Machine Learning Engineer Nanodegree

Capstone Project

"Development of a CNN model for Google Landmark Recognition Challenge using a reduced dataset"

Gap Kim
June 15, 2018

I. Definition

Project Overview

Over the last years, deep learning methods have been shown to outperform previous state-of-the-art machine learning techniques in many fields such as visual, audio, medical, social, and sensor. In particular, object recognition has gained tremendous interest by engineers and scientists in artificial intelligence and computer vision. Deep learning allows computational models of multiple processing layers to learn and represent data with multiple levels of abstraction mimicking how the brain perceives and understands multimodal information, thus implicitly capturing intricate structures of large-scale data [1]. Among various methods developed in object recognition, Convolutional Neural Network (CNN) is of interest for this project. CNNs were inspired by the structure of a visual system and were shown to significantly outperform traditional machine learning approaches in computer vision and pattern recognition [2]. CNNs have been used in variety of fields, which includes but are not limited to object detection [3], face recognition [4], and action/activity recognition [5].

The modern history of object recognition goes along with the development of CNN when AlexNet [6] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a wide margin with top-5 error of 16.4%. In 2013, ZFNet [7] made minor modification of AlexNet using 7x7 kernel and resulted in top-5 error of 11.7%. In 2015, GoogleNet/Inception V1 [8] achieved top-5 error of 6.67%, which was very close to human level of performance. In 2015, ResNet [9] achieved 3.57% top-5 error outperforming human performance.

In this project, a deep learning model based on CNNs is proposed for Google Landmark Recognition Challenge from Kaggle [10]. The Landmark Recognition Challenge presents a dataset with a very large numbers of classes (~15,000 classes), but the number of training examples per class may not be very large. This makes the problem different from image classification challenges like the ILSVRC where the aim is to recognize 1000 general object categories. Due to computational limitation, however, a reduced dataset is proposed with 100 unique landmark ids in approximately 8,100 images. Stratified sampling is employed so that the ratios of images among the classes are preserved.

Problem Statement

One of great obstacles to landmark recognition research is the lack of large annotated datasets. Hence Google Landmark Recognition Competition presents the largest worldwide dataset to date and challenges to build models that recognize the correct landmarks from the test images. A technology that can accurately predict landmark labels directly from image pixels can broadly benefit applications in various areas such as photo management, maps, aviation, and satellite images.

Evaluation Metric

The accuracy is used as the overall evaluation metric when comparing the performance of the benchmark model and the solution model. The accuracy can be calculated by:

$$\text{accuracy} = \frac{\text{Number of correctly predicted class}}{\text{Total number of predictions}}$$

When the dataset consists of severely imbalanced classes, using accuracy may be inappropriate, which can lead to favoring the class with large numbers. Fig. 1 shows the proportion of 100 landmark ids in the training dataset ordered from greatest to smallest. Although there is imbalance of class, it is not severe enough to cause the model to favor only the class with highest number of classes from training. Moreover, there is no reason to treat false positives and false negatives differently. Hence use of accuracy, which treats all misclassifications the same, may be justified for the given purpose of this project.

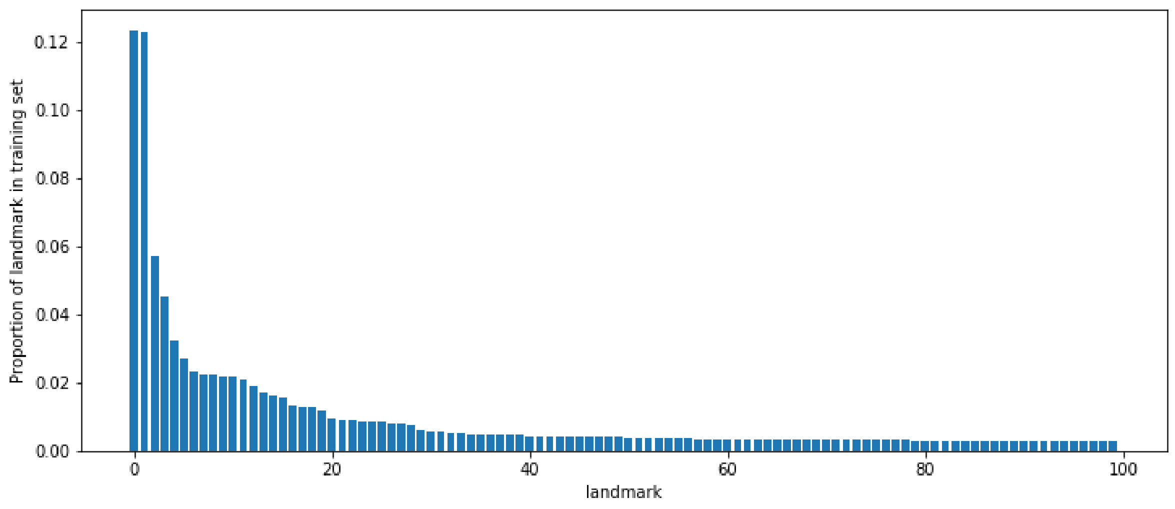


Fig. 1: Proportion of landmark ids in training dataset in descending order

II. Analysis

Data Exploration and Visualization

The original Landmark Recognition Challenge dataset provides two files, train.csv and test.csv. The training set images each depict exactly one landmark. Test images may depict no landmark, one landmark, or more than one landmark. Each image has a unique id (a hash) and each landmark has a unique id (an integer). Due to restrictions on distributing the actual image files, the dataset contains a url for each image.

The original number of images for training and test dataset are 1,225,029 and 117,703, respectively with a total of 14,951 unique landmark ids. Use of full dataset may require large storage capacity (in the order of hundred GB) and computational power not easily available to an individual. Therefore a subset of dataset has been used for this project as follows:

1. First, top 100 landmark ids most frequently appearing are identified among 14,951 landmark ids.
2. Then, 2% of images from each of the 100 landmark ids are downloaded.

The procedure is similar to stratified sampling applied to the top 100 frequent classes to preserve the ratio of images among the classes. The outlined procedure resulted in 8158 downloaded images with frequency distribution as shown in Fig. 2. It can be observed that some landmark ids have much higher frequency than others.

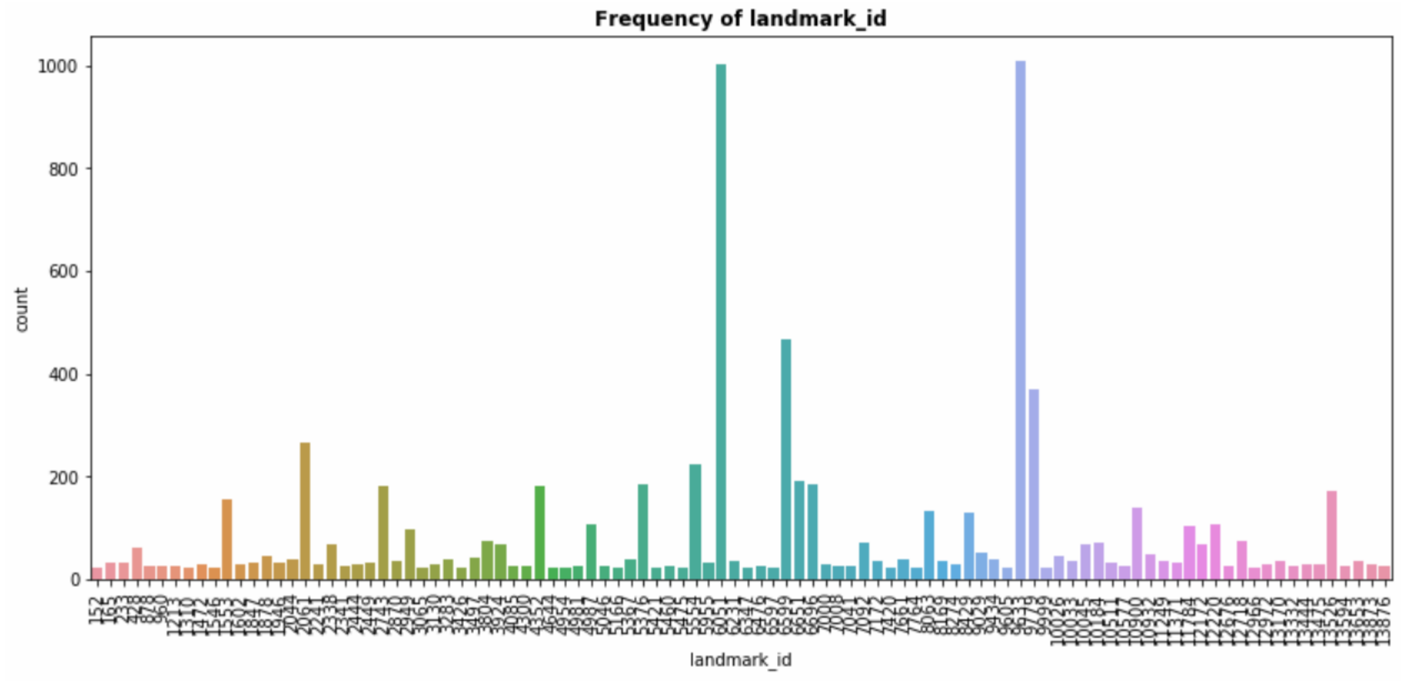


Fig. 2: Frequency distribution of 100 landmark ids in the dataset

An example of two images with the same landmark id is provided in Fig. 3.

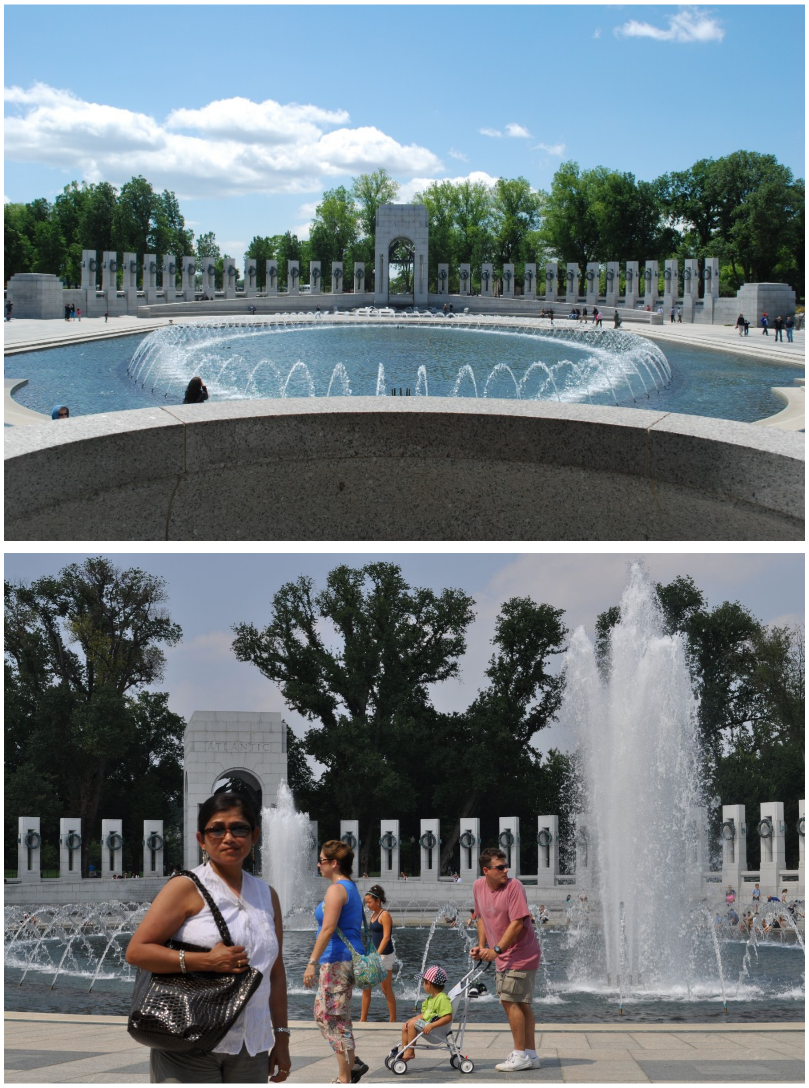


Fig. 3: Two images with the same class (landmark_id=3065)

Among the images downloaded, there were corrupted or blank images.

Number of training images: 6480 Number of validation images: 805 Number of test images: 809

The 8158 images have been split into training, validation, and test datasets with split ratio of 0.8:0.1:0.1 using stratified sampling method.

Among the 8158 images downloaded, 64 images were either corrupted or missing. These images were manually removed from the respective dataset, which resulted in training, validation, and test dataset sizes of 6480, 805 and 809, respectively with 100 landmark ids in each of the dataset.

Algorithms and Techniques

Base CNN

CNNs are a category of neural networks that have been successfully used in image recognition and classification. In this project, the CNN model is built using the framework of TensorFlow with Keras library. A schematic of the structure of a CNN is shown in Fig. 4 with an example input image and output predictions.

There are mainly three main neural layers: convolutional layer, pooling or subsampling layer, and fully connected layer. In the convolutional layer, multiple kernels are used to create feature maps for the input image. Each kernel produces a convolved image from the input image. A pooling layer typically follows a convolutional layer, which downsamples and reduces the dimensionality of the feature maps. This downsampling helps with overfitting issues and computational cost. The pooling layer also makes the network invariant to small transformations, distortions and translations in the input image and leads to almost scale invariant representation of the image [11]. The convolutional and pooling layers can be repeated to form a deeper network. As the convolutional network becomes deeper, the feature map becomes smaller, more abstract and loses spatial information. These reduced feature maps are then fed into the fully connected layers where the high-level reasoning is performed. The neurons in the fully connected layer have full connections to all activation in the previous layer. The fully connected layer eventually converts 2D feature map to 1D feature vector and classifies them. In summary, the convolutional and pooling layers act as feature extractors from the input image while the fully connected layers act as a classifier.

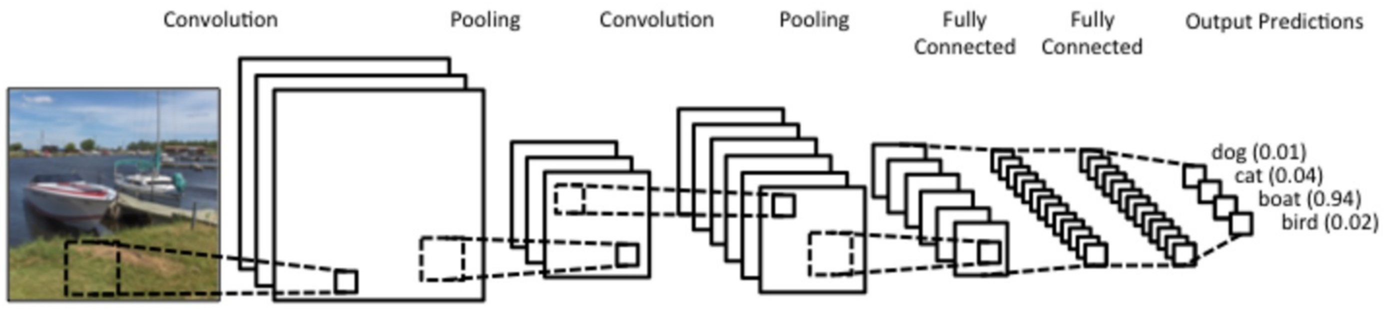


Fig. 4: A schematic of the CNN model [9]

Data augmentation

Data augmentation increases the training dataset by adding more images using existing images. New images can be created in various ways. For example, new images can be generated from the original image by random cropping, translation, rotation, reflection, distortion, scaling, etc. An example from [12] shows an original image of a cat and newly created images through data augmentation.

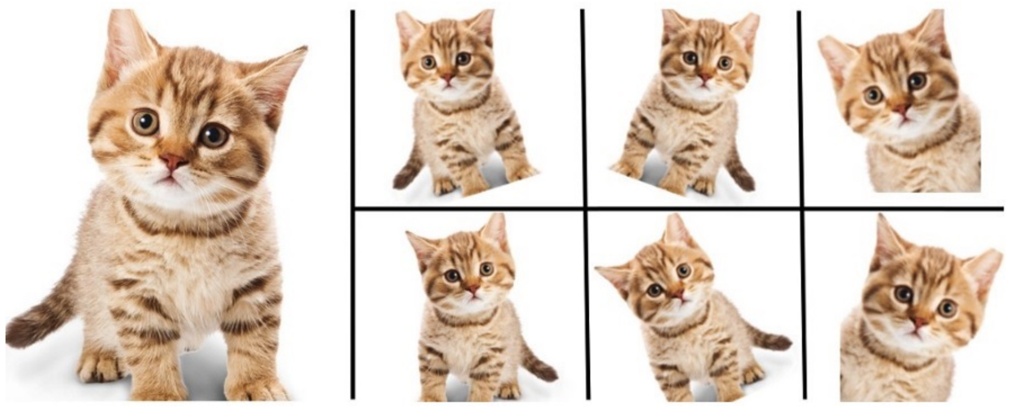


Fig. 5: An example of images generated from data augmentation

As noted in Fig. 2, some landmark ids have only small amount of images. Data augmentation may help increase the prediction accuracy for those landmark ids with smaller amount of training data.

Batch normalization

Batch normalization involves normalization of the features across the training examples in each mini-batch. The layer first normalizes the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation. Then, the layer shifts the input by a learnable offset and scales it by a learnable scale factor [13]. While there are competing explanations why batch normalization works, its effectiveness is apparent in various studies. Empirically it appears to stabilize the gradient (less exploding or vanishing values) and batch-normalized models appear to overfit less [14].

Benchmark

As shown in Fig. 2, the frequency of the 100 landmarks appearing in the dataset varies significantly. In other words, the expected accuracy of random guessing may vary depending on the selected test dataset. The accuracy of the benchmark model, given the test dataset, can be calculated as the following:

(i) Given the test dataset and 100 unique landmark ids, the probability of correctly classifying a landmark id, $P(id)$ can be calculated as,

$$P(id) = n_{id}/N$$

where n_{id} is the number of the landmark id in the test dataset, and N is the total number of images in the test dataset. For example, if landmark id = 99 appears 50 times in the test dataset with size of 800, $P(99) = 50/800$.

(ii) The expected value of correct number of landmark id classified is:

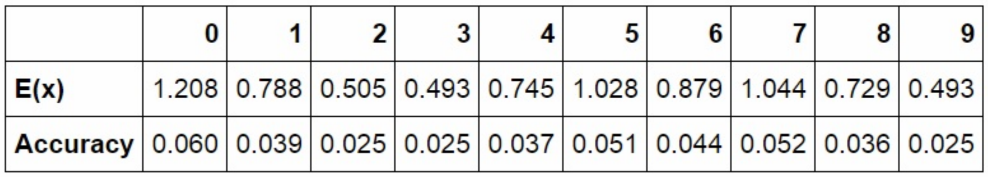
$$E(x) = \sum_{x=0}^{x=nid} (x \cdot P(id(x)))$$

(iii) Finally, the expected accuracy for correctly classifying the landmark id is:

$$E(accuracy) = \frac{E(x)}{N}$$

The expected accuracy of the benchmark model as given above can be calculated for a given test dataset. Unfortunately, huge number of combinatorial calculation in step (ii) caused memory overflow and too long of a computational time. Hence a Monte Carlo simulation has been employed to estimate the expected accuracy. Random sampling size of n_{id} landmark id was used, and simulations were run 10 times. Table 1 summarizes the expected correct number of E(x) and expected accuracy for 20 random selection from test dataset.

Table 1: Results of expected accuracy from Monte Carlo simulation



	0	1	2	3	4	5	6	7	8	9
E(x)	1.208	0.788	0.505	0.493	0.745	1.028	0.879	1.044	0.729	0.493
Accuracy	0.060	0.039	0.025	0.025	0.037	0.051	0.044	0.052	0.036	0.025

Therefore, the mean number of correctly predicting landmark id is 0.791 out of 20 randomly selected images. Thus the mean expected accuracy is 3.96%. This means that random guessing of the landmark id will result in accuracy of approximately 4.0%.

III. Methodology

Data Preprocessing

To improve the model accuracy and computational efficiency, regularization techniques such as dropout, batch normalization, and data augmentation has been considered when building the optimal CNN for the reduced dataset. The batch normalization technique [15], which transforms mean activation close to 0 and the activation standard deviation close to 1, has shown to improve accuracy with fewer training steps in image recognition problems. Many times, landmark images may be rotated, take up only a portion of the whole image, and may be out of center. The data augmentation technique may help improve accuracy for such images. Moreover, data augmentation strategy can help with those landmark ids with only small number of training images.

The 8158 images have been split into training, validation, and test datasets with split ratio of 0.8:0.1:0.1 using stratified sampling method. Upon inspection of all images, however, some images were either corrupted or missing. Hence those images were manually removed from the respective dataset. Finally, the resulting training, validation, and test dataset sizes were 6480, 805, and 809 with 100 landmark ids in each of the dataset.

The training, validation, and test images are moved to respective directories using their image ids as filenames. Each image is loaded and converted to a 4D tensor (number of images, height, width, channels) that can be used for training, validating and testing in the Keras CNN model. The procedure is as follows:

1. Load a RGB image into PIL image format with given target size of (192, 256)
2. Convert the PIL image to 3D tensor Numpy array of (192, 256, 3)
3. Convert the 3D tensor to a 4D tensor with shape (1, 192, 256, 3)
4. Stack all 4D tensors from all images into 4D tensor with shape (total number of images, 192, 256, 3) and rescale the pixel values by dividing them by 255

Implementation

Base CNN

The Base CNN followed the sequence shown in Fig. 4. The base CNN only uses the three main types of layers: convolutional, pooling, and fully connected (dense) layers. The first convolutional layer consisted of 16 filters, and the filter numbers doubled as more convolutional layers were subsequently added to form deeper network. The kernel window size varied between 2 and 4, mostly using size of 3. All convolutional layers employed ReLU activation function with 'same' padding, which means that input and output length of the convolutional layer is the same. After each convolutional layer, a max pooling layer with size of 2 was used to prevent overfitting.

With the input shape of (192 256), the combination of convolutional and max pooling layers will result in output shapes of (96, 128), (48, 64), (24, 32), (12, 18) and (6, 9) as the network deepens. It seems reasonable to speculate that representative features would be at levels 3, 4 and 5 considering the size of the feature that can capture the characteristics of the landmark. Therefore, the combination of convolutional and max pooling layers were repeated 3 to 5 times to form a deeper network as a feature extractor. The features are then fed into series of fully connected layers with node numbers selected among 1024, 512, and 256. After each fully connected layer, a dropout layer with value of 0.3 was used. Finally, a fully connected layer with 100 nodes, which is the same as the total number of landmark ids, is used with the Softmax activation to predict the probabilities of each class (landmark id). The series of fully connected layers act as a classifier.

Total of 11 runs with various settings have been performed as summarized in Table 2. As an example, the CNN model for Run 1 is provided as follows:

1. Conv2D(filters=16, kernel_size=3, padding='same', activation='relu')
2. MaxPooling2D(pool_size=2)
3. Conv2D(filters=32, kernel_size=3, padding='same', activation='relu')
4. MaxPooling2D(pool_size=2)
5. Conv2D(filters=64, kernel_size=3, padding='same', activation='relu')
6. MaxPooling2D(pool_size=2)
7. Flatten()
8. Dense(512, activation='relu')
9. Dropout(0.3)
10. Dense(100, activation='softmax')

Table 2: Summary of parameter settings used in Base CNN

Run	Accuracy	Filter	Kernel	Dense	Batch size
1	0.2361	(16,32,64)	(3,3,3)	(512,100)	64
2	0.3461	(16,32,64,128)	(3,3,3,3)	(512,100)	64
3	0.2917	(16,32,64,128)	(3,3,3,3)	(512,100)	128
4	0.3004	(16,32,64,128)	(3,3,3,3)	(512,100)	32
5	0.3337	(16,32,64,128)	(4,4,3,3)	(512,100)	64
6	0.3696	(16,32,64,128)	(4,3,3,2)	(512,100)	64
7	0.3523	(16,32,64,128)	(4,3,3,2)	(1024,512,100)	64
8	0.2930	(16,32,64,128)	(4,3,3,2)	(256,100)	64
9	0.3337	(16,32,64,128)	(4,3,3,2)	(512,256,100)	64
10	0.3498	(16,32,64,128,256)	(3,3,3,3,3)	(512,100)	64
11**	0.4227	(16,32,64,128,256)	(4,3,3,2,2)	(1024,512,100)	64

** Optimal setting

All other runs followed the same sequence with parameter settings given in Table 2. The test accuracy of the Base CNN ranged from 0.236 to 0.423, which was significantly higher than the benchmark model of random guessing (around 0.04). Looking at Runs 1, 2 and 10, it seems 4 to 5 convolutional-maxpooling layers are needed as Run 1 with 3 convolutional layers resulted in much lower test accuracy. Comparing Runs 2, 3 and 4, batch size of 64 gave the best test accuracy. Runs 2, 5 and 6 compare tweaking of kernel sizes, and decreasing the kernel for deeper network seems to give good results. Finally, Run 11 was selected as the best performing base CNN model with test accuracy of 0.423.

Data Augmentation

Data augmentation is performed using Keras ImageDataGenerator function considering following parameters: rotation_range, width_shift_range, height_shift_range, and zoom_range. Using the best performing base CNN model (Run 11 from Table 2), different parameter combinations have been tested as shown in Table 3. Due to extensive running time, requiring epochs over 50, only three settings have been tested. Run 1 resulted in the highest test accuracy of 0.666.

Table 3: Summary of parameter settings used in Data Augmentation

Run	Accuracy	Rot_angle	width_shift	height_shift	zoom
1**	0.6663	10	0.2	0.2	0.4
2	0.6267	45	0.2	0.2	0.3
3	0.5290	90	0.2	0.2	0.3

** Optimal setting

Batch Normalization

There are abundant discussions as to how batch normalization needs to be implemented in a convolutional layer. Some scientists suggest that batch normalization layer should be used before the activation layer as in the original paper by Loffe and Szegedy [15] and as noted in various web blogs [16] [17]. An example code block is provided below:

```
modelBN.add(Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False))
modelBN.add(BatchNormalization())
modelBN.add(Activation("relu"))
modelBN.add(MaxPooling2D(pool_size=2))
```

On the other hand, some recommend using the batch normalization layer after the activation layer [18] as:

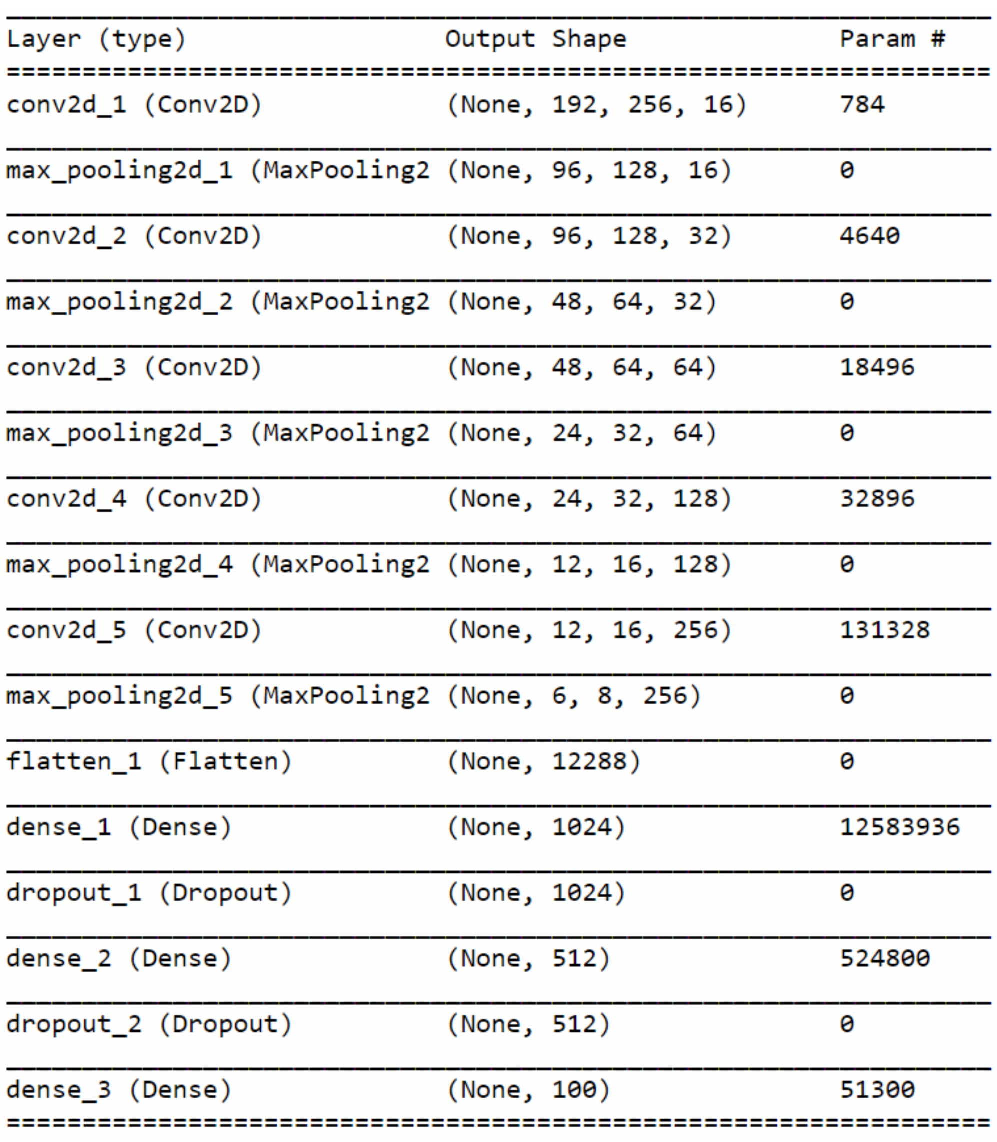
```
modelBN.add(Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False))
modelBN.add(Activation("relu"))
modelBN.add(BatchNormalization())
modelBN.add(MaxPooling2D(pool_size=2))
```

The batch normalization layer has been implemented both ways using the setting from Run 7 in Table 2. By trying different learning rates and different adaptive optimizers, the accuracy similar to Base CNN was achieved but required much longer training time with more epochs. For the dataset given, the batch normalization was not helpful in reducing time or improving accuracy, and therefore, was not included in the final model.

IV. Results

Model Evaluation and Validation

A schematic of the final model selected (Run 11 in Table 2) is shown in Fig. 6. The model had total of 13,348,180 parameters. The convolutional and max pooling layers had five repeating structures where the image size of (192,256) reduced to feature extraction of (6,8). The extracted features then underwent three fully dense layers for classification of the image.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 192, 256, 16)      784
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 96, 128, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 96, 128, 32)       4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 48, 64, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 48, 64, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 24, 32, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 32, 128)       32896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 16, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 16, 256)       131328
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 8, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12288)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              12583936
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               51300
=================================================================
```

Fig. 6: Final model used in prediction of Landmark

The Base CNN had test accuracy of 0.4227, and the accuracy and loss functions are plotted in Fig.7. It can be seen that overfitting occurs after epoch 6 where the validation accuracy starts to decrease and validation loss increases.

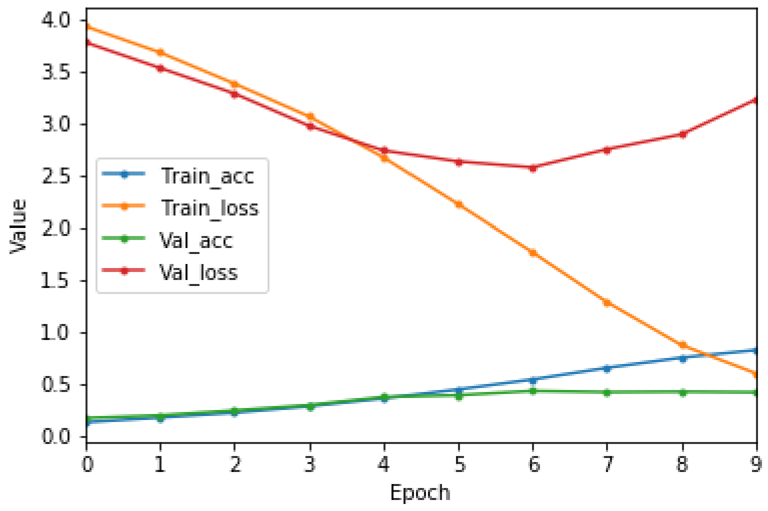


Fig. 7: Accuracy and loss function of training and validation dataset of the Base CNN model

The final model incorporated a data augmentation scheme with settings given in Run 1 of Table 3. The test accuracy increased from 0.4227 to 0.6663. The accuracy and loss function of the training and validation dataset is provided in Fig. 8. While the training loss function continually decrease, the validation loss function flattens after epoch 40. Comparing the results of Fig. 7 and 8, data augmentation successfully increased the input data and helped with overfitting shown in Base CNN model.

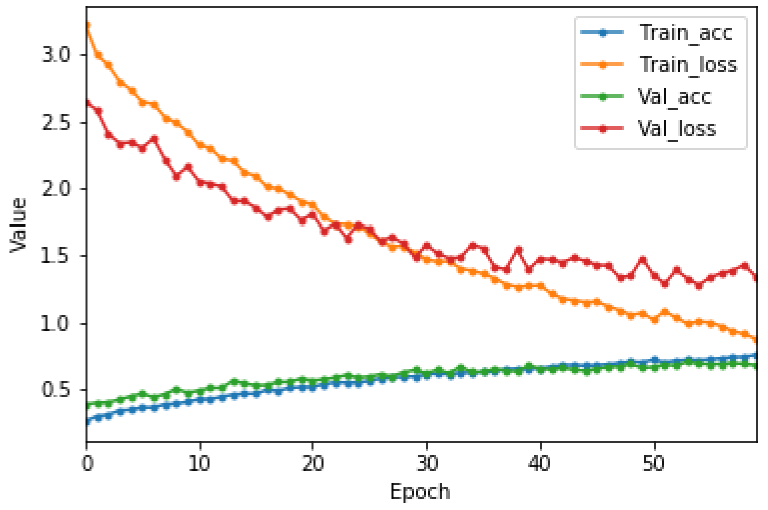


Fig. 8: Accuracy and loss function of training and validation dataset of the Base CNN model with data augmentation

The training, validation and test accuracies of the final model were very close around 0.65. This suggests that the model is neither underfitting or overfitting the data and has appropriate complexity. It seems creating a similar class imbalance in training, validation and test dataset through stratification may have helped to acquire appropriate model for the dataset. In regards to actual competition, it is questionable whether such approach would result in a good model. The distribution of classes in the test dataset is unknown and contains landmark ids not present in the training dataset. Perhaps, balancing the training dataset using weights may be considered while using the same evaluation metric from the competition.

Justification

The benchmark metric as described earlier was the probability of randomly guessing the correct landmark id, which was only about 4%. It is very obvious the CNN model with data augmentation outperformed the random guessing by a wide margin.

The Global Average Precision (GAP) at *k*=1, which was used as the evaluation metric in the Google Landmark Recognition Challenge, has been calculated as a reference. The GAP metric as defined in the competition is the following:

$$GAP = \frac{1}{M}\sum_{i=1}^{N} P(i)rel(i)$$

where:

- *N* is the total number of predictions returned by the system, across all queries
- *M* is the total number of queries with at least one landmark from the training set visible in it (note that some queries may not depict landmarks)
- *P(i)* is the precision at rank *i*
- *rel(i)* denotes the relevance of prediction *i*: it’s 1 if the *i*-th prediction is correct, and 0 otherwise

However, it must be noted that a direct comparison between the scores from the competition and the project may be inappropriate. The images used in this project is only about 2% of the training dataset. Moreover, the original test dataset consists of landmark ids not in the training dataset. Thus *M* is greater than *N* in the original competition. By design of this project, however, all landmark ids in the test dataset are included in the training dataset. Therefore, *M* equals *N* in this project, and the GAP score from this project is expected to be much higher than those of competition scores. Nonetheless, the scores from competition may be used as a reference score. The top three final leaderboard scores from the competition were 0.304, 0.290 and 0.289 as reported from the competition website [19]. The GAP score calculated from the final model with data augmentation is 89%, which is much higher than those from the competition.

V. Conclusion

Free-Form Visualization

A schematic of the final CNN model developed is provided in Fig. 9. The input image with shape of (192, 256) is supplied to the image extraction network, which consists of five sequential combination of convolutional and max pooling layers. The extracted features are supplied to three fully connected layers with dropout layer to identify 100 landmark ids. The accuracy of final CNN with data augmentation was 66.6%, which was significantly higher than random guession of CNN without data augmentation.

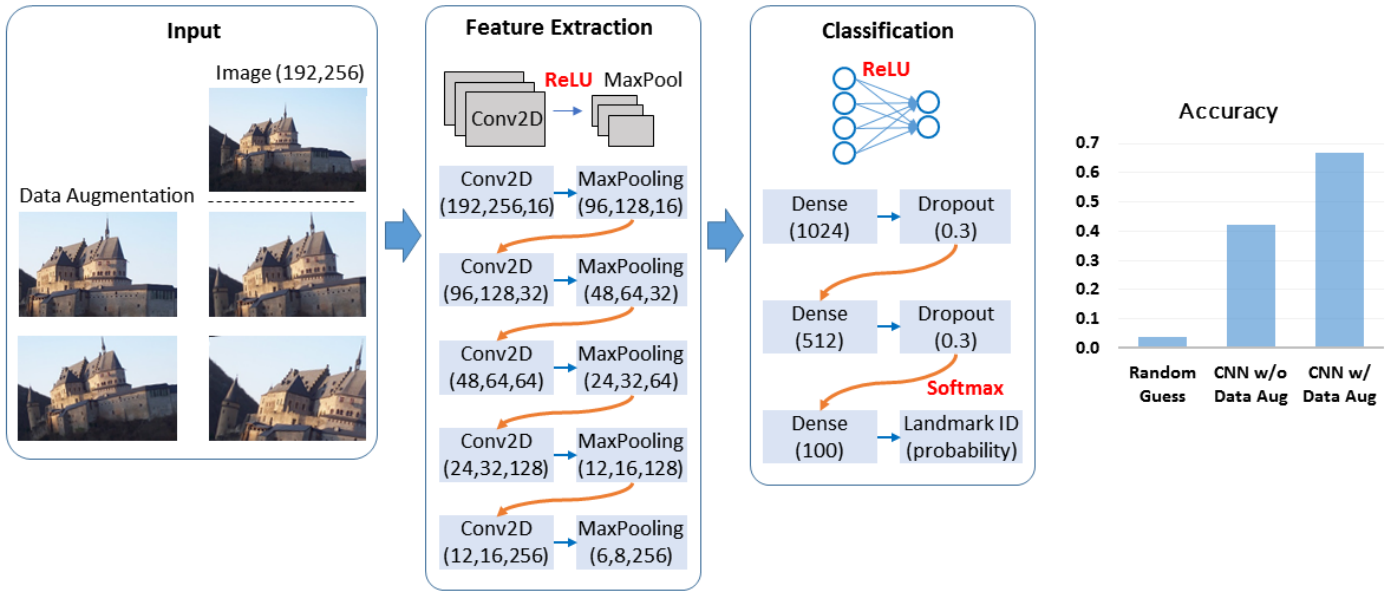


Fig. 9: Final CNN model and performance

Reflection

In this project, a deep learning model based on CNNs is proposed for Google Landmark Recognition Challenge from Kaggle. The original dataset from Google Landmark Recognition Challenge has been reduced for this project due to computational limitation. First, top 100 landmark ids most frequently appearing were identified among 14,951 landmark ids. Then, 2% of images from each of the 100 landmark ids were downloaded. The procedure is similar to stratified sampling applied to the top 100 frequent classes. Among the 8158 downloaded images, the dataset was split into training, validation, and test datasets with split ratio of 0.8:0.1:0.1 using stratified sampling.

The Base CNN consisted of three main types of layers: convolutional, pooling, and fully connected (dense) layers. The convolutional layer was followed by the max pooling layer, which acted as a feature extraction layer. The combination of convolutional-max pooling layer were repeated to form a deep network. As the network deepens the relevant features are identified while the spatial information is lost. The extracted features are supplied to series of fully connected layers, which function as a classifier. The last fully connected layer employed Softmax function to predict the probability of each landmark id (class).

Total of 11 different settings were tested. The best performing Base CNN had five series of convolutional-max pooling layer that identified feature of shape (6, 8) pixels from original image shape of (192, 256) pixels. The features then underwent three fully connected layers to predict the probability of each landmark id. The best performing Base CNN resulted in accuracy of 42.3%.

The data augmentation technique was employed as it can effectively increase the input data without overfitting issues. Since the same landmark image may shift, rotate or zoom differently in various images, the technique can particularly help with improving accuracies of those landmark ids with small number of training images. As a result, the accuracy significantly improved to 66.6%.

There were two significant challenges that were encountered during this project. First challenge was dealing with the Big Data. This was my first time dealing with such huge dataset. I did not realize how difficult it was to manipulate the original dataset from Landmark Recognition (~ hundreds of GBs). Coming up with a reasonable strategy to reduced the dataset without losing the characteristics of the original intent of the competition took much time.

The second challenge was preprocessing the image dataset so that it can be loaded into the Keras CNN model. This involved manual identification and removal of corrupted or missing images as well as transforming image formats. This required complete understanding of detailed steps in converting the image files, which was not an easy task for someone who had limited experience in CNN or data involving images.

Improvement

There may be several ways to improve the model. One method is to take advantage of transfer learning. Pre-trained models that can identify humans, trees, animals (like pets), and vehicles are available. The landmark images often contain these objects since many photos are taken by tourists, which include people and natural surroundings. Excluding these objects from the images may help with training and identifying the relevant features to extract.

Another method may be to use a different classifier in developing the model. Instead of using the fully connected layers as the classifier, other classifiers such as support vector machines, variants of decision trees (with boosting or random forest), and nearest neighbor may be used after the feature extraction layer.

VI. References

1. A. Voulodimos et al., "Deep Learning for Computer Vision: A Brief Review", Computational Intelligence and Neuroscience Volume 2018, Article ID 7068349.
2. Yoshua Bengio (2009), "Learning Deep Architectures for AI", Foundations and Trends® in Machine Learning: Vol. 2: No. 1, pp 1-127.
3. R. Girshick et al, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.
4. S. Lawrence et al, "Face recognition: a convolutional neural-network approach", IEEE Transactions on Neural Networks, Vol:8, Iss:1, Jan 1997.
5. T. Kautz et al., "Activity recognition in beach volleyball using a Deep Convolutional Neural Network", Data Mining and Knowledge Discovery, Vol:31, Iss:6, pp 1678-1705, 2017.
6. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", Advances in neural information processing systems, pages 1097-1105, 2012.
7. Matthew D Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks", arXiv:1311.2901 [cs.CV], 2013.
8. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", In Proc. Conf. Computer Vision and Pattern Recognition, pages 1-9, 2015.
9. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", In CVPR, 2016.
10. https://www.kaggle.com/c/landmark-recognition-challenge
11. https://github.com/rasbt/python-machine-learning-book/blob/master/faq/difference-deep-and-normal-learning.md

12. https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced
13. https://www.mathworks.com/help/nnet/ref/nnet.cnn.layer.batchnormalizationlayer.html
14. https://gluon.mxnet.io/chapter04_convolutional-neural-networks/cnn-batch-norm-scratch.html
15. Sergey Ioffe, Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv:1502.03167 [cs.LG].
16. https://gluon.mxnet.io/chapter04_convolutional-neural-networks/cnn-batch-norm-scratch.html
17. https://www.dlology.com/blog/one-simple-trick-to-train-keras-model-faster-with-batch-normalization
18. https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md
19. https://www.kaggle.com/c/landmark-recognition-challenge/leaderboard