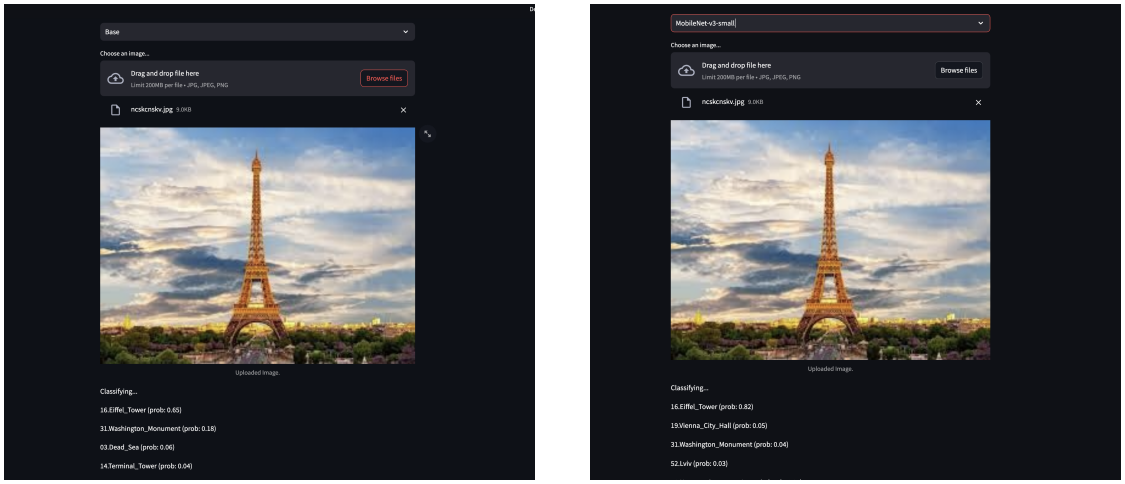

Figure 23: Both models are correct. Angles matter

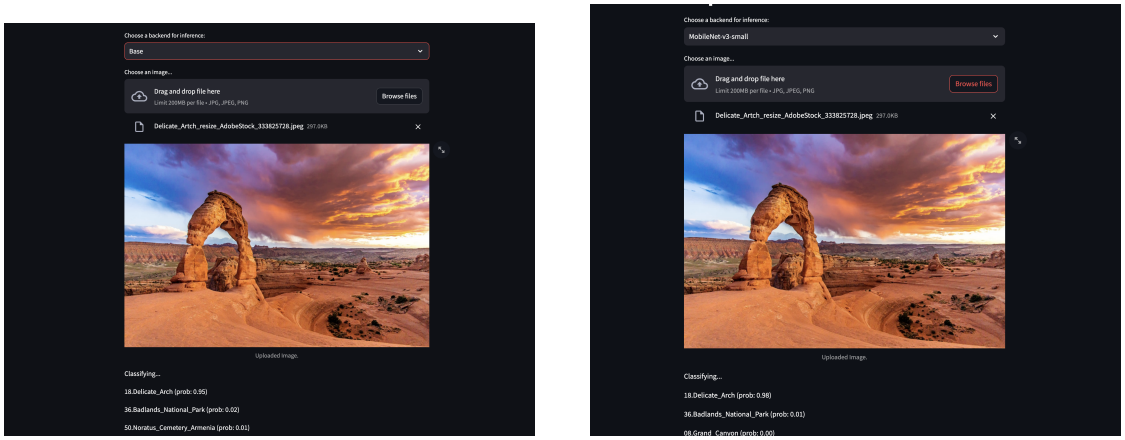

Figure 24: Both models are correct.

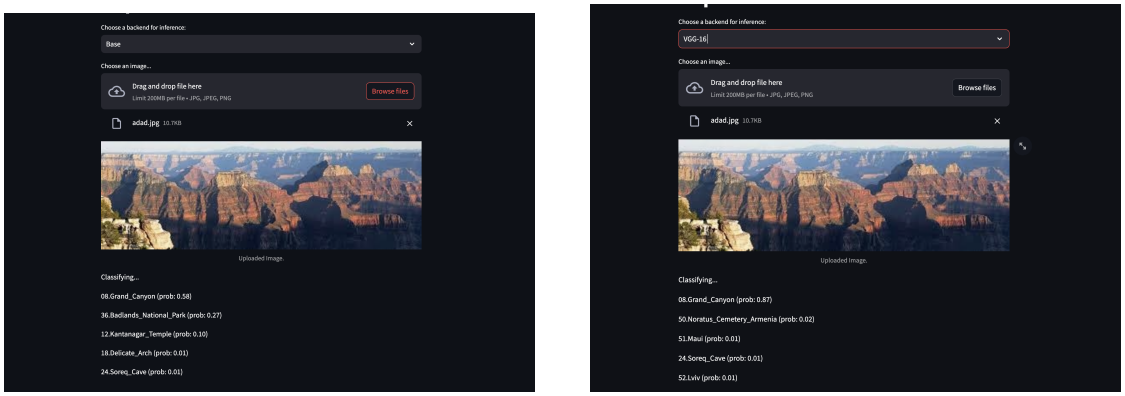

Figure 25: Both models are correct.

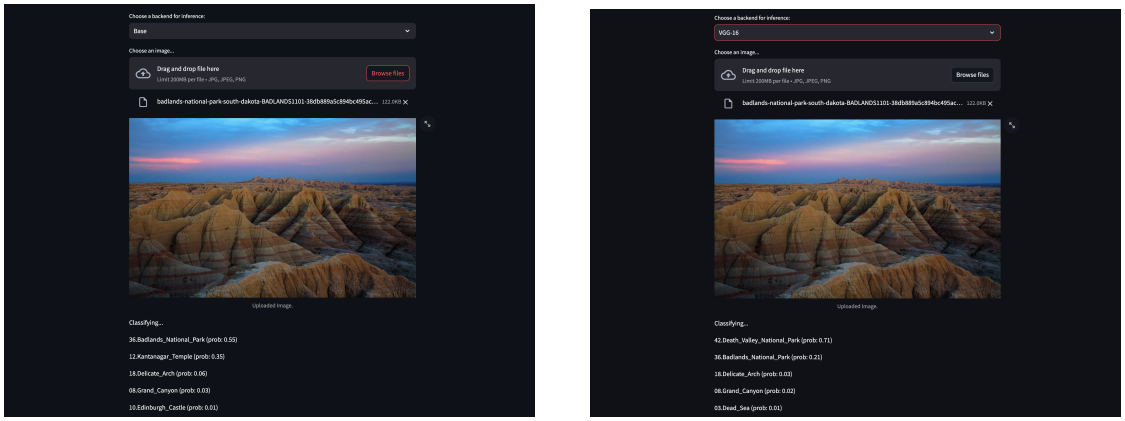

Figure 26: Base model is correct. Transfer is not

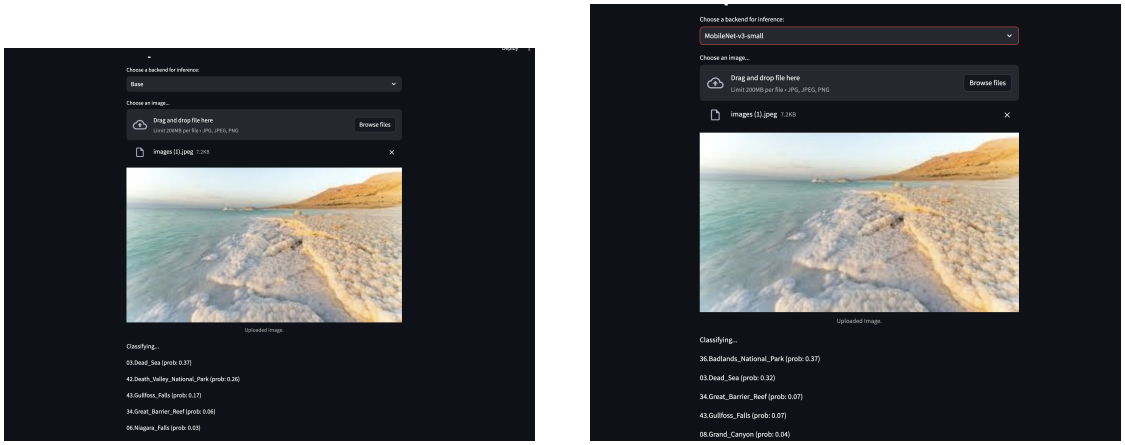

Figure 27: Base model is correct. Transfer is not

4 Contributions

Milestone	Contributor
Data Collection and Visualization	Yunik Tamrakar
Exploratory Data Analytics	Yunik Tamrakar, Dilochan Karki
Dataset Annotation, Data resampling	Yunik Tamrakar
Data Preprocessing	Yunik Tamrakar
CNN Model Code in Pytorch, Benchmarking	Yunik Tamrakar
Distributed Training Environment Setup	Dilochan Karki
Distributed Training	Dilochan Karki
Backend API Integration with trained model	Yunik Tamrakar
Recommendation logic using DNNs	Yunik Tamrakar
Streamlit UI	Yunik Tamrakar
Travel Fares Data Curation	Yunik Tamrakar
Apache Solr Core Setup, Schema Design, Queries	Yunik Tamrakar

References

[1] Alexia Anagnostopoulos and Jan Fehr. Rebound and steep increase of international travel after the covid-19 pandemic: where are we going from here? *Journal of Travel Medicine*, 31(1):taad158, 2024.

[2] Xinquan Cheng and Wenlong Su. Recommendation model of tourist attractions based on deep learning. *Mathematical Problems in Engineering*, 2022:1–7, 08 2022.

[3] Riccardo Del Chiaro, Andrew D Bagdanov, and Alberto Del Bimbo. Noisyart: A dataset for webly-supervised artwork recognition. In *VISIGRAPP (4: VISAPP)*, pages 467–475, 2019.

[4] István Fehérvári and Srikar Appalaraju. Scalable logo recognition using proxies. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 715–725. IEEE, 2019.

[5] Alexis Joly and Olivier Buisson. Logo retrieval with a contrario visual query expansion. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 581–584, 2009.

[6] Yannis Kalantidis, Lluis Garcia Pueyo, Michele Trevisiol, Roelof Van Zwol, and Yannis Avrithis. Scalable triangulation-based logo recognition. In *Proceedings of the 1st ACM international conference on multimedia retrieval*, pages 1–7, 2011.

[7] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. PyTorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13:3005–3018, 2020.

[8] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1096–1104, 2016.

[9] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465, 2017.

[10] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

[11] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof Van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM international conference on multimedia retrieval*, pages 1–8, 2011.

[12] Xiu-Shen Wei, Quan Cui, Lei Yang, Peng Wang, and Lingqiao Liu. Rpc: A large-scale retail product checkout dataset. *arXiv preprint arXiv:1901.07249*, 2019.

[13] Ke Yan, Yonghong Tian, Yaowei Wang, Wei Zeng, and Tiejun Huang. Exploiting multi-grain ranking constraints for precisely searching visually-similar vehicles. In *Proceedings of the IEEE international conference on computer vision*, pages 562–570, 2017.

[14] Dominik Zapletal and Adam Herout. Vehicle re-identification for automatic video traffic surveillance. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 25–31, 2016.