2018 Open Data Analysis Competition Implementation Proposal

Team Name: Taiwan Deep Travel
Project Title: Deep Learning-Based Taiwan Travel Destination Recommendation App
Presenters: Cheng-Han Wu, Chi Li
Advisor: Dr. Yi-Ping Chang

I. Problems

Taiwan is blessed with beautiful natural and cultural landscapes, and there is a desire to promote Taiwanese attractions through a travel recommendation system that provides travel information tailored to the needs of domestic and international tourists. Additionally, modern life is busy, and people have limited time to search for travel information. If users can input images of their interested destinations into a mobile app, the app can recommend travel destinations that they might like, saving them the time of searching for information online. This app includes two ways of loading images into the model: A. Uploading an image of an interested destination from the device's photo library. B. Instantly capturing images of destinations through the camera function of the mobile app and loading them into the model. The system will utilize deep learning image recognition technology to recognize multiple scenes in real-time and recommend Taiwan travel destination images that are like the recognized scenes, along with detailed information about those destinations.

Contrasting with existing travel recommendation apps on the market, the advantages and features of an image-based travel destination recommendation system are as follows:

A. Shortcomings of current travel recommendation systems: Most systems rely on users searching for preferred destinations using keywords or tags. However, it can be challenging for users to accurately describe their ideal destinations with limited text and tags, resulting in less precise or overly broad recommendations from the system, which may lack value for users as a reference.

B. Precisely meeting user needs: Searching and recommending destinations using images, a "search by image" approach, allows for more precise fulfillment of user needs. Images can capture multiple features, eliminating the need for users to input each feature separately for the system to provide recommendations. Moreover, users often struggle to express their ideal scenes in words, making image-based recommendations more aligned with their intentions and better suited to meeting their specific requirements.

C. Focus on travel destination recommendations: An image-based travel destination recommendation system places emphasis on recommending travel destinations rather than other non-visual tourist experiences. However, for Taiwan, with its abundance of beautiful natural and cultural attractions, scenic appreciation itself is a significant selling point for travel. Therefore, such a recommendation system still holds market value.

D. Convenient user experience: With the widespread use of mobile devices, developing an app offers a more convenient user experience. Users can easily capture images of their surroundings using the camera function within the recommendation system, allowing the system to recommend Taiwan travel destination images and information that are similar to the scenes captured in the photos.

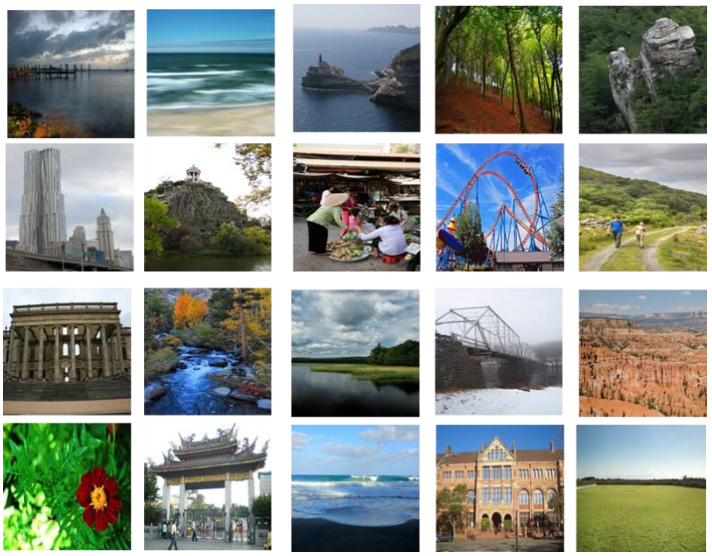
II. Source and Usage of Open Data

2.1 Model Dataset

The source of the dataset is "Places365-Standard[1]" from MIT CSAIL. The dataset includes both training and validation data. The training dataset covers 365 scenes, with 5000 image data per scene. Each image is in the format of 256x256x3 pixels. The validation dataset also covers 365 scenes, with each scene corresponding to the scenes in the training dataset. Each scene in the validation dataset consists of 100 images, with each image also in the format of 256x256x3 pixels. The usage involves subjectively selecting 20 scenes from

[1] MIT CSAIL Places365-Standard：http://places2.csail.mit.edu/download.html

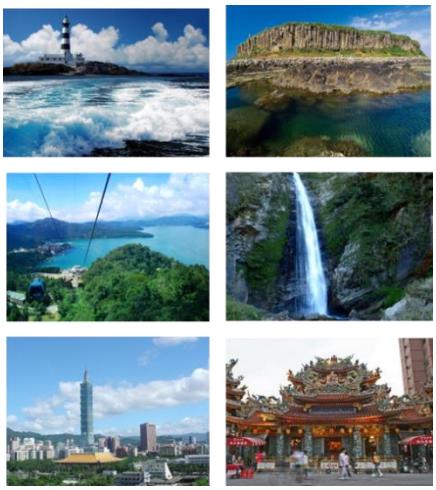
the 365 scenes in the Places365-Standard dataset that are likely to appear in travel destinations.



The 20 scenes were selected from the training dataset within the Places365-Standard dataset, with each scene containing 5000 image data. These 20 scenes, totaling 20x5000 image data, were used as the training data for the model. Similarly, the 20 scenes were also extracted from the validation dataset within the Places365 dataset, with each scene containing 100 image data. These 20 scenes, totaling 20x100 image data, were used as the validation data for the model. The Convolutional Neural Network (CNN) was trained and validated using the 20x5000 training data and 20x100 validation data. The reason for selecting only 20 scenes for model training is to ensure smooth operation of the model in the app. By reducing the number of scenes, the overall model parameters and size can be reduced to prevent the model from becoming too large. Additionally, training a model to recognize a larger number of scenes may lead to a decrease in recognition accuracy. Furthermore, considering time constraints, training a model to recognize all 365 scenes would take a longer time. Therefore, only 20 scenes were selected from the total of 365 scenes for training purposes.

2.2 Recommended Attractions Dataset

The source of the dataset is the "Attractions - Tourism Information Database" provided by the Tourism Bureau of the Ministry of Transportation and Communications on the Government Data Open Platform. The dataset includes information on 4809 attractions covering various locations throughout Taiwan. Each attraction is accompanied by one image and a text description. The purpose is to allow users to input a random image of an attraction into the app. The system will generate a 1x20 vector, where each value represents the probability of the model predicting each scene. Simultaneously, each image from the recommended attractions dataset is also inputted into the image recognition model, generating 4809 1x20 vectors. The vector generated from the user's input image and the vectors generated from the 4809 tourist attractions are compared using cosine similarity 4809 times. Finally, the top five attractions with the highest similarity are recommended to the user.



III. Method

3.1 Tools Used

Keras was used as the deep learning tool. Keras is a high-level deep learning framework written in Python and can run on top of TensorFlow. It features simplicity and rapid design and supports convolutional neural networks and GPU computation. Our team utilized the Keras deep learning framework for implementing and training the convolutional neural network. Additionally, we utilized the compute engine service provided by the Google Cloud Platform (GCP) to set up virtual machine instances. The virtual machines had two GPUs,

13GB of memory, and ran on the Windows Server 2016 operating system. The standard persistent disk had a capacity of 200GB. The GPU specification was NVIDIA Tesla K80 with 24GB of memory. We used the Remote Desktop Protocol (RDP) file provided by the Google Cloud Platform for remote control. Furthermore, we installed GPU drivers and Python packages on the virtual machine.

3.2 Data Processing

Shuffle all the image paths of the training dataset's 20 scenes to randomize their order. Then, resize the images of the training and validation datasets' 20 scenes from 256x256x3 pixels to 224x224x3 pixels to match the input size required by the ResNet50 model used in subsequent transfer learning. Additionally, normalize the pixel values of the training and validation datasets' images. This normalization process can improve the accuracy of the trained model's predictions and facilitate faster convergence during gradient computations. Finally, utilize Keras Image Data Augmentation to augment the validation dataset. This involves applying various transformations such as rotation, resizing, scaling, adjusting brightness and color temperature, and flipping to the original 20x100 image data. Afterwards, randomly split the augmented validation dataset into a validation dataset and a testing dataset. The validation dataset is used to determine the model's hyperparameters, while the testing dataset is used to evaluate the final model.

3.3 建模流程

The following modeling process involves training all layers for 5 epochs and using 2000 augmented validation images for validation.

3.3.1 Determining the appropriate transfer learning model (Transfer Learning)

Transfer learning in neural networks can be described as the process of transferring the weights of each node in the network from a pre-trained network to a completely new network. However, there is no need to retrain the weights of each layer in the network. Instead, only the fully connected layer of the model

architecture is modified. This approach is taken because building a model from scratch is complex and time-consuming, while transfer learning allows for faster learning efficiency.

The performance of six transfer learning models, namely VGG16, ResNet50, MobileNet, DenseNet, NasNetMobile, and MobileNetV2, was compared. VGGNet, developed collaboratively by researchers from the Visual Geometry Group at the University of Oxford and Google DeepMind in 2014, is a deep convolutional neural network. VGGNet exhibits strong scalability and generalization when transferred to other image datasets. It successfully constructed deep networks with 16 to 19 layers. ResNet, which received the Best Paper Award at CVPR (IEEE Conference on Computer Vision and Pattern Recognition) in 2016, and won the championship in the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2015 competition, addressed the issues of vanishing gradients and gradient explosion that occur with increasing network depth. Compared to VGGNet, ResNet successfully constructed deeper neural networks with 50, 101, and 152 layers. MobileNet, introduced by Google in 2017, is a lightweight neural network model designed to address the problem of running overly complex models on low-end devices such as smartphones. The Google team has demonstrated good results using MobileNet for various image recognition tasks. DenseNet, awarded the Best Paper Award at CVPR 2017, improved the backward propagation of gradients compared to ResNet, making the model easier to train. Comparing DenseNet with ResNet on the CIFAR-100 and ImageNet datasets, DenseNet showed lower prediction errors and parameter sizes than ResNet.

Table 1 shows the results of training each model using transfer learning, with all layers trained for 5 epochs. It can be observed that MobileNet performs better in terms of accuracy, size, and training time compared to the other models.

Table 1

Model	Loss	Top-1 Acc	Top-3 Acc	Val loss	Top-1 Val_acc	Top-3 Val_acc	Size	Parameters	Depth	Time
VGG16 (2014 ILSVRC)	0.8538	0.7390	0.9250	0.9885	0.6760	0.9160	266 MB	23,238,356	27	47 ms/step
ResNet50 (2015 ILSVRC)	0.3632	0.8760	0.9800	1.0926	0.7060	0.9140	417 MB	36,435,476	180	40 ms/step
MobileNet (2017 Google)	0.9317	0.7000	0.9170	1.0584	0.6940	0.9030	110 MB	9,654,100	92	17 ms/step
DenseNet121 (2017 CVPR)	0.7228	0.7650	0.9350	0.9701	0.7030	0.9150	154 MB	13,462,740	431	38 ms/step
NASNetMobile (2018)	0.7656	0.7650	0.9320	0.8591	0.7320	0.9250	128 MB	10,895,656	774	34 ms/step
MobileNetV2 (2018 Google)	0.8226	0.7480	0.9240	0.9828	0.6780	0.9120	118 MB	10,288,852	160	20 ms/step

3.3.2 Determining the optimal optimization algorithm

Comparing the Adam and Adafactor optimization algorithms for MobileNet, it is found that Adam performs better. Adafactor is an optimization algorithm developed by Google in 2018, based on the Tensor2Tensor deep learning library in Tensorflow. It addresses the issues present in popular optimization methods like SGD and Adam, specifically focusing on memory consumption, parameter updates, and learning rate. Table 2 compares the performance of using Adam and Adafactor as the optimization algorithms after training all layers for 5 epochs and validating with 2000 augmented validation data. It is evident that Adam outperforms Adafactor in terms of performance.

Table 2

Model	Loss	Top-1 Acc	Top-3 Acc	Val loss	Top-1 Val_acc	Top-3 Val_acc	Size	Parameters	Depth	Time
MobileNet with Adam (2015)	0.9624	0.6760	0.9030	1.5239	0.5565	0.8020	110 MB	9,654,100	92	22 ms/step
MobileNet with Adafactor (2018)	3.2579	0.0510	0.1300	3.4619	0.0500	0.1500	37 MB	9,654,100	92	21 ms/step

3.3.3 Deciding on Batch Normalization (BN) and activation functions

Proposed by Google in a 2015 paper, Batch Normalization (BN) is a technique that aims to stabilize the distribution of data passed from each layer to the activation function in a neural network. It helps prevent the issue of vanishing

gradients by maintaining a more stable distribution of inputs to the activation function in each layer. Table 3 compares three combinations: ReLU with BN, ReLU without BN, and SeLU with BN. The combination of ReLU without BN performs the best.

Table 3

Model	Loss	Top-1 Acc	Top-3 Acc	Val loss	Top-1 Val_acc	Top-3 Val_acc	Size	Parameters	Depth	Time
ReLU with BN	0.8410	0.7250	0.9230	1.3066	**0.5835**	0.8315	117.7 MB	9,855,316	95	21 ms/step
ReLU without BN	0.9624	0.6760	0.9030	1.5239	**0.5565**	0.8020	110 MB	9,654,100	92	22 ms/step
SeLU with BN (2015)	0.7213	0.7840	0.9390	1.3839	**0.5695**	0.8215	117.7 MB	9,855,316	94	20 ms/step

3.3.4 Deciding on Dynamic or Static Learning Rate

According to Table 4, it can be observed that a static learning rate yields the best results, particularly when the learning rate is set to 0.0002.

表四

Model	Loss	Top-1 Acc	Top-3 Acc	Val loss	Top-1 Val_acc	Top-3 Val_acc	Size	Parameters	Depth	Time
MobileNet (LR = 0.00002)	0.8410	0.7250	0.9230	1.3066	0.5835	0.8315	117.7 MB	9,855,316	95	21 ms/step
MobileNet (LR = 0.0002)	0.7651	0.7460	0.9360	1.3139	**0.5800**	0.8400	112 MB	9,855,316	95	21 ms/step
MobileNet (LR = 0.002)	1.1672	0.6360	0.8590	1.7032	0.4620	0.7455	112 MB	9,855,316	95	22 ms/step
MobileNet (LR = 0.002 _ 0.00002)	1.0588	0.6670	0.8770	1.4206	0.5540	0.7965	112 MB	9,855,316	95	21 ms/step

3.3.5 Hyper-parameter Tuning

L1 regularization and L2 regularization can be seen as penalty terms in the loss function, aiming to impose constraints on certain parameters in the loss function to reduce model complexity and address the issue of overfitting.

Table 5

kernel_regularizer activity_regularizer	Dropout	Dense	Loss	Top-1 Acc	Val loss	Top-1 Val_acc	Size	Parameters	Depth	Time
0	0.5	128	0.9438	0.6880	1.2661	0.5840	112 MB	9,855,316	94	21 ms/step
0	0.5	256	0.8881	0.6970	1.2515	0.5910	185 MB	16,281,044	94	22 ms/step
0	0.2	128	0.8632	0.7090	1.2756	0.5700	112 MB	9,855,316	94	21 ms/step
0	0.2	256	0.9959	0.6630	1.2755	0.5780	185 MB	16,281,044	94	21 ms/step
0.01	0.5	128	4.4488	0.0500	4.4374	0.0510	117.7 MB	9,855,316	94	22 ms/step
0.01	0.5	256	5.6078	0.1880	5.3630	0.2795	194.8 MB	16,281,044	94	21 ms/step
0.01	0.2	128	3.6754	0.3710	3.6977	0.3865	117.7 MB	9,855,316	94	22 ms/step
0.01	0.2	256	5.4007	0.4580	5.4992	0.4165	194.8 MB	16,281,044	94	22 ms/step

3.4 Model Performance Evaluation

The best models and parameters selected from each stage of the modeling process in Section 3.3 were applied to train and validate the data. Finally, the trained models were tested using the test dataset. A comparison was made between training for 5 epochs and training for 10 epochs.

3.4.1 Training and Testing Results for the 5-epoch Model

Under the 5-epoch training, the top-1 accuracy on the training data is 0.69, and the top-1 accuracy on the validation data is 0.5835 (Table 6). The top-1 accuracy on the testing data is 0.614 (Table 7).

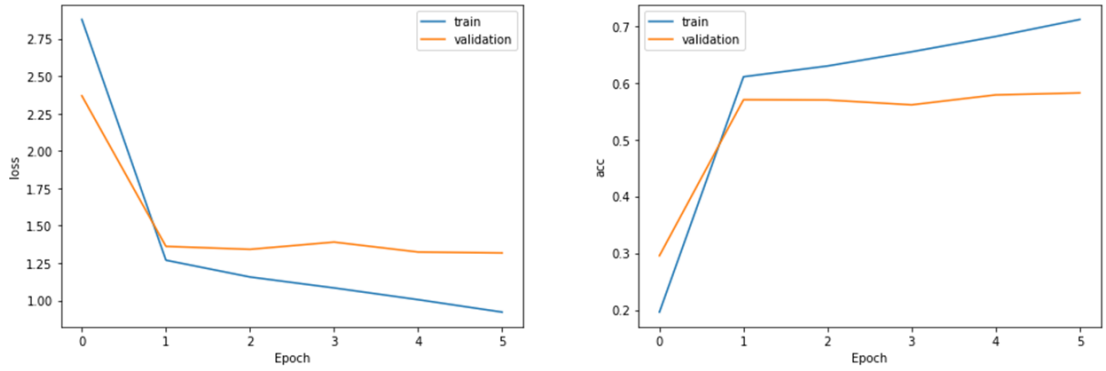


Figure 1

Table 6

Model	Loss	Top-1 Acc	Top-3 Acc	Val_loss	Top-1 Val_acc	Top-3 Val_acc
5週期	0.9600	0.6900	0.9020	1.2687	0.5835	0.8515

Table 7

Model	Loss	Top-1 Acc	Top-3 Acc
5週期	1.2499	**0.5970**	0.8510

3.4.2 Training and Testing Results for the 10-epoch Model

Under the 10-epoch training, the top-1 accuracy on the training data is 0.738, and the top-1 accuracy on the validation data is 0.607 (Table 8). The top-1 accuracy on the testing data is 0.614 (Table 7).

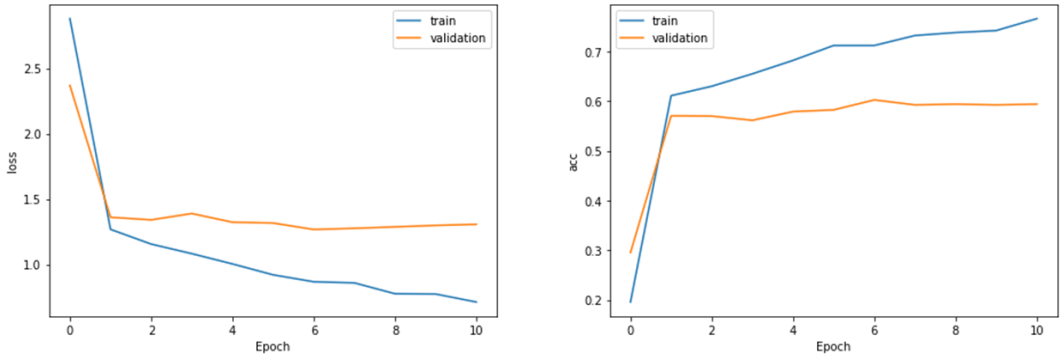


Figure 2

Table 8

Model	Loss	Top-1 Acc	Top-3 Acc	Val_loss	Top-1 Val_acc	Top-3 Val_acc
5_5週期	0.8040	0.7380	0.9360	1.2467	**0.6075**	0.8535

Table 9

Model	Loss	Top-1 Acc	Top-3 Acc
5_5週期	1.2206	0.6140	0.8465

3.5 Recommendation System

Inputting 4,809 Taiwan tourist attraction images into the model, each image will generate a probability vector of size 1x20 corresponding to 20 different scenes, which will be stored individually. When a user inputs a photo from

their mobile gallery or takes a picture on the spot into the image recognition model, the model will also generate a probability vector of size 1x20 representing the predicted probabilities for the 20 typical tourist scenes. The user's input image's probability vector (size 1x20) will be compared with the 4,809 probability vectors (size 1x20) of the Taiwan tourist attraction images using cosine similarity. The system will recommend the top five Taiwan attraction images and information based on the highest cosine similarity values.

Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them. When the two vectors are the same, the cosine similarity value is 1. When the two vectors are unrelated, the cosine similarity value is 0. When the two vectors are completely negatively correlated, the cosine similarity value is -1. Cosine similarity primarily distinguishes differences based on direction. However, it cannot measure the differences in the values of each dimension and is insensitive to absolute values, leading to errors in the results. Therefore, a modified cosine similarity (Figure 3) is used to address this issue. The modified cosine similarity subtracts the mean of each dimension's elements in the vector from the corresponding dimension's elements during calculation.

$$\frac{\sum_{i=1}^{20}[(A_i - \boldsymbol{\mu_A}) \times (B_i - \boldsymbol{\mu_B})]}{\sqrt{\sum_{i=1}^{20}(A_i - \boldsymbol{\mu_A})^2}\sqrt{\sum_{i=1}^{20}(B_i - \boldsymbol{\mu_B})^2}}$$

Figure 3

3.6 User Interface

All Python code in this system is manually converted into Swift programming language code, and the system is developed into an iOS mobile app using Apple's Xcode integrated development environment. The Keras model's .h5 file is converted into an iOS-compatible .mlmodel file using Apple's Core ML framework, and the model is loaded into the Xcode development environment. The Core ML 2.0 framework was introduced by Apple at the WWDC conference in 2018 as a machine learning development framework that allows for the seamless integration of pre-trained machine learning models into an app.

IV. Demo

4.1 Load an image from the phone's gallery

Video Link

4.2 Capture a photo using the phone

Video Link

V. Outlook

To enhance the model's ability to recognize scenes and improve the accuracy of recommended attractions, it is desired to train the model with more diverse scenes. Additionally, to improve the model's accuracy, increasing the training samples is expected to enhance the accuracy of validation and test data. It should be noted that only the validation and test data contain augmented image data, while the training dataset does not include augmented images. However, there may be some disparities between augmented and original image data, and if the model is not trained on these data, the accuracy may decrease when validating and testing the augmented data using the existing model architecture. In terms of adjusting hyperparameters, more combinations can be explored for comparison. For recommending attractions, incorporating more open data is desired to enrich the system's recommendations. TensorBoard can also be utilized for visualizing the neural network. Finally, regarding the app, expanding the system to the Android mobile operating system, collecting user evaluations and suggestions for the app would be beneficial.

VI. Additional Information

In this study, we have overcome challenges related to slow CPU computation speed and the conversion of Keras models to mobile device models.

Regarding the slow CPU computation speed, training convolutional neural network models typically requires a significant amount of data, and through testing, we observed a performance difference of approximately 30 times between CPU and GPU computations. To address this issue, our team invested considerable time in successfully researching and implementing the

use of virtual machines with GPUs on the Google Cloud Platform, which significantly improved the computation speed.

Regarding the conversion of Keras models to mobile device models, we encountered an issue when using Apple's Core ML framework to convert Keras models from the .h5 format to the. ML model format compatible with iOS operating systems. Specifically, there was an error related to importing the ReLU6 parameter. After discussing the problem with experts on the Stack Overflow website, we discovered that Keras versions 2.2.2 and later do not include the ReLU6 parameter in their packages. However, the coremltools package continued to have instructions for importing the ReLU6 parameter. We resolved this issue by modifying the code within the coremltools package.

VII. References

- 洪文麟。2016。深度學習應用於以影像辨識為基礎的個人化推薦系統-以服飾樣式為例。碩士論文。台南: 成功大學工程科學研究所
- Audrey Tam. 2018. Beginning Machine Learning with Keras & Core ML. http://www.raywenderlich.com/188-beginning-machine-learning-with-keras-core-ml
- Justin.h. 2017.图片数据集太少？看我七十二变，Keras Image Data Augmentation 各参数详解. github.com/JustinhoCHN/keras-image-data-augmentation
- Keras Documentation. Available at keras.io/applications/
- Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 2015. Deep Residual Learning for Image Recognition
- MIT CSAIL Com. 2018. Release of Places365-CNNs. github.com/CSAILVision/places365n
- Shao-Hua Sun. 2017. SELUs (scaled exponential linear units) - Visualized and Histogramed Comparisons among ReLU and Leaky ReLU. github.com/shaohua0116/Activation-Visualization-Histogram

- Stackoverflow. 2018. Cosine Similarity between 2 Number Lists. stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists