W251 Final Project

LANDMARK RECOGNITION WITH DEEP LEARNING

Deepak Nagaraj, Michael Stackhouse and Roland Lim

OVERVIEW

A. Background

Today, sharing our travel experiences and capturing memories of places we have visited are done by taking photos of locations and posting them on social media. It comes as no surprise that social media platforms such as Instagram and Facebook have been described as the "Modern-Day Travel Agent". In a recent survey done, 48% of people who want to choose their next travel destination go to Instagram and 35% use Instagram to discover new places.

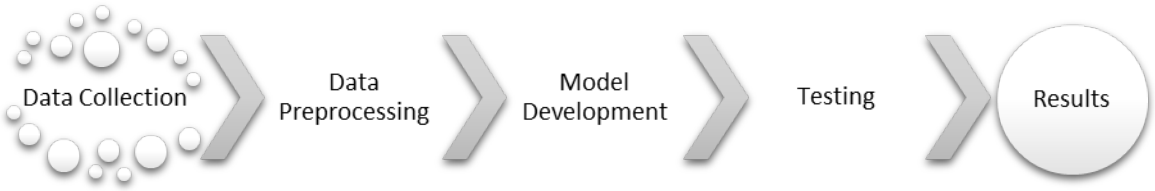
A quick look at Instagram shows many incredible photos of landmarks associated with various cities. While some landmarks are world-famous, some are not nearly as well-known. Searching for the name and location of a monument that people are posing in front of can actually be time-consuming, especially without having more information other than the picture itself.

In addition, our own vacation photos may contain pictures of buildings and places whose names we cannot fully remember (or spell), which can make organizing and categorizing our personal photo collections more difficult than it is.

This project aims to develop a landmark recognition tool using deep learning not only to address the issues mentioned above, but to also serve as a building block to meet the growing number of applications with this tool – from autonomous driving to visual search engine purposes.

B. Objectives

The overall objective is to train and test Deep Learning models to recognize well-known landmarks based on images using the approach below:



In the process, the project will require:

- Preparing a dataset that mimics real-world photos e.g. blurred, grayscale, unknown landmarks/classes outside scope
- Testing different network architectures using transfer learning and comparing results

The target of our model will be for deployment on an edge device, such as a smartphone. This would allow a user to identify landmarks with their device locally, without reaching back out to the cloud and relying on a stable connection. This would be beneficial for travelers without unlocked phones, or in remote areas where a cellular network may not be strong enough to support the data transfer necessary for inference on the cloud.

DATA COLLECTION

To select our dataset, we examined two potential data sources:

A. Google Landmark Recognition challenge on Kaggle

There is a Google Landmark Recognition Challenge on Kaggle to build a model to correctly classify landmarks. The training data alone contains over 1.2 million images with about 15,000 classes. However, a quick inspection revealed 40% of classes having less than ten images each. In fact, many classes contain only one image. Within the classes, there is also a large fraction not showing the landmarks at all.

Given the extreme class imbalance and the huge dataset involved, which would then require additional computing resources, we decided to use another dataset instead.

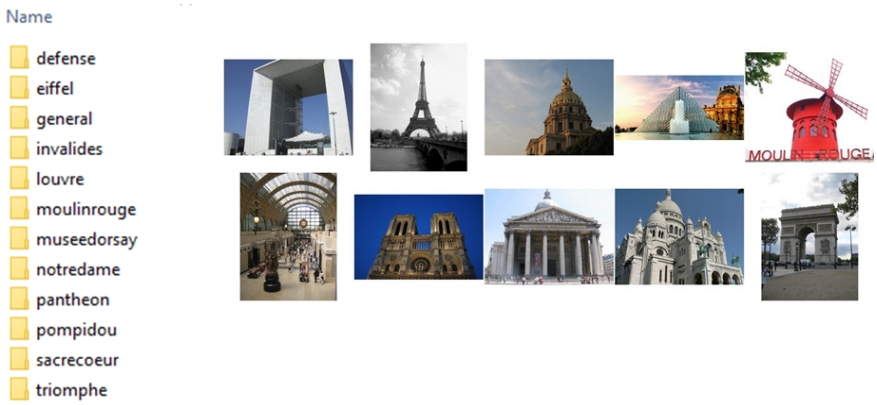
B. The Paris 6K Dataset

The Paris Dataset was compiled by the University of Oxford. It contains 6,412 images of Paris landmarks collected from Flickr. The landmarks are as follows:

- La Defense Paris
- Eiffel Tower Paris
- Hotel des Invalides Paris
- Louvre Paris
- Moulin Rouge Paris
- Musee d'Orsay Paris
- Notre Dame Paris
- Pantheon Paris
- Pompidou Paris
- Sacre Coeur Paris
- Arc de Triomphe Paris
- Paris (general images all over Paris)

When downloading the data, every image is saved in a folder named after the landmark/class. The folder structure and sample images are shown below:

Folder Structure of The Paris 6K Dataset and Sample Images



DATA PREPROCESSING

A. Cleaning the Data

The Paris dataset included "groundtruth" files to describe the quality of the image, based on the following criteria:

- *Good:* A nice, clear picture of the object/building
- *OK:* More than 25% of the object is clearly visible
- *Bad:* The object is not present
- *Junk:* Less than 25% of the object is visible, or there are very high levels of occlusion or distortion

In order to have our model to accurately recognize the images, we decided to remove the bad and junk images (see below). However, this resulted to a reduction in the size of our dataset from 6,412 to 2,014 images.

Sample of Bad and Junk Images

B. Query Additional Images

With the significant reduction in images after removing images classified as bad or junk, some classes were left with a significant imbalance. To overcome this imbalance, we supplemented those classes using a tool called Google Images Download. This tool allows you to keyword search google images and automatically download a desired number of images from the search results.

The three classes that were supplemented were the Eiffel Tower, Musée d'Orsay, and The Centre Pompidou. Around 75 additional images were added to each class as supplements.

C. Image Augmentation

To further increase the size of our dataset and mimic real-world images of landmarks that people upload to social media platforms, we performed the following image augmentation techniques using PIL / Python Image Library:

Horizontal Flip / Mirror Image:

Sample:

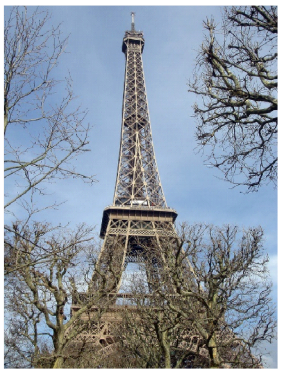
Code:

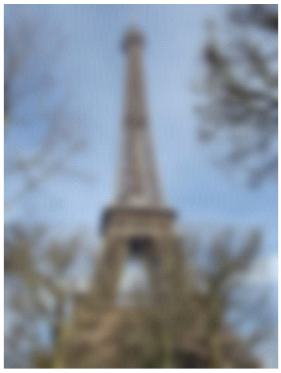
```
import os
from PIL import Image
from pathlib import Path

image_folder = "<PATH TO FOLDER>"
for image_filename in os.listdir(image_folder):
    image = Path(image_folder + image_filename)
    img = Image.open(image)
    # Mirror image and save the image with the same filename
    img.transpose(Image.FLIP_LEFT_RIGHT).save(image)
    #img.rotate(180).save(image)
    # close the image
    img.close()
```

Blurred

Sample:





Code:

```
image_folder = "<PATH TO FOLDER>"
for image_filename in os.listdir(image_folder):
    image = Path(image_folder + image_filename)
    img = Image.open(image)
    # Blur
    img.filter(ImageFilter.GaussianBlur(radius=12)).save(image)
    # close the image
    img.close()
```

Grayscale

Sample:

Code:

```
import os
from PIL import Image
from PIL import ImageFilter
from pathlib import Path

image_folder = "<PATH TO FOLDER>"
for image_filename in os.listdir(image_folder):
    image = Path(image_folder + image_filename)
    img = Image.open(image)
    # Convert to Grayscale
    img.convert('L').save(image)
    # close the image
    img.close()
```

D. Distractor Images

One major issue with the Paris 6K Dataset is the distraction images. Distraction images serve to give the model a class to utilize when no other classes are successfully matched. This is useful, as when an image is passed to our model for inference and the model has not been trained to recognize it, a response of "not recognized" would be more beneficial than a best guess from the landmarks for which it has been trained.

The "general" folder in the Paris 6K failed to do this properly. The majority of the images within the folder actually belong to other classes. For example, over half of the photos were of the Eiffel Tower. Upon initial training of the model using the original dataset, we noticed that this was in fact confusing the model, and in many cases photos of the Eiffel Tower were misclassified as "General".

The resolve this issue, we sorted the photos from within the "general" folder into their proper labels, and supplemented the "general" folder with landmarks from cities other than Paris, such as Philadelphia, Los Angeles, New York, and Chicago.

MODEL DEVELOPMENT

To develop our model, we leveraged methods from two different sources:

- Tensorflow for Poets
- NVIDIA DIGITS

Tensorflow for Poets targeted exactly what we needed to accomplish our goal. The code was already setup to easily apply transfer learning. Additionally, it comes pre-equipped to apply some image augmentation methods at the time of training. The major benefit of Tensorflow for Poets is that it utilizes bottleknecking to accelerate model retraining. This is significant, as by only retraining the last layer of the model, a large amount of time and computational resources can be saved. The downside of tensorflow for poets is that it unfortunately is only set up to use MobileNet and Inception V3.

To test some additional models, we also used transfer learning with DIGITS. This allowed us to test AlexNet, and VGG16. DIGITS unfortunately does not currently have the capability of bottleknecking, so

the entire model needs to be retrained on our dataset. While this results in a longer run time, since we are using a model that is already pretrained, we can significantly decrease the number of epochs used compared to what would be required if we were training a fresh model - so we trained on 3 epochs and the models did successfully converge.

RESULTS

A. Prediction Accuracies and Discussion - DIGITS

As mentioned in the earlier section, one of the best prediction results came from the pretrained VGG16 model that we ran on DIGITS.

For photos where the object is in-focus and clear, the VGG16 model correctly classified the landmark with a high degree of confidence (>95%).

La Defense

Predictions

defense	98.44%
general	0.71%
pantheon	0.32%
triomphe	0.14%
eiffel	0.13%

Centre Pompidou

Predictions

pompidou	98.07%
notredame	0.46%
triomphe	0.41%
pantheon	0.35%
general	0.29%

Sacre Coeur

Predictions

sacrecoeur	97.0%
invalides	2.66%
general	0.16%
notredame	0.12%
pompidou	0.02%

Photos with people or other objects in the background were also classified correctly with high confidence as long as the landmark is evident and clear. For example, in the Musee d'Orsay sample below, the clock, which is featured in almost all Musee d'Orsay photos, is clearly seen and this likely contributed to the high prediction score (>99%). The same is true with the Moulin Rouge photo where distinct features of the landmark are prominent in the photo resulting to the high accuracy score even if other objects are present.

Musee d'Orsay (with a person in front and only the Musee d'Orsay clock seen)

Predictions

museedorsay	99.93%
pompidou	0.03%
notredame	0.02%
triomphe	0.01%
moulinrouge	0.0%

Moulin Rouge (with a person and vehicles in the photo)

Predictions

moulinrouge	98.34%
pompidou	0.9%
general	0.62%
triomphe	0.05%
eiffel	0.04%

The model also correctly classified photos taken from a distance, cropped images, and black-and-white photos provided there is a clear and unobstructed view of the landmark.

Louvre Museum (from a distance)

Predictions

louvre	79.17%
invalides	7.88%
eiffel	3.08%
general	2.71%
defense	1.91%

Arc de Triomphe (cropped; only one pillar in the photo)

Predictions

triomphe	95.22%
defense	2.05%
louvre	1.17%
pompidou	0.59%
pantheon	0.27%

Pantheon (black-and-white)

Predictions

pantheon	82.85%
triomphe	4.66%
general	4.08%
defense	2.67%
invalides	1.79%

Photos with excessive blurring and/or distortion were also correctly classified but with lower prediction scores.

Notre-Dame de Paris (blurred)

Predictions

notredame	62.99%
triomphe	8.24%
louvre	4.16%
sacrecoeur	3.8%
moulinrouge	3.72%

Eiffel Tower (blurred)

Predictions

eiffel	43.54%
invalides	25.79%
louvre	14.8%
pompidou	4.07%
defense	3.23%

Eiffel Tower (distorted)

Predictions

eiffel	40.93%
invalides	38.4%
louvre	10.08%
general	3.94%
pantheon	1.8%

Even photos containing non-Paris landmarks were classified correctly as “general” but with lower confidence.

Brooklyn Bridge (New York)

Predictions

general	48.25%
defense	17.34%
pantheon	11.48%
triomphe	5.9%
eiffel	5.61%

Staples Center (Los Angeles)

Predictions

general	42.02%
moulinrouge	25.27%
triomphe	9.43%
eiffel	9.43%
pompidou	5.71%

Capitol Records Building (Los Angeles)

Predictions

general	38.78%
eiffel	25.28%
pompidou	11.82%
moulinrouge	8.68%
pantheon	7.13%

Universal Studios Globe (Los Angeles)

Predictions

moulinrouge	33.8%
general	33.32%
eiffel	9.36%
notredame	5.79%
louvre	4.49%

Apart from non-Paris landmarks contributing to lower prediction scores, the following factors would have also affected the model's performance:

- It was observed that the model would often confuse Dome des Invalides with Sacre Coeur. This can be explained by the fact that both landmarks are churches built in the 19th century with relatively similar design (dome structure)

Sacre Coeur (misclassified as Dome des Invalides)

Predictions

invalides	56.9%
sacrecoeur	40.17%
notredame	1.34%
general	1.25%
louvre	0.1%

- In cases where other objects appear larger and more prominent in the photo than the landmark itself, this also resulted in the misclassification of images.

Eiffel Tower (misclassified as La Defense)

Predictions

defense	27.53%
general	22.99%
louvre	14.11%
eiffel	11.6%
triomphe	11.07%

Eiffel Tower (misclassified as Arc de Triomphe)

Predictions

triomphe	52.95%
eiffel	11.39%
general	7.8%
invalides	6.86%
defense	6.23%

B. Prediction Accuracies and Discussion - Tensorflow for Poets

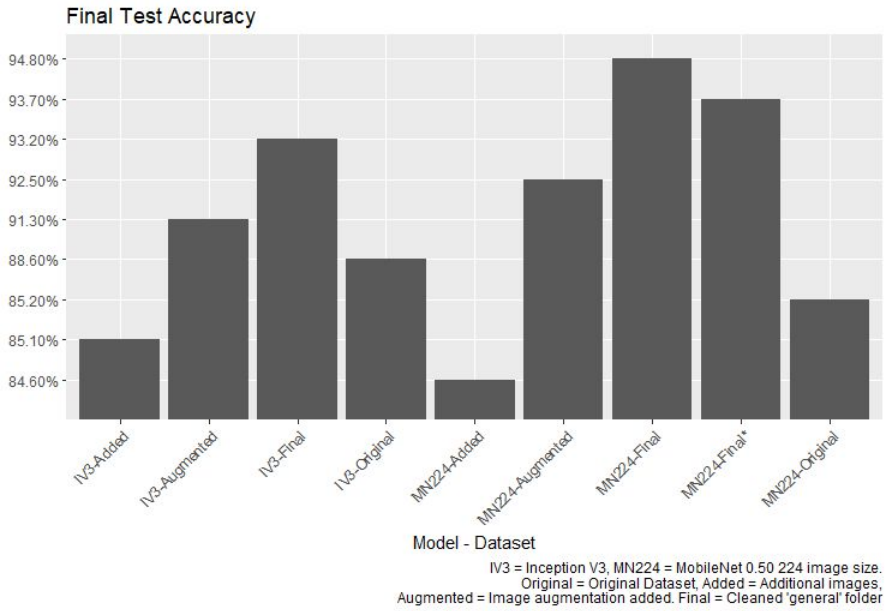
Using Tensorflow for Poets as our base, we were able to obtain even higher accuracy than AlexNet and VGG16 (trained on 3 epochs). The models we were able to test included MobileNet 0.50 and Inception V3. Using bottleknecking, training times for these models were kept significantly low. Using the NVIDIA Jetson TX2, both of these models completed 4000 steps in under 20 minutes - and the resulting accuracy was quite impressive.

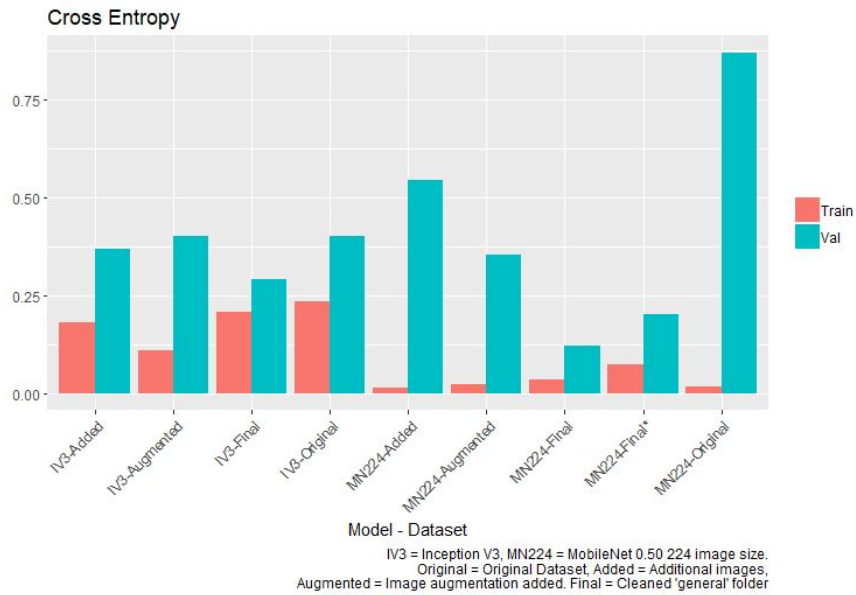
Training was done in a few stages:

- Original dataset (with poor quality images removed)
- Supplemented dataset (balancing classes lacking images)
- Augmented dataset (image augmentation applied)
- Final dataset (with the issues discussed about distractor images resolved)

Most findings were as expected; the performance of the model improved with image augmentation, Inception V3 performed better initially than MobileNet. But other findings were rather surprising. Balancing out weak classes actually decreased the model performance. This could be that the model started overfitting those classes - or the benefit may not have been significant in the first place because most accuracy of the model is provided by the bottlenecked layers in the first place. Additionally, once image augmentation was applied, MobileNet outperformed Inception V3. Perhaps MobileNet's smaller graph assists in preventing overfitting - especially when we have the smaller dataset that is being used to retrain these final layers. Lastly, Tensorflow for Poets has the option to apply image augmentation at runtime. In the charts below, this is marked by *MN224-Final**. This runtime image augmentation decreased performance by 1% - but also negated most of the benefits of the bottleknecking, as augmented images needed to be run through the full graph again. This ultimately pushed training time over 7 hours.

Ultimately, the best performing model was MobileNet trained on the final dataset - with test set accuracy topping out at 94.8%.



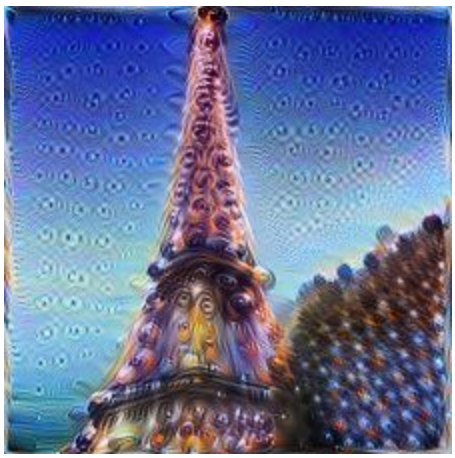


C. Neural Net Visualization

To understand what is happening at the neural network, it is helpful to visualize the image as it passes through each layer. We used a Lucid notebook to look at the pictures as they passed through each layer of the network.

Here are some pictures of Eiffel Tower as it transits through some layers of Inception v1.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

conv2d0	
conv2d1	
mixed3b	

mixed4b	
mixed4e	
mixed5b	

Given this knowledge, we can compare pictures to get a sense of why the neural network classified an image the way it did.

For example, the following picture of Dome des Invalides is misclassified as Sacre Coeur.

Dome des Invalides ...	... misclassified as Sacre Coeur

It is hard to say how the classifier could have made a mistake with the picture.

Let us look at the corresponding pictures as seen by the neural net. The Sacre Coeur score was 0.48 for the misclassified Invalides, so we choose a true Sacre Coeur picture that has a similar score (0.61).

They look eerily similar. The street lights are comparable to the windows, and the convolutions, pooling and mixing have morphed the sky into shapes comparable to the domes.

We can use such visualizations to explore how augmentation affects the images as well.

CONCLUSION AND NEXT STEPS

Ultimately, we determined MobileNet to be our model of choice. As stated earlier, with the target of edge inference, our model needs to be fairly lightweight. MobileNet's smaller architecture keeps runtime short and the files size is quite small (~5.5 MB). This is despite MobileNet actually not being our highest performing model. Trained on 30 epochs (using a P100 GPU in the cloud), VGG16 achieved accuracy of 97% - but this came with some different problems. VGG16 is a significantly larger model, and despite higher accuracy, we noticed some new overfitting challenges that would need to be addressed.

Sacre Coeur (now classified correctly)

Predictions

sacrecoeur	100.0%
general	0.0%
notredame	0.0%
invalides	0.0%
triomphe	0.0%

Death Star (misclassified as a Paris landmark)

Predictions

pompidou	79.85%
louvre	12.54%
general	6.91%
defense	0.55%
pantheon	0.12%

As for next steps, NVIDIA's transfer learning toolkit offers some cutting edge techniques that would significantly benefit our model application. By pruning our model, even more of the dead weight could be cut off, leading to even more efficient inference and better deployment for an edge device.

Another issue identified in our model is that it performs poorly on landmarks photographed from a distance. This is reasonable given that our model is performing object classification; if the object we're trying to classify is a very small part of the picture, the other aspects of the image could easily confuse the model. Taking this into consideration, one approach would be to change this from classification to object detection. While the model would be more complex and heavier, it would allow users to identify potentially multiple landmarks within an image, and recognize them from afar.

The next challenge would be expanding out the number of landmarks the model can classify. The most obvious challenge here is data: the amount of data to classify landmarks in even just major cities would be massive. But a second, more subtle challenge would be overfitting. In our small sample, we noticed some misclassification of Hotel des Invalides Paris with Sacre Coeur Paris and vice versa. Both of these buildings share a similar style of architecture, so it is reasonable that the model may confuse them With

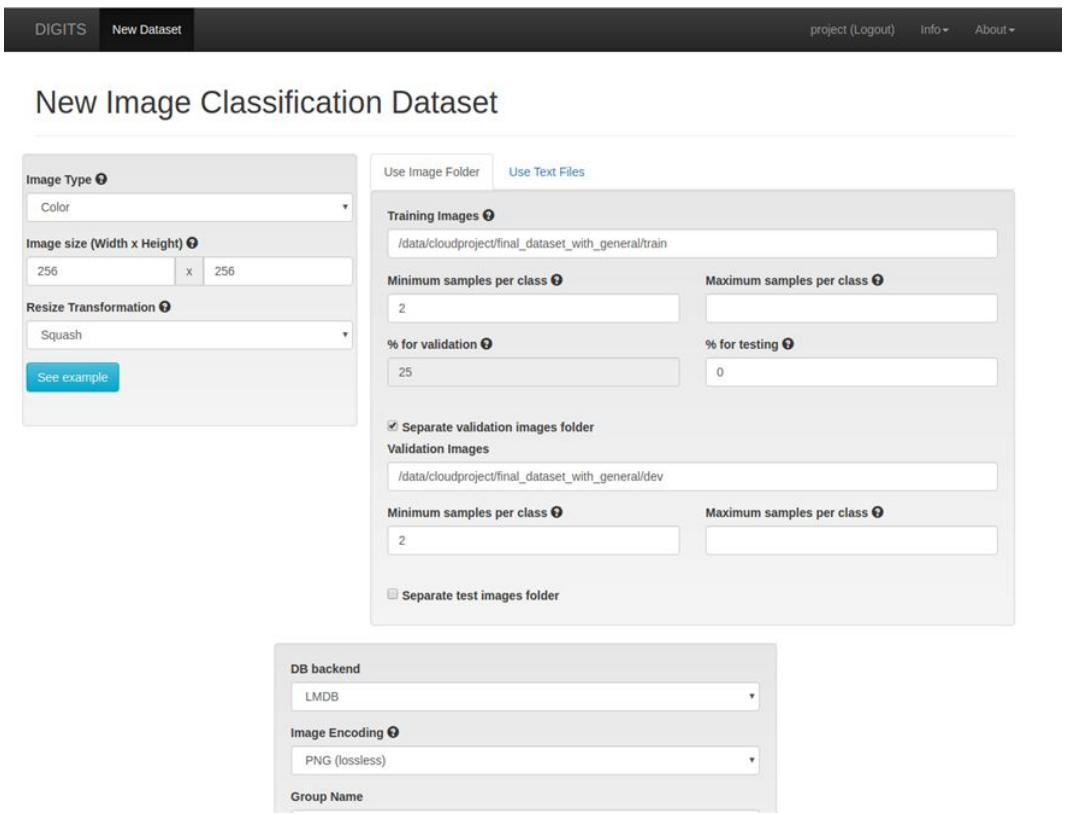
the addition of more landmarks in more cities, this problem would be exacerbated - as to prevent overfitting, the model needs to generalize these classes which could make it easier to confused similar looking objects. A simple solution would be to integrate location data with the classifications, and other options for training and overfitting would need to be explored. Perhaps at this stage, a different model architecture would need to be considered.

APPENDICES

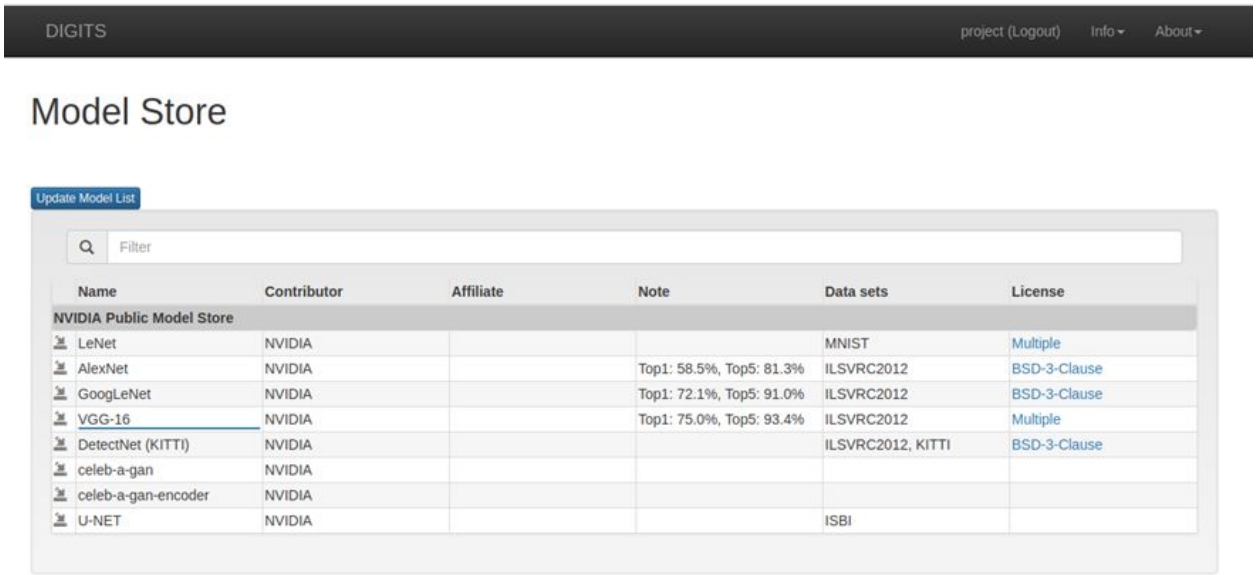
Github Repository Link: https://github.com/deepix/w251finalproject.git

Steps taken to use and deploy the Pretrained VGG16 model on Digits

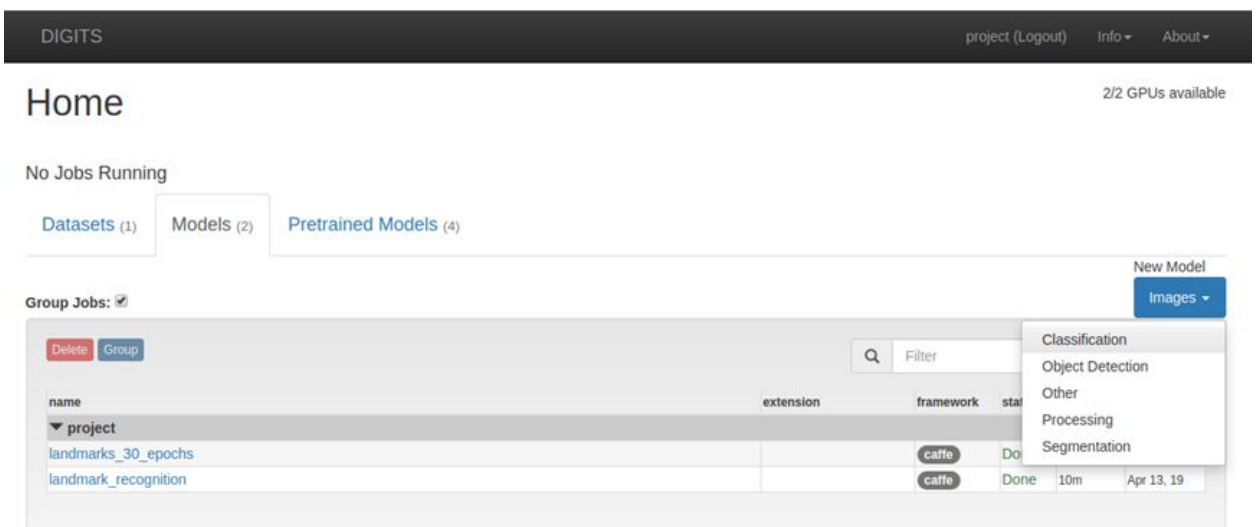
- Select "Dataset" tab
- Select Classification
- Input path to the train, validation and test folders
- Input the group name, dataset name



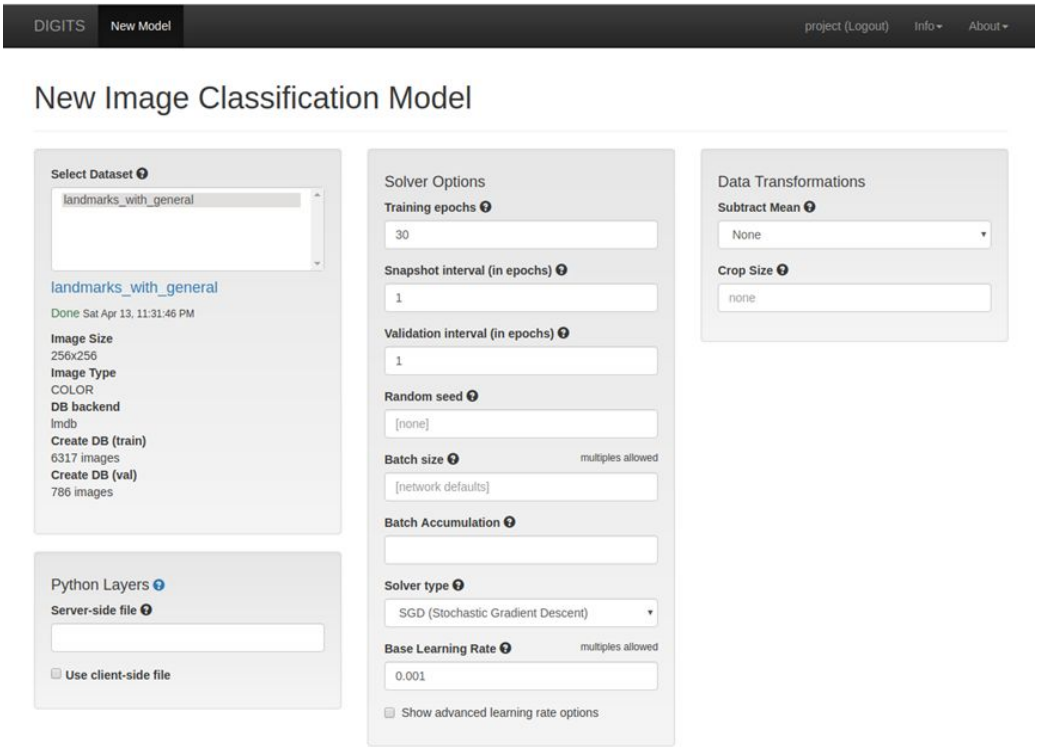
- Click Create
- Once dataset has been created, go back to the main page
- Select "Pretrained Models" tab
- Select Retrieve from Model Store
- Click the download icon beside the desired pretrained model i.e. VGG16



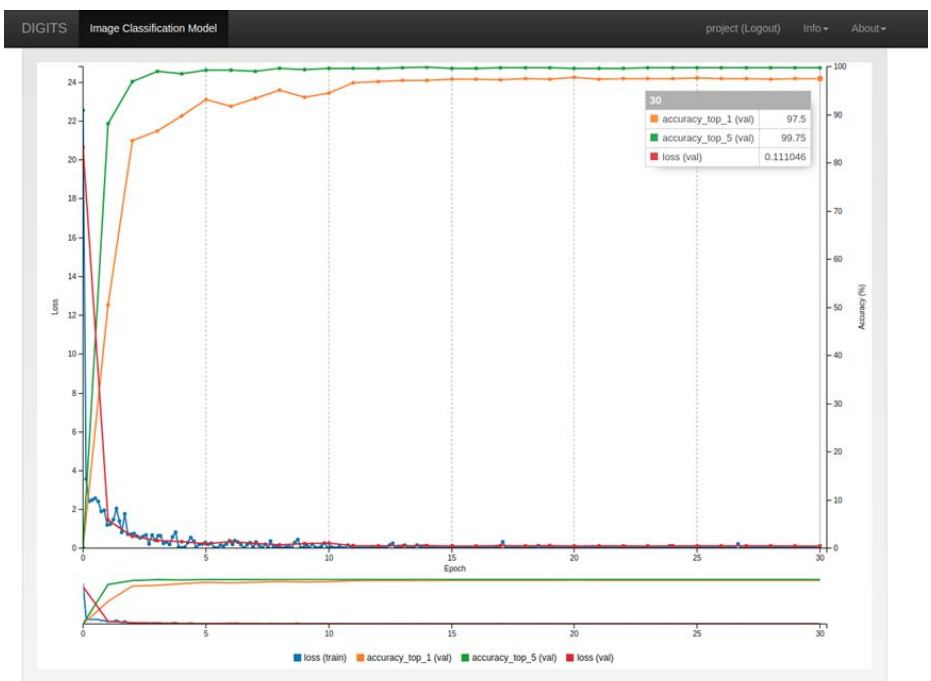
- Once model has been downloaded, go back to the main page
- Select “Models” tab
- Select Classification



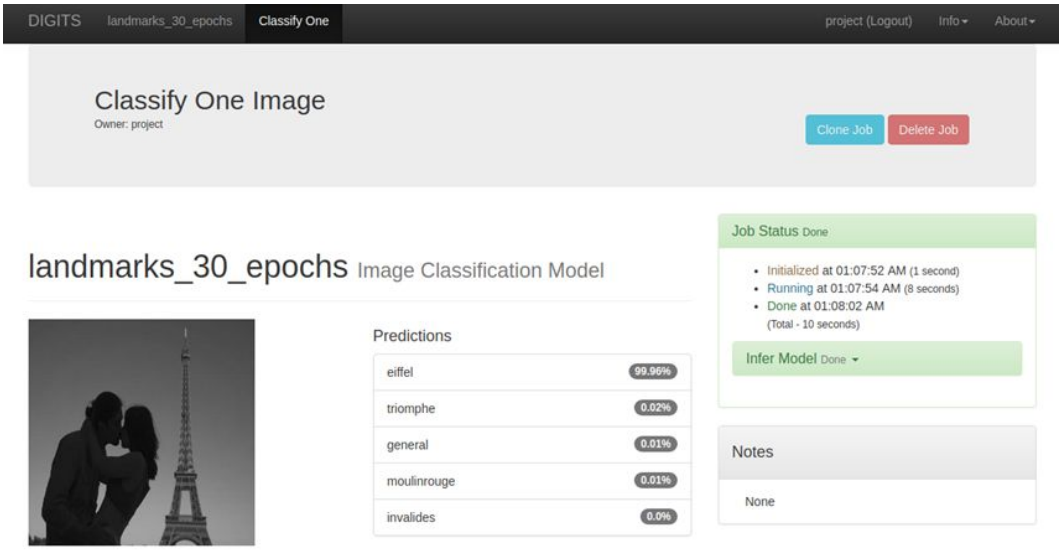
- Select the dataset
- Set Subtract Mean to None



- Select the Pretrained Networks tab
- Select the desired downloaded pretrained VGG16 network
- Input the group name, model name
- Click Create
- Examine the results



- Test the model



Steps taken to use and deploy MobileNet 0.50 on Tensorflow for Poets

Note: These instructions assume that the necessary software is installed, or an adequate docker container is being used. Setup instructions may also be followed here: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/

- In the git repository, unpack the tf_for_poets.tar.gz file:

```
tar -xzf tf_for_poets.tar.gz
```

- Copy the contents of the `selected_images_everything` folder into the folder `tensorflow-for-poets-2/tf_files/paris_augment`
- To train the model - use the following command:

```
python -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks/final_model \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/mobilenet_0.50_224/ \
  --output_graph=tf_files/final_model.pb \
  --output_labels=tf_files/classes.txt \
  --architecture=mobilenet_0.50_224 \
  --image_dir=tf_files/paris_augment
```

- To use the model for inference:

```
python -m scripts.label_image \
    --graph=tf_files/models/final_model.pb \
    --labels=tf_files/models/classes.txt \
    --image=tf_files/paris_augment/<class>/<image filename>
```

- Some inference examples:

```
python -m scripts.label_image \
    --graph=tf_files/models/final_model.pb \
    --labels=tf_files/models/classes.txt \
    --image=tf_files/paris_augment/notredame/paris_notredame_000342.jpg

Evaluation time (1-image): 4.625s

notredame (score=0.95711)
general (score=0.03010)
defense (score=0.00897)
invalides (score=0.00277)
eiffel (score=0.00060)


python -m scripts.label_image \
    --graph=tf_files/models/final_model.pb \
    --labels=tf_files/models/classes.txt \
    --image=tf_files/paris_augment/pantheon/paris_pantheon_000662.jpg

Evaluation time (1-image): 4.647s

pantheon (score=0.99999)
general (score=0.00001)
louvre (score=0.00000)
sacrecoeur (score=0.00000)
invalides (score=0.00000)


python -m scripts.label_image \
    --graph=tf_files/models/final_model.pb \
    --labels=tf_files/models/classes.txt \
    --image=tf_files/paris_augment/louvre/paris_louvre_000235.jpg

Evaluation time (1-image): 4.884s

louvre (score=0.99975)
invalides (score=0.00017)
eiffel (score=0.00004)
notredame (score=0.00002)
general (score=0.00001)
```

REFERENCES:

http://www.robots.ox.ac.uk/~vgg/data/parisbuildings/

https://landmarksworkshop.github.io/CVPRW2018/

https://www.kaggle.com/c/landmark-retrieval-challenge

https://www.kaggle.com/c/landmark-recognition-challenge

https://www.trekksoft.com/en/blog/how-instagram-has-changed-the-tourism-and-travel-industry

https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/