PNIE PROCEEDINGS OF THE NATIONAL INSTITUTE OF ECOLOGY OF THE REPUBLIC OF KOREA

Review Article

PNIE 2022;3(2):67-72
https://doi.org/10.22920/PNIE.2022.3.2.67
pISSN 2765-2203, eISSN 2765-2211

Bioimage Analyses Using Artificial Intelligence and Future Ecological Research and Education Prospects: A Case Study of the Cichlid Fishes from Lake Malawi Using Deep Learning

Deokjin Joo[1]*, Jungmin You[2], Yong-Jin Won[3]*

[1]Hashed, Seoul, Korea
[2]Research Institute of Ecoscience, Ewha Womans University, Seoul, Korea
[3]Division of EcoScience, Ewha Womans University, Seoul, Korea

ABSTRACT

Ecological research relies on the interpretation of large amounts of visual data obtained from extensive wildlife surveys, but such large-scale image interpretation is costly and time-consuming. Using an artificial intelligence (AI) machine learning model, especially convolution neural networks (CNN), it is possible to streamline these manual tasks on image information and to protect wildlife and record and predict behavior. Ecological research using deep-learning-based object recognition technology includes various research purposes such as identifying, detecting, and identifying species of wild animals, and identification of the location of poachers in real-time. These advances in the application of AI technology can enable efficient management of endangered wildlife, animal detection in various environments, and real-time analysis of image information collected by unmanned aerial vehicles. Furthermore, the need for school education and social use on biodiversity and environmental issues using AI is raised. School education and citizen science related to ecological activities using AI technology can enhance environmental awareness, and strengthen more knowledge and problem-solving skills in science and research processes. Under these prospects, in this paper, we compare the results of our early 2013 study, which automatically identified African cichlid fish species using photographic data of them, with the results of reanalysis by CNN deep learning method. By using PyTorch and PyTorch Lightning frameworks, we achieve an accuracy of 82.54% and an F1-score of 0.77 with minimal programming and data preprocessing effort. This is a significant improvement over the previous our machine learning methods, which required heavy feature engineering costs and had 78% accuracy.

Keywords: Animal detection, Artificial intelligence, Citizenscience, Deep learning, Education, Machine learning

Received October 20, 2021; **Revised** December 13, 2021; **Accepted** December 13, 2021

***Co-Corresponding author:**
Deokjin Joo, **e-mail** djin.joo@gmail.com, https://orcid.org/0000-0002-3463-0436
Yong-Jin Won, **e-mail** won@ewha.ac.kr, https://orcid.org/0000-0003-1343-8418

Introduction

Ecological research has recently depended on a tremendous increase in the usage of visual data. Motion-sensor cameras called "camera traps" produce hundreds of millions of images worldwide (Swanson *et al*., 2015). Uncrewed aerial vehicles (UAVs) and drones cover large areas with sub-decimeter resolution produce visual data in terabytes for a single project Kellenberger *et al*. (2020). While the camera traps can take millions of images, extracting information from these is traditionally done by humans (i.e., experts or community of volunteers), making the task time-consuming and costly that much of the valuable knowledge in these data repertoires remains untapped (Norouzzadeh *et al*., 2018). Moreover, humans suffer from the consistency problem and the trade-off issue between accuracy and computational efficiency. Deep learning is a revolutionary paradigm not only in the machine-learning field but also in ecological research. Recently, as deep-learning-based methods—particularly convolutional neural networks (CNNs)—have outperformed conventional methods in object detection, they are increasingly used in ecological research. In the future, attempts to utilize deep learning-based object recognition technology are expected to become more and more widespread in ecological research.

Ecological Research Using Artificial Intelligence (AI)

This section briefly summarizes several recent ecological research using image data processing technologies that leverage AI, such as CNNs. First, researchers can monitor various wildlife species and their numbers using a deep learning algorithm on the vast amount of data collected through camera traps. As digital camera technology advances in ecological research that requires the interpretation of large amounts of visual data obtained from extensive wildlife surveys, the number of studies using camera traps has increased to at least 125 in 2011 (Fegraus *et al*., 2011). However, when a motion sensor "camera traps" is used to collect wildlife pictures, it is time-consuming and expensive to extract images from video data manually. Norouzzadeh *et al*. (2018) collected wildlife pictures from the camera traps and then trained the images extracted from the image data through a deep neural network (DNN). The Snapshot Serengeti project is a large-scale camera trap project that has been shooting wildlife with 225 motion-sensing camera traps in Serengeti National Park, Tanzania, since 2011. It contains 1.2 million photos of 48 species of wildlife. The trained model classified 48 species, counted wildlife, and described animal behavior and companionship with young individuals with over 93.8% accuracy on the 3.2 million Snapshot Serengeti Dataset. If only species classification is done, species identification can be automated with an accuracy of 99.3%, and a crowdsourcing team made up of volunteers showed an accuracy of 96.6% in species classification of wildlife data.

Additionally, it would take 8.4 years for 30,000 volunteers to process 3.2 million images over a 40-hour week, but an automated model saves 99.3% of the time. This improvement in efficiency underscores the importance of using AI to automate data extraction and demonstrates the potential to transform ecology, wildlife biology, zoology, conservation biology, and wildlife behavior through data science. However, this study did not deal with images containing more than one species, and the performance was low for non-classification results such as the number of wildlife in the images. In addition, learning was inevitably performed on a background including wildlife, and when applied to object image data in a new place, the accuracy was lowered. For example, an image data model trained in the United States was less accurate in identifying the same species in Canada (Tabak *et al*., 2019). Norouzzadeh *et al*. (2021) conducted research using an *object detection* model to solve the limitations of existing researches. Object detection algorithms present a bounding box on each object and the model's certainty of the object class. The algorithm can handle multiple species in one image and can be efficiently applied to new places by learning and focusing only on wildlife. Norouzzadeh *et al*. (2021) analyzed eMammal Machine Learning data, in which researchers and citizen scientists classified 450,000 camera trap images from 270 species worldwide, and Caltech Camera Traps data, in which 245,000 camera trap images from 22 species were classified in the southwestern United States. It was used as pre-training data for image detection. The trained model was tested on the Snapshot Serengeti and North America Camera Trap Images (NACTI) data of 3.7 million camera trap images from 28 species in five regions of the northern United States. This model showed 91.92% accuracy in wildlife detection and 97.7% accuracy in species classification, and 93.2% accuracy in species classification in NACTI data.

Meanwhile, AI technology is being applied to individual identification studies of wildlife. Recently in Korea, a study of automatic individual identification of Indo-Pacific bottlenose dolphins (*Tursiops aduncus*) using AI is being conducted (personal communication). In dolphin research, various external features of dolphins such as body, pectoral fin, caudal fin, dorsal fin, head shape, and wounds are used to identify individual dolphins. Since the dorsal fin is constantly exposed when a dolphin rises above the water to breathe, the shape of the wounded dorsal fin is generally used for identification. Identifying dolphins from dorsal fin data is an easy but time-consuming task. In the past, researchers directly cropped and

compared the dorsal fins, but now, using deep learning, the time taken in the recognition-identification process is significantly reduced, thereby increasing the efficiency of analysis and accuracy.

Second, AI technology is being introduced to wildlife conservation. With the number of elephants and rhinos rapidly declining in Africa, various strategies exist to combat poaching using UAVs or drones. In particular, UAVs equipped with thermal infrared cameras can be used for night surveillance, combating night-time poaching activities (Bondi *et al*., 2018). However, it is complicated for park managers to monitor these UAV video images continuously. In addition, the more the drones are added, the burden of additional video monitoring increases. Bondi *et al*. (2018) developed an integrated cloud-based framework SPOT (Systematic POacher deTector) that detects the location of poachers in real-time using Faster R-CNN (Ren *et al*., 2015) using thermal infrared camera images collected by UAVs or drones. The program trains the AI offline using the image frames as the training data set with labeled images. The trained model automatically detects poachers online and displays the wildlife in a live video stream. Several experiments were performed to ensure that the online detection could be completed in the cloud and a balance between local computer and remote could be maintained. The developed SPOT system will be deployed in several national parks in Africa, including Botswana.

Finally, ecological surveys collect terabytes of images in a single survey, and such large-scale surveys require a large amount of photographic interpretation. Conventional models require programming and labeled image data sets, but most data collected from ecological studies are not annotated, making CNN training difficult. Kellenberger *et al*. (2020) developed Annotation Interface for Data-driven Ecology (AIDE) (accelerating image-based ecological surveys with interactive machine learning) to solve this problem, an open-source web framework designed for image annotation for ecological surveys based on deep learning. AIDE is a web-based, open-source collaboration platform that integrates machine learning models and multi-purpose labeling tools for image annotation without writing any code. AIDE utilizes active learning (AL) to do this through a feedback loop. AL in AIDE enables the machine learning model to be iteratively trained on the latest annotations provided by the user, and when training is complete, the model is used to obtain predictions for yet unlabeled images. AL allows machine learning models to be trained with large-scale image annotation processing and potentially small training data. In addition, AIDE provides an easy-to-use, customization labeling interface and supports multiple users, database storage, and capability to the cloud or multiple systems. Through this web-based open source collaboration platform, citizens can easily participate in ecological research, provide more observers and ideas, increase ecological literacy, save research costs and time, and learn about science and scientific research processes. As a result, science literacy will be strengthened, and social activities for ecosystem conservation will begin.

Ecological Education Prospects

Just as AI has begun to be used as a powerful tool for ecological research in the 21st century, it is expected that AI will become increasingly relevant to ecological education. Today, the global ecosystem is facing an ecological crisis. Air and water pollution, biodiversity reduction, resource depletion, climate change, etc., are important issues that humankind, including ecosystems, faces. Ecological education is the surest and essential solution to overcome the environmental and ecological crises and solve the conservation of the ecosystem and the existence of humankind (Capra, 1996). Ecological education is an ecosystem-based education and started when ecology began as a discipline of science and contains the principles of maintaining the Earth's ecosystem. It involves shifting the learner's cognitive base to an ecological paradigm. The scope of eco-education extends education related to ecology in the humanities or social sciences originating from ecology and ecology education. In addition, eco-education aims to promote ecological literacy, the ability to understand the natural systems that enable living things on the planet, as the basic knowledge to solve the crisis of the ecosystem (Kim, 2015).

Since the 2009 revision of the education curriculum in Korea, climate change and biodiversity have been accepted as essential topics, and related education has been expanded, increasing the need for education. Biodiversity conservation is a core value of sustainable development, leading human life to an ecological paradigm. Therefore, it is required to recognize the seriousness of the ecological crisis caused by the loss of biodiversity that humanity is facing and make efforts to conserve biodiversity (Lim & Lee, 2018). As a solution for the conservation of biodiversity, the voluntary actionability of members of society is required. Therefore, it is essential to provide opportunities for students to conclude on their own through participation in self-directed exploration and exploration of information through education (Noh, 2003).

In 2019, the Korean government announced the 'Artificial Intelligence National Strategy' (Ministry of Science and ICT, 2019). Furthermore, the 'Artificial Intelligence Talent Nurturing' strategy includes an education plan to strengthen AI education competency. In addition, 'Artificial Intelligence Education' will be introduced through the 2022 revised curriculum, which will be effective from 2025 (Ministry of Education, 2020). AI education consists

of 'programming' education that develops creativity by recognizing and solving problems and demonstrating it on a computer, 'AI principles' and 'AI utilization' education as basic communication for the future, and 'AI ethics' as critical thinking skills. However, despite the policy support related to AI education, the relevant curriculum structure and environment creation are still insufficient (Lee, 2020).

Ecological education integrated with AI can allow students to creatively solve the social problem of ecosystem destruction based on the principles of AI through educational programs that can develop AI literacy. Data recognition technology, which has entered the most stable stage among various AI models, can be used for future ecological education. For example, image classification/recognition technology can identify living things that appear in photos or videos and determine the species. Through an educational program that learns and classifies images of endangered wildlife and ecosystem-disrupting species through machine learning, it is possible to know the principles of machine learning and creatively use them in activities to conserve the ecosystem. In addition, by introducing ecological research using AI, it is possible to guide students to various careers and occupations related to ecology. If the ecological education program integrated with AI is applied, students' ecological literacy will increase, contributing to ecosystem conservation and environmentally sustainable society. In addition, as explained above with the case of AIDE, citizen science, which connects scientists and citizens by building an ecological research platform using AI, is expected as a powerful method for the education of AI and biodiversity and ecological research. Through this, we can accumulate more observation data and reduce the cost of scientific research.

Identification of Cichlid Fishes from Lake Malawi Using Deep Learning

The information that can be extracted from the bio-image data of wild animals and the purpose of use are very diverse. The representative case is species identification. Ecological research works which use AI to obtain useful information from bio-images are now moving into the application stage. At this point, we would like to look back on our 2013 past study. At that time, we developed a pipeline that extracts features from photographic data of African cichlid fishes through a computer vision method which automatically classifies them through a Support Vector Machine and Random Forests. The program we developed was able to classify photographic images of 594 cichlids belonging to 12 different classes (species and sex) with an average accuracy of 78%.

In this paper, we intended to compare the results of the early 2013 study to that of more modern deep learning methods. We reanalyze the 594 samples with the CNN deep learning method. For this purpose, we had to develop new programs and tutorials that incorporate newly developed deep learning methods since 2013. We hope that our experience and results will be helpful to researchers who want to analyze large amounts of ecological image data using AI.

For reproducibility, we made both the code and the 594 samples publicly available on the Github repository (https://github.com/forcecore/ghoti-2021). The full raw data will be also made available on Zenodo in the near future, after a thorough review. The URL will be published on the Github repository.

Trial and Error with CameraTraps Project

Microsoft is hosting AI for Earth (Microsoft AI, 2021) project, and CameraTraps (Microsoft, 2021) is one of the subprojects. The repository contains many valuable tools for detecting images collected from motion-triggered camera traps. In this section, we record an attempt to adapt an open-source project to our needs. Such a task may or may not succeed, as most open-source AI projects are unstable and usually become un-runnable in a couple of years. Eventually, we failed to meet our needs. However, our failure may provide a useful debugging process that can be valuable to the researchers. We briefly outline what we did here.

Following the CameraTraps Tutorial

CameraTraps so far seems to depend on Tensorflow, which is one of the most popular deep learning frameworks and graphics processing unit (GPU) may be used for faster training of the neural networks.

The first step is to set up a new "virtual environment" for running Python programs. We do this as each Python project may require different package (library) requirements, which can induce "dependency hell" when not careful. Programmers bypass this problem by isolating projects with virtual environments, one environment for each project. After the creation of the virtual environment, the required packages may be installed. The requirements are usually stated in requirements.txt by the project maintainers.

To feed our dataset, we need to figure out the data processing pipeline the project has. We need both the project documentation on the data format and the example dataset. It is common for deep learning research projects to fail over time for two reasons. One is, the researchers move onto the next phases of their research, and the previous programs get out of sync with the later stages. The other reason is that, even when the project

stays consistent, the underlying deep learning framework is developed rapidly, and the project becomes incompatible. In our case, we had both of the problems. Moreover, we later found out that the document of the project was out-of-sync with the code. We reported this issue to the authors and confirmed that the repository is currently specific to the original authors' needs and the tutorials are outdated. In this case, we decided to write our own code from scratch, instead of trying to fix the problem.

Starting from a PyTorch Example

We highly recommend PyTorch over Tensorflow in your research phase. Most state-of-the-art deep learning algorithms are implemented in PyTorch as it is much easier to implement ideas than Tensorflow. PyTorch has an extensive tutorial so that researchers may cherry-pick them as a starting point. Since image classification is our task, we pick the image classification task tutorial and employ a pre-trained ResNet50 as the neural network architecture of choice, which is one of the most well-known for the image classification task. Then we apply a transfer-learning technique, which is to replace the final fully connected (FC) layer of the pre-trained neural network. This reduces the number of required training samples. To feed the neural network with data for training requires a matching dataset class, ImageFolder class, in PyTorch.

Experimental Results

To evaluate the performance of the deep-learning-based classifier, we ran ten randomized split/train/test runs. The training and test sets had 535 and 59 samples, respectively. Each training run halted in roughly 300 epochs. We got the results as shown in Table 1.

Table 1. Accuracies and F1-scores on the test set of the 10 randomized classification trials

Trial no.	Accuracy	F1-score
1	0.76271	0.7609
2	0.84746	0.8596
3	0.86441	0.6936
4	0.88136	0.77163
5	0.77966	0.76917
6	0.84746	0.80512
7	0.76271	0.6706
8	0.84746	0.82617
9	0.83051	0.73485
10	0.83051	0.81271
Average	0.8254±0.0423	0.770435±0.059

In the previous study by Joo *et al*. (2013), they report accuracies of 0.6649 and 0.7562 for 48-feature-classifier and 82-feature-classifier, respectively. Our results in Table 1 indicate the superiority of deep learning over the classifier based on hand-crafted features. Moreover, we expect further improvement if more deep learning techniques are applied and the hyperparameters are tuned carefully.

Discussion

The African Lake Malawi cichlid fish species, which have undergone radiative evolution in a very short evolutionary time, show a remarkable diversity in colors and various striped patterns in the appearance. From a biological point of view, biological discrimination against closely related species, such as cichlid fishes, which have diversified relatively recently and still have interspecific gene flow, cast a challenge to automatic classification methods. We adopted recently progressed AI methods to solve the difficulty and connected them for seamless automatic analysis of the cichlid fish photographic data that we used in 2013 (Joo *et al*., 2013). In this work, we employed a transfer-learning technique to replace and fine-tune the final FC layer of a pre-trained neural network. Note that the embeddings (the extracted features) of the pre-trained neural network are not real-world and may not work on some datasets. The reason is, "images found in current image datasets are not drawn from the same distribution as the set of all possible images naturally occurring in the wild" (Chang & Lipson, 2019), and we advise the readers to try multiple neural network architectures which are pre-trained on different datasets so that the reader may find better performing neural nets. Please refer to the supplementary material appended to this paper for more detailed information about our methods and image data of cichlid fishes. The supplementary material includes Linux commands and program codes to help the readers understand the deep learning research process.

One of the most important differences between the previous approach and the new deep learning method is that the direction of the feature extraction. The deep learning approach found the features in bottom-up fashion from the training samples, as opposed to the previous approach where the human engineers have somewhat arbitrarily applied a small set of features which works, by trial and error in top-down manner. To make matters worse, these top-down features had limited expressivity, as they could only be defined mathematically through a programming language.

This in turn leads to another important implication. To make the bottom-up approach feasible, the researchers must first collect sufficient high-quality data. This may require adjusting the experiment design and/or process. We suggest that this data-driven approach with AI tools

will lead to deeper understanding of the nature. Sutton (2019) argues that with current path of technology development, human feature engineering efforts are of little help and more computing resources and data are the keys of the improvements. In this spirit, we propose to employ more AI/data-driven approaches to overcome our potential narrow understanding of the nature.

In future research, the Cichlid Fishes image identification model could improve that image data set extended by reconstruction, convolution of feature extraction by neural network (ResNet50 and AlexNet, VGG19, GoogLeNet, etc.), deep learning algorithm comparison (CNNs, R-CNN, and YOLO), activation function comparison, etc.

Conflict of Interest

The authors declare that they have no competing interests.

Acknowledgments

We thank the officials and staff members of the National Institute of Ecology (NIE) for inviting us as speakers for the Forum held on September 9, 2021 entitled "The Convergence of AI and Ecology: How will Artificial Intelligence Change the Future of Ecology?" The photographic image data of Malawian cichlid fishes were kindly provided by our colleagues, Catarina Pinho and Jody Hey. A research fund for the field study in Lake Malawi was granted to Catarina Pinho from FCT (Portugal, PTDC/BIA-BDE/66210/2006). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R1I1A2A02057134).

References

Bondi, E., Fang, F., Hamilton, M., Kar, D., Dmello, D., Choi, J., *et al*. (2018). SPOT poachers in action: augmenting conservation drones with automatic detection in near real time. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 7741-7746.

Capra, F. (1996). *The Web of Life: A New Synthesis of Mind and Matter*. London: Harper Collins.

Chang, O., and Lipson, H. (2019). *Seven Myths in Machine Learning Research*. Retrieved December 9, 2021 from http://arxiv.org/abs/1902.06789.

Fegraus, E.H., Lin, K., Ahumada, J.A., Baru, C., Chandra, S., and Youn, C. (2011). Data acquisition and management software for camera trap data: a case study from the TEAM Network. *Ecological Informatics*, 6, 345-353.

Joo, D., Kwan, Y.S., Song, J., Pinho, C., Hey, J., and Won, Y.J. (2013). Identification of cichlid fishes from Lake Malawi using computer vision. *PloS One*, 8, e77686.

Kellenberger, B., Tuia, D., and Morris, D. (2020). AIDE: accelerating image-based ecological surveys with interactive machine learning. *Methods in Ecology and Evolution*, 11, 1716-1727.

Kim, K.D. (2015). Contents and prospects of ecological education. *Journal of Holistic Convergence Education*, 19, 1-19.

Lee, E.K. (2020). A comparative analysis of contents related to artificial intelligence in national and international K-12 curriculum. *The Journal of Korean Association of Computer Education*, 23, 37-44.

Lim, H.M., and Lee, S.W. (2018). A study on pre-service elementary school teachers' knowledge, awareness and attitude of the biodiversity conservation. *Journal of Korean Practical Arts Education*, 31, 19-44.

Microsoft. (2021). *CameraTraps*. Retrieved July 9, 2021 from https://github.com/microsoft/CameraTraps.

Microsoft AI. (2021). *AI for Earth*. Retrieved July 9, 2021 from https://www.microsoft.com/en-us/ai/ai-for-earth.

Ministry of Education. (2020). *Comprehensive Plan for Convergence Education that Changes the Learning Paradigm (20-24)*. Sejong: Ministry of Education.

Ministry of Science and ICT. (2019). *National Strategy for Artificial Intelligence*. Sejong: Ministry of Science and ICT.

Noh, H.J. (2003). Intrinsic value in biodiversity and moral education. *Journal of Korean Philosophical Society*, 86, 115-137.

Norouzzadeh, M.S., Morris, D., Beery, S., Joshi, N., Jojic, N., and Clune, J. (2021). A deep active learning system for species identification and counting in camera trap images. *Methods in Ecology and Evolution*, 12, 150-161.

Norouzzadeh, M.S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M.S., Packer, C., *et al*. (2018). Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, 115, E5716-E5725.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28, 91-99.

Sutton, R. (2019). *The Bitter Lesson*. Retrieved December 9, 2021 from http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Swanson, A., Kosmala, M., Lintott, C., Simpson, R., Smith, A., and Packer, C. (2015). Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. *Scientific Data*, 2, 150026.

Tabak, M.A., Norouzzadeh, M.S., Wolfson, D.W., Sweeney, S.J., Vercauteren, K.C., Snow, N.P., *et al*. (2019). Machine learning to classify animal species in camera trap images: applications in ecology. *Methods in Ecology and Evolution*, 10, 585-590.

Supplementary Materials: Identification of Cichlid Fishes from Lake Malawi Using Deep Learning

Prerequisites

In this article, we assume that the readers are familiar with deep learning terminologies and can code with Python language (Van Rossum & Drake, 2009).

Another critical component is Github. For reproducibility, we made both the code and the 594 samples publicly available on the Github repository (https://github.com/forcecore/ghoti-2021). The full raw data will be also made available on Zenodo in the near future, after a thorough review. The URL will be published on the Github repository.

Trial and Error with CameraTraps Project

In this section, we record an attempt to adapt an open-source project to our needs. Such tasks may or may not succeed, as most open-source AI projects are unstable and usually becomes un-runnable very quickly. Eventually, we failed to meet our needs. However, our failure may provide a useful debugging process that can be valuable to the readers. Uninterested readers may skip to the next section.

About CameraTraps Project

Microsoft is hosting AI for Earth (Microsoft_AI, 2021) project and CameraTraps (Microsoft, 2021) is one of the subprojects. The repository contains many useful tools for detecting images collected from motion-triggered camera traps.

Following the CameraTraps Tutorial

Environment Setup

The usual process when adopting a Github-hosted project is to follow the instructions provided by the authors. As of Sept. 2021, the project contained a tutorial that claimed the repository was capable of adaptation to custom datasets.

The first step is to setup a new “virtual environment” for running Python. We do this as each Python project may require different package (library) requirements, which can induce

"dependency hell" when not careful. Programmers bypass this problem by separating projects with virtual environments, one environment for each project. The procedure is as follows:

```
$ python --version
Python 3.9.6

$ mkdir -p ~/usr
$ python -m venv --system-site-packages ~/usr/venv-tf39
$ source ~/usr/venv-tf39/bin/activate

$ export CAMERATRAPS_DIR=`pwd`/CameraTraps
$ git clone https://github.com/Microsoft/CameraTraps.git
$CAMERATRAPS_DIR
```

From here on, we prefix the commands with dollar signs ($) to mean that you will type the commands to the system command prompt.

The export command initializes the variable for the bash shell. The variable is not persistent, meaning that if you close the terminal, you'd need to run the command again. There are many subtle Linux/Python details like this, which are beyond the scope of this article. We encourage the readers to learn these through experience.

Let us continue to the next instruction from the tutorial. It is to install the requirements by running the "conda" command as below:

```
$ conda env create -f ${CAMERATRAPS_DIR}/environment-classifier.yml
```

However, we are not using Anaconda (Anaconda_Inc., 2021) in this example and conda isn't available. Instead, we open environment-classifier.yml with a text editor and manually install the required dependencies. Some of the dependencies may be installed as below:

```
$ pip install "tensorflow>=2.3"
$ pip install NumPy
$ pip install pillow
$ pip install tqdm pycocotools
```

Whenever unavailable requirements are found, Python will raise ImportError, which can be easily solved by installing the requirement with the pip command.

CameraTraps so far seems to depend on Tensorflow, which is one of the popular deep learning frameworks. To test if it is successfully installed, you can use the python command as below:

```
$ python
Python 3.9.6 (default, Jun 30 2021, 10:22:16)
[GCC 11.1.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import tensorflow
2021-09-01 18:46:21.613069: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudart.so.11.0'; dlerror:
libcudart.so.11.0: cannot open shared object file: No such file or
directory
2021-09-01 18:46:21.613108: I
tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
>>>
```

After seeing the >>> prompt, you can press the Ctrl+D key combination to exit. In our case, Tensorflow is failing to utilize the GPU, which is hinted by libcudart.so.11.so part. While we are familiar with such error messages, the readers may have to resort to internet search engines to copy-paste the error message to determine the cause.

Since GPUs are used for faster training of neural networks, we need to address the problem at some point. However, for now, we advise the readers to defer the resolution until they are certain that the training procedure runs; the readers may discover other projects which might be easier to tame than this one.

Dataset Format

To use our dataset, we first need to figure out the data format the project expects. We need both the project documentation on the data format and the example dataset. For the latter, we follow the project's tutorial.

```
$ cd serengeti
$ BASEURL=https://lilablobssc.blob.core.windows.net/snapshotserengeti-
v-2-0
$ dest="SnapshotSerengeti.json.zip"
$ azcopy cp "${BASEURL}/SnapshotSerengeti_S1-11_v2_1.json.zip"
"${dest}"
$ unzip -q ${dest}
```

There should be one JSON file. It is not easy to open this file with ordinary text viewers, as the uncompressed JSON file is 5.2 Gigabytes in size. Instead of text viewers, we use head and tail commands:

```
$ head -n 10000 SnapshotSerengeti_S1-11_v2.1.json
$ tail -n 10000 SnapshotSerengeti_S1-11_v2.1.json
```


```
{
 "info": {
  "version": "2.1",
  "description": "Camera trap data from the Snapshot Serengeti
program, seasons 1-11",
  "date_created": "2019",
  "contributor": "University of Minnesota Lion Center"
 },
 "categories": [
  {
   "id": 0,
   "name": "empty"
  },
  {
   "id": 1,
   "name": "human"
  },
  ...
  {
   "id": 60,
   "name": "lioncub"
  }
 ],
 "images": [
  {
   "id": "S1/B04/B04_R1/S1_B04_R1_PICT0001",
   "file_name": "S1/B04/B04_R1/S1_B04_R1_PICT0001.JPG",
   "frame_num": 1,
   "seq_id": "SER_S1#B04#1#1",
   "width": 2048,
   "height": 1536,
   "corrupt": false,
   "location": "B04",
   "seq_num_frames": 1,
   "datetime": "2010-07-18 16:26:14"
  },
  {
   "id": "S1/B04/B04_R1/S1_B04_R1_PICT0002",
   "file_name": "S1/B04/B04_R1/S1_B04_R1_PICT0002.JPG",
   "frame_num": 1,
   "seq_id": "SER_S1#B04#1#2",
```

```
    "width": 2048,
    "height": 1536,
    "corrupt": false,
    "location": "B04",
    "seq_num_frames": 1,
    "datetime": "2010-07-18 16:26:30"
   },
   ...
   {
    "sequence_level_annotation": true,
    "id": "453e191b-91f8-11e9-bd66-000d3a198845",
    "category_id": 1,
    "seq_id": "SER_S11#T11#1#131",
    "season": "SER_S11",
    "datetime": "2015-11-18 10:23:36",
    "subject_id": 31711358,
    "count": NaN,
    "standing": 0.7,
    "resting": 0.0,
    "moving": 0.4,
    "interacting": 0.0,
    "young_present": 0.0,
    "image_id": "SER_S11/T11/T11_R1/SER_S11_T11_R1_IMAG0335",
    "location": "T11"
   },
   {
    "sequence_level_annotation": true,
    "id": "453e191c-91f8-11e9-bacf-000d3a198845",
    "category_id": 1,
    "seq_id": "SER_S11#T11#1#131",
    "season": "SER_S11",
    "datetime": "2015-11-18 10:23:36",
    "subject_id": 31711358,
    "count": NaN,
    "standing": 0.7,
    "resting": 0.0,
    "moving": 0.4,
    "interacting": 0.0,
    "young_present": 0.0,
    "image_id": "SER_S11/T11/T11_R1/SER_S11_T11_R1_IMAG0336",
    "location": "T11"
   }
  ]
}
```


Their meaning and description are in the README file located in data_management directory:
https://github.com/microsoft/CameraTraps/blob/master/data_management/README.md

Rather than writing a program to create this dataset description immediately, we recommend writing a minimum dataset of two samples by hand to make sure that we are on the correct path, before scaling up to the full samples. Our 2-sample dataset, $GHOTI/data/ghoti.json looks like the following:

```
{
  "info": {
    "version": "0.1",
      "description": "GHOTI",
      "date_created": "2021",
      "contributor": "Deokjin Joo, Ye-seul Kwan, Jongwoo Song,
Catarina Pinho, Jody Hey and Yong-Jin Won"
  },

  "categories": [
    { "id": 0, "name": "gm_f" },
    { "id": 1, "name": "lf_m" }
  ],

  "images": [
    { "id": "08-0128", "file_name": "gm_f/08-0128.jpg" },
    { "id": "08-0311", "file_name": "lf_m/08-0311.jpg" }
  ],

  "annotations": [
    { "id": "08-0128", "image_id": "08-0128", "category_id": 0 }
    { "id": "08-0311", "image_id": "08-0311", "category_id": 1 }
  ]
}
```

Then we use the command from the tutorial to convert this description into the dataset that the deep learning script expects:

```
GHOTI=$HOME/work/ghoti-2021
cd ~/work/ghoti-
2021/CameraTraps/data_management/databases/classification

python make_classification_dataset.py \
    $GHOTI/data/ghoti.json \    # input_json
    $GHOTI/data/ \                          # image_dir
    $MEGADETECTOR_PB \                       # frozen_graph
    --coco_style_output $COCO_STYLE_OUTPUT \
    --tfrecords_output $TFRECORDS_OUTPUT \
    --location_key location
```

The script does not work and emits the following error:

```
Traceback (most recent call last):
  File "/home/jdj/work/ghoti-
2021/CameraTraps/data_management/databases/classification/make_classifi
cation_dataset.py", line 329, in <module>
    def run_detection(sess: tf.Session,
AttributeError: module 'tensorflow' has no attribute 'Session'
```

It is common for deep learning research projects to fail over time, for two reasons. One is the researchers move onto some other phases of research and previous ones get out of sync with later stages. The other reason is that in deep learning projects, the underlying deep learning framework is developed rapidly, and the project no longer runs on the new version of the framework. Googling the error message, we now know that we need to replace all "import tensorflow as tf" like the following:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

Whenever you encounter errors related to tf.session, fix the "import tensorflow" line as above.

Having done that on many Python files, now we encounter a new error, "KeyError: 'location'". Our manually created JSON file doesn't have the location key for each sample. (A key is the left-hand side of the colon in the dictionary item.) For example, we added "location": "dummy" for 08-0128.jpg as below:

```
{ "id": "08-0128", "file_name": "gm_f/08-0128.jpg", "location":
"dummy" },
```

We repeatedly ran the dataset compilation script. Each run revealed new errors. We had to rename the key from "annotation" to "annotations".

```
File "/home/jdj/work/ghoti-
2021/CameraTraps/data_management/databases/classification/make_classifi
cation_dataset.py", line 509, in save_outputs
    imsize = [cur_image['height'], cur_image['width']]
```

Now the dataset creator needs “height” and “width” information for each image. We only have two images to correct, for now.

```
Use tf.gfile.GFile.
WARNING: No images were written to tfrecords
```

The fixes worked. There’s one warning from the preprocessor as above.

```
~/work/ghoti-2021/mydataset $ du -sh *
128K    cropped_coco_style
100K    cropped_tfrecords
```

A check on file sizes seems to imply that the conversion worked.

Making the Full Version of the JSON File

It takes a lot of effort to create the entries manually. We created a script to generate entries for each sample like the format we corrected in the previous subsections. The reader may refer to 11_make_json.py in our Github repository. With the full version of the JSON file created, we may rerun the dataset creation script.

Dataset Statistics

The CameraTraps tutorial recommends that we see the dataset statistics. The readers may create a bash script to ease the execution as follows.

```
#!/bin/bash

GHOTI=`pwd`
DATASET_DIR=$GHOTI/mydataset
COCO_STYLE_OUTPUT=$DATASET_DIR/cropped_coco_style
TFRECORDS_OUTPUT=$DATASET_DIR/cropped_tfrecords

CAMERTRAPS_DIR=$GHOTI/CameraTraps
MEGADETECTOR_PB="${CAMERTRAPS_DIR}/pbs/md_v4.1.0.pb"

cd $CAMERTRAPS_DIR/data_management/databases/classification

python cropped_camera_trap_dataset_statistics.py \
    $GHOTI/data/ghoti.json \
    $COCO_STYLE_OUTPUT/train.json \
    $COCO_STYLE_OUTPUT/test.json \
```

```
    --classlist_output $COCO_STYLE_OUTPUT/classlist.txt
```

We saved the script as 21_dataset_stats.sh. To run the script, run “bash 21_dataset_stats.sh” without quotes. The execution may look something like below:

```
$ bash 21_dataset_stats.sh
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Statistics of the training split:
Locations used:
[]
In total 12 classes and 0 images.
Classes with one or more images: 0
Images per class:
ID    Name            Image count
    0 gm_f                      0
    1 lf_m                      0
    2 mv_f                      0
    3 pe_m                      0
    4 pf_f                      0
    5 pg_f                      0
    6 tg_f                      0
    7 tg_m                      0
    8 tm_f                      0
    9 tm_m                      0
   10 toc_f                     0
   11 toc_m                     0


Statistics of the testing split:
Locations used:
['dummy']
In total 12 classes and 644 images.
Classes with one or more images: 12
Images per class:
ID    Name            Image count
    0 gm_f                     13
    1 lf_m                     12
    2 mv_f                     11
    3 pe_m                     37
    4 pf_f                     11
    5 pg_f                     26
    6 tg_f                    192
    7 tg_m                    100
    8 tm_f                     24
    9 tm_m                     17
   10 toc_f                   122
```

```
   11 toc_m                    79
```

We found a problem. The training set is empty! Our examination of the scripts in CameraTraps revealed that they use the "location" key of the image sample for the train/test set partitioning. This made us to partition the dataset manually by binning each sample into either one of location=test or location=training bins. This can be accomplished by the following lines of code:

```
def split_train_test(df: pd.DataFrame, random_state: int, test_size:
Union[float, int]) -> pd.DataFrame:

    _df_train, df_test = sklearn.model_selection.train_test_split(df,
random_state=SEED, test_size=TESTSET_SIZE, stratify=df["category"])
    df["location"] = "train"  # Initially set everyting as train set.
    df.loc[df_test.index, "location"] = "test"
    return df
```

Here, we use sklearn's train_test_split function. It has the "stratify" option which considers the samples' classes so that the training and test sets have the same distributions of classes. Also, note that we use Pandas library's DataFrame to manage the dataset. We recommend that the readers learn Pandas, as many projects utilize this library.

Unfortunately, we couldn't specify which locations belong to the train or test set. We had to modify the dataset creation script to fit our needs – that's why we encourage the readers to learn Python.

Near line 280 of make_classfication_dataset.py,

```
# test_locations = sorted(
#     random.sample(locations, max(1, int(test_fraction *
len(locations)))))
# JOO: train_locations = sorted(set(locations) - set(test_locations))
train_locations = ["train"]  # JOO
test_locations = ["test"]  # JOO
```

We made the above changes. The original script randomly selected the locations, with the implicit assumption that the locations have the same distributions of species. Since we partitioned the datasets by ourselves, we hard-coded the train and test locations with the above modifications. The statistics now display more acceptable outputs:

```
Statistics of the training split:
```

```
Locations used:
['train']
In total 12 classes and 516 images.
Classes with one or more images: 12
Images per class:
ID    Name            Image count
    0 gm_f                     10
    1 lf_m                     10
    2 mv_f                      9
    3 pe_m                     28
    4 pf_f                      9
    5 pg_f                     21
    6 tg_f                    153
    7 tg_m                     81
    8 tm_f                     19
    9 tm_m                     14
   10 toc_f                    97
   11 toc_m                    65


Statistics of the testing split:
Locations used:
['test']
In total 12 classes and 128 images.
Classes with one or more images: 12
Images per class:
ID    Name            Image count
    0 gm_f                      3
    1 lf_m                      2
    2 mv_f                      2
    3 pe_m                      9
    4 pf_f                      2
    5 pg_f                      5
    6 tg_f                     39
    7 tg_m                     19
    8 tm_f                      5
    9 tm_m                      3
   10 toc_f                    25
   11 toc_m                    14
```

Training on the Dataset

CameraTraps tutorial states that we run the following commands to download a pretrained inception v4 network.

```
$ PRETRAINED_DIR=`pwd`/pretrained
$ mkdir $PRETRAINED_DIR && cd $PRETRAINED_DIR
$ wget -O inc4.tar.gz
$ http://download.tensorflow.org/models/inception_v4_2016_09_09.tar.gz
```

```
$ tar xzf inc4.tar.gz
```

And the following commands are supposed to be run:

```
cd $CAMERATRAPS_DIR/classification/tf-slim/datasets/
cp cct.py serengeti.py
```

Unfortunately, there's no cct.py. The tutorial document is out of date. After examination, we found that for training, CameraTraps/classification/train_classifier.py is used in the current codebase. Although we downloaded inc4.tar.gz already, this is no longer in use, judging from the contents in train_classifier.py code. Another thing we noticed is that they've moved onto the PyTorch framework from Tensorflow. As a side note, we highly recommend PyTorch over Tensorflow in your research phase. Most state-of-the-art deep learning algorithms are implemented in PyTorch as it is much easier to implement ideas than Tensorflow. Even if you are to run the algorithm on limited hardware in the future, it is wiser to defer the optimization to the later stages of the research and development.

Training Attempt with Undocumented Code

Having known that CameraTraps/classification/train_classifier.py is the training script in the most recent version of the code, and the tutorial isn't covering it, the only way is to run the script and try to guess what its inputs are. Running the script emits the following error:

```
FileNotFoundError: [Errno 2] No such file or directory:
'mydataset/cropped_coco_style/classification_ds.csv'
```

It seems the training script now requires an unexpected format of the training dataset, while there are mentions of COCO style datasets on the front page of the CameraTraps project, as if it is still usable on this project.

We tried both train_classifier.py and train_classifier_tf.py. The former is a PyTorch implementation, and the latter is a TensorFlow implementation. The latter is no longer in development as commented inside the script file, due to the convenience of PyTorch. Unfortunately, neither runs.

Googling "CameraTraps classification_ds.csv" yielded this file:
https://github.com/microsoft/CameraTraps/blob/master/classification/README.md

The documentation is rather specific to the original authors' datasets and there are no instructions for customizing the inputs. At this point, we advise the readers to find better alternative projects. Another possible choice is to report this issue to the authors. The readers can create their own Github account and file an issue report. We did so and got a confirmation that the repository is currently specific to the original authors' needs and the tutorials are outdated, as can be seen here:
https://github.com/microsoft/CameraTraps/issues/259

Starting from a PyTorch Tutorial Example

The readers may visit the tutorials page of PyTorch framework (https://pytorch.org/tutorials/), and start from the closest matching tutorial, as each tutorial covers unique AI tasks. For example, there are tutorials for image classification, object detection, and audio feature extraction. Most recently, there's a higher-level framework built on top of PyTorch, namely, PyTorch Lightning. This greatly reduces the amount of coding.

Defining the Neural Network Architecture

Since image classification is our task, we select ResNet, which is one of the most well-known neural network architectures for the task.

```
import pytorch_lightning as pl
import torch
from torch import nn
import torchvision

class MyResnet(pl.LightningModule):
    def __init__(self, num_classes: int):
        super().__init__()
        # model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50',
pretrained=True)  # Causes timeout error
        model = torchvision.models.resnet50(pretrained=True, progress=T
rue)
        model.fc = nn.Linear(2048, num_classes)  # Replace the classifi
cation layer
        self.resnet = model
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, x):
        return self.resnet.forward(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
```

```
        y_pred = self.forward(x)
        loss = self.criterion(y_pred, y)
        self.log("train_loss", loss)
        return loss

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=3e-4)
```

We start by adding instantiation code for resnet50, where 50 is the number of neural network layers. The definition of the architecture must first inherit pl.LightningModule, as the code above.

Here, we intend to replace the final fully connected (FC) layer of the pre-trained resnet50 model so that we can apply a *transfer-learning* technique. Specifically, we reuse the pre-trained resnet50 model, except for the final FC layer. The reused part of the neural network will be "frozen" so that its weight will not be altered during the training. That way, it will act as the feature extractor. On top of it, we attach our own FC layer which will be trained and adapted to make it perform classifications on the cichlid classes. We redirect readers to see the references (Course_Website, 2021 p. 231) and (Inkawhich, 2021) for details on this technique.

forward() function must define how the previously defined neural network computes its output. Since we only created a ResNet instance and no other components are used, our forward function only runs ResNet's forward function.

training_step() function defines how the loss is computed for the input batch. In our case, the batch variable is a pair of X (tensor representing an image) and Y (class labels). We split the batch into x and y first then run the forward function. The difference between the prediction result, y_pred, and the correct answer, y, is computed as the loss by self.criterion().

Dataset Class for Feeding the Data

To feed the neural network with data for training requires a matching dataset class, in PyTorch. The easiest ready-made class in our case torchvision.datasets.ImageFolder class.

```
    transforms = tv.transforms.Compose([
        tv.transforms.ToTensor()
    ])

    dataset = tv.datasets.ImageFolder(
        root="./data",
```

```
        transform=transforms
    )
```

ImageFolder class expects the root directory of the image collection as the “root” parameter. After its instantiation, the dataset may be examined as follows:

1. print(dataset.class_to_idx) to see what classes are defined and with their enumerated class ids are. In our case, we have {'gm_f': 0, 'lf_m': 1, 'mv_f': 2, 'pe_m': 3, 'pf_f': 4, 'pg_f': 5, 'tg_f': 6, 'tg_m': 7, 'tm_f': 8, 'tm_m': 9, 'toc_f': 10, 'toc_m': 11}.
2. Individual samples by printing the samples with the for loop:
   for x, y in dataset: print(x.shape)

Putting Pieces Together and Training

We didn’t split the dataset into train, validation, and test sets yet. However, to see if all the classes are defined correctly, we’ll run the training.

```
net = MyResnet()

transforms = tv.transforms.Compose([
    tv.transforms.ToTensor(),
])

dataset = torchvision.datasets.CocoDetection(
    root="./data",
    transform=torchvision.transforms.ToTensor()
)

train_loader = DataLoader(dataset)
trainer = pl.Trainer()
trainer.fit(net, train_loader)
```

The training fails, as (1) the training sample images have different sizes, and (2), ResNet expects input size of width=224, height=224. We can solve the problem by applying the resize transform.

```
transforms = tensorvision.transforms.Compose([
    tensorvision.transforms.ToTensor(),
    tensorvision.transforms.Resize((224, 224))
])
```

The training loop should start working.

Applying the Transfer Learning Technique

For transfer learning, we'd freeze all but the final fully connected layer which we have replaced with our own. Just before pl.Trainer() line, we write the following lines:

```
# Pretrain the FC layer before doing the full training
    net.freeze()
    for parameters in net.resnet.fc.parameters():  # resnet.fc
doesn't have unfreeze() so...
        parameters.requires_grad = True
```

We first freeze all the layers by running net.freeze(). After that, we unfreeze the final FC layer. We may run net.fc.unfreeze(), when it works. At this time, it doesn't and we had to use the for loop to set requires_grad = True. As a result, the pre-trained ResNet will act as a feature extractor and the FC layer will do the classification task for cichlids, after the training.

Making the Training Proper

Although we can run the training loop, we have further work to do. Some of them are

1. Splitting the dataset into training and validation sets,
2. There are too many hard-coded configurations that are better moved into a separate configuration file.
3. We highly recommend the Hydra framework (https://hydra.cc) or OmegaConf (https://github.com/omry/omegaconf).
4. Apply Data Version Control (https://dvc.org/) to ease the control of data flow.

Applying all these in this article would only distract the readers from learning the basic usage of PyTorch Lightning. Instead, we advise the readers to visit our Github repository here: https://github.com/forcecore/ghoti-2021.

Experimental Results

To evaluate the performance of the deep-learning-based classifier, we ran 10 randomized split/train/test runs. The training and test sets had 535 and 59 samples, respectively. While we specified the random seed for train/test set splits, the readers will acquire different results than ours for two reasons. One is that we didn't specify the random seed for neural network initialization and the other is that some of the neural network operations are not deterministic, due to the execution mechanism on the GPU. During the training, we used hyper-parameters as in Table 1 and acquired the results shown in Table 2.

Table 1. Hyperparameters of the experiment

Hyperparameter	Value	Description
test_ratio	0.1	We partition the samples so that 0.1 (10%) of the samples will be used as the test set and the remaining ones become the training set.
split_seed	An int, 43 to 52	We use this seed value to run the 10 randomized trials
batch_size	64	Also known as mini-batch. In deep learning, we stream the samples into the neural network due to memory limitations. Since we have 535 training samples, there will be 9 mini-batches (=ceil(535/9)).
max_epochs	1000	We run the training for a maximum of 1000 epochs. If the neural network has been fed with all 535 training samples, it is called one epoch.
patience	50	During training, when there's no improvement on some metric (accuracy, in our setup) for these many epochs, the training halts.
learning_rate	3e-4 (3×10^{-4})	This parameter determines how much the weights in the neural network get adjusted after seeing each batch. Due to numerical precision, the researchers must find a good number that works by trial and error. Nonetheless, we use 3e-4, which is a magic number known in the deep learning community (Karpathy, 2019).

With our setup, the training halts in roughly 300 epochs. Also note that, since we omit hyper-parameter tuning, the validation set isn't used. However, the readers should have the validation set during the hyperparameter tuning.

Table 2. Accuracies and F1 scores on the test set, 10 randomized classification trials

Trial no.	Accuracy	F1-score
1	0.76271	0.7609
2	0.84746	0.8596
3	0.86441	0.6936
4	0.88136	0.77163
5	0.77966	0.76917
6	0.84746	0.80512
7	0.76271	0.6706
8	0.84746	0.82617
9	0.83051	0.73485
10	0.83051	0.81271
Average	**0.8254±0.0423**	**0.770435±0.059**

In (Joo *et al.*, 2013), they report accuracies of 0.6649 and 0.7562 for 48-feature-classifier and 82-feature-classifier, respectively. Our results in Table 2 indicate the superiority of deep learning over the classifier based on hand-crafted features; the accuracy is 0.8254 with the deep learning methods. We expect further improvement if more deep-learning techniques are applied.

Discussions

In this work, we employed a transfer-learning technique, which was to replace and fine-tune the final fully connected layer of a pre-trained neural network. Note that the *embeddings* (the extracted features) of the pre-trained neural network does not real-world and may not work on some datasets. The reason is, "images found in current image datasets are not actually drawn from the same distribution as the set of all possible images naturally occurring in the wild" (Chang & Lipson, 2019), and we advise the readers to explore other neural network architectures which are pre-trained on different datasets, so that configurations that fit the research needs may be found.

References

Anaconda_Inc. (2021) Anaconda Software Distribution, Anaconda documentation, https://docs.anaconda.com.

Chang O, and Lipson H (2019) Seven myths in machine learning research. *ArXiv190206789 Cs Stat*, http://arxiv.org/abs/1902.06789.

Course_Website (2021) CS231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/transfer-learning/

Inkawhich N (2021) Finetuning Torchvision Models — PyTorch Tutorials 1.2.0 documentation. https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

Joo D, Kwan Y-S, Song J, Pinho C, Hey J, and Won Y-J (2013) Identification of cichlid fishes from Lake Malawi using computer vision. *PloS One,* 8**,** e77686.

Karpathy A (2019) A Recipe for Training Neural Networks. https://karpathy.github.io/2019/04/25/recipe/

Microsoft (2021) CameraTraps, https://github.com/microsoft/CameraTraps.

Microsoft_Ai (2021) AI for Earth, https://www.microsoft.com/en-us/ai/ai-for-earth.

Van Rossum G, and Drake F (2009) Python 3 Reference Manual. Scotts Valley, CA: CreateSpace; 2009.