Can we use Transfer Learning to Implement a Convolutional Neural Network to do Large Mammal Classification on an Edge Device?

Noah Cole & Dr. Ryan Botts

Point Loma Nazarene University

Table of Contents

Abstract

Conservation efforts of large mammal species in Costa Rica rely on extensive camera tracking networks to record these species’ presence, activity, and interactions. By using transfer learning on an edge device to retrain a Convolutional Neural Network the process of tracking and identifying these mammals will be streamlined. Transfer learning on edge devices was found to be effective in retraining and deploying CNN image classifiers.

Introduction

Why:

Costa Rica is one of the most biodiverse countries in the world. By providing the Costa Rican government with reliable and extensive data on their native wildlife's presence, activity, and interactions, more effective conservation efforts can be enacted. Dr. Mooring from Point Loma Nazarene University, with assistance from Dr. Botts and PLNU summer researchers, is in the process of enacting this research, using extensive camera tracking networks to record data on large mammals in the Costa Rican Jungle. The current process for properly tagging the collected images is painstaking and inefficient: researchers must search the images and identify the species manually, resulting in many different formats for how these pictures are tagged, misidentified images, and inconsistent storing of images. Additionally, the images must be stored on the camera's hard drives because transmitting those from Costa Rica to San Diego is not feasible. This means that the research team must travel down to Costa Rica to retrieve the data by hand, before it can be labelled and analyzed.

These problems are not just limited to PLNU's research team, but apply to other camera trap studies as well. Memory limitations from storing a large number of detailed photos are common, and require many hours to maintain cameras and swap out memory cards. Additionally, once images are actually captured and put into the hands of the researchers, many man-hours are needed to identify and organize the animals by species.

How:

Edge devices are low-powered Central Processing Units (CPU) built for specific computational tasks. The edge device chosen for our research was a Raspberry Pi, similar to most edge devices in that it is a small computer that can be deployed easily, providing a low-power processor in areas that need a smaller piece of hardware. Raspberry Pi's are also able to integrate cameras into their systems, which can be purchased online easily and cheaply, with options for different lenses and IR image capturing as well. Edge devices have been shown to be an effective way to minimize the amount of unnecessarily stored data through filtering out unneeded data by using edge computation (Long et al.). Alternatively, in Wang et al. they were used to store only local data that would be deleted once decisions were made by the machine learning system, effectively minimizing stored data as well. Thus, we propose that the use of edge devices may be a feasible approach for limiting the amount of data recorded in the field.

Machine learning is a data analytics method centered around automating analytical model building. Additionally, "it is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention" ("Machine Learning: What it is and why it matters"). One of machine learning's applications is in image classification, where analytical models are developed to analyze and identify images. Afkham et al. used machine learning image classification on "realistic images of animals in complex natural environments," showing that image classification algorithms can be applied to identify wildlife in nature. By using classification algorithms to identify wildlife species, we can streamline the process by eliminating the need for every image to be classified manually by researchers.

One of the many machine learning algorithms is a Convolutional Neural Network (CNN). CNN's differ from other algorithms in their architecture: CNN's use a series of layers that perform mathematical calculations on the input (an image) and result in a calculated label for that image. These layers alternate between Convolution and Pooling layers—explained later in Methodology: *Convolutional Neural Networks*. The underlying commonality between CNN's is their architecture and the fact that they "have been proven very effective in areas such as image recognition and classification" (Ujjwalkarn). However, CNN's can be applied to many different use cases in Image Recognition. For instance, Cao et al. used CNN's to perform Marine Animal Classification and Lawrence et al. applied a CNN for human facial recognition. We propose the use of machine learning image classification and a CNN to identify and label the images of Costa Rican wildlife acquired in the field.

One method of developing and testing machine learning models is transfer learning. In transfer learning, instead of building a model from scratch on a very specific dataset, pre-developed models made for a different, but similar, task are 'retrained' on the new dataset. Shao et al. describe this well by writing "Transfer Learning addresses such cross-domain learning problems by extracting useful information from data in a related domain and transferring them for being used in target tasks." For identifying mammals, this could mean using models developed for mammals in a different habitat, or even for general animal classification. Thus, we propose the application of transfer learning to maximize efficiency in the development and deployment of our CNN's.

Using an edge device represents both a possible obstacle, as well as key advantage to solving this problem. Edge devices, such as a Raspberry Pi, are very low-power devices, and do not have advanced Graphics Processing Units (GPUs). GPUs are specifically designed to aid in the graphics of a computer by displaying or changing the Red/Green/Blue color counts of each pixel. This RGB per pixel structure results in a 3-Dimensional Matrix (3 values per pixel). Similarly, Neural Networks (including CNNs) possess a similar matrix structure; although the dimensions can go much higher than just 3. Due to this, GPUs turn out to be highly effective at training and deploying Neural Networks. As the edge device is a key component of our proposed research, and it lacking a GPU represents a large problem. Fortunately, Google and other companies have developed what they call a TPU—Tensor Processing Unit—designed to do exactly what we are trying to accomplish: “The TPU is a coprocessor optimized for handling neural networks, intended to push artificial intelligence out from the centralized clouds to embedded devices” (Cass par. 4). We propose the use of a TPU configure on an edge device to assist in the transfer learning process and deployment of our CNN.

Our desired project outcomes are as follows:

1. Build and configure a low-powered edge device with the ability to acquire images and apply machine learning image classification techniques, resulting in a much lower data footprint.
2. Perform transfer learning on a pre-trained CNN to adapt it for this data.
3. Assess the performance of the pre-trained CNN for image classification on this device.

Determining Success:

First, the edge device and associated hardware need to work independently. In order for this project to successfully deploy a model on edge, the edge device needs to work independently; without it, the model will have to be deployed locally and the data will still have to be retrieved manually.

Second, the edge devices involved must successfully execute transfer learning on a CNN. The process of transfer learning does not need to be done on an edge device; the CNN could be retrained on a larger machine and then the best model could be stored on the edge device. However, we would like to also demonstrate that the transfer learning process could be efficiently done on an edge device with the aid of a TPU. Failure to retrain on the edge device would not result in project failure but being able to do so would provide a valuable demonstration for the abilities of TPUs.

Finally, the project should develop a model that can confidently take in new images and label them accurately. The model does not need a 100% success rate, few deployed in real life have that kind of threshold. It does need to be able to identify a large percent of images accurately and provide metrics on which species are harder for it to identify. Providing the researchers with which species typically get confused will allow them to highlight those species and label them manually, still saving time by limiting the number of images that need labelling from the researchers.

Methodology

Hardware:

A Basic Starter Kit from *Vilros* was purchased, containing a Raspberry Pi Project Board, compact case, power supply, and heat sink. The Raspberry Pi Project Board was comprised of a Raspberry Pi 3 Model B Rev 1.2, which I embedded in the transparent compact case. The Raspberry Pi has an ARMv7 rev 4 Processor and a CPU Revision 4. The heat sink was also attached to the Raspberry Pi, which helps with overheating by directing the flow of heat through the heat sink and into the air, rather than dispersing it on the Raspberry Pi board itself.

A 128GB Micro SD card and an SD Card adapter were also used to install Raspbian, the standard operating system for a Raspberry Pi. This process involves formatting the card, downloading NOOBS (New out of the box software)—Raspberry Pi's official installer for Raspbian, inserting the Micro SD Card into the Raspberry Pi and allowing the installer to perform the installation process automatically.

A Raspberry Pi Remote Camera module was also purchased to integrate into the Raspberry Pi 3The process of setting up these remote cameras is also well documented online: https://thepihut.com/blogs/raspberry-pi-tutorials/16021420-how-to-install-use-the-raspberry-pi-camera.

A remote 1TB Hard Drive was also used to store all the data and plug it into the Raspberry Pi, as opposed to trying to store it all on the Raspberry Pi itself. This has no impact of the successfulness of the edge device or transfer learning; the data simply needs to be present in some fashion on the Raspberry Pi to prove that transfer learning can be done on an edge device. In application, once the model is trained, the storage device no longer needs to be attached to the Raspberry Pi, as storing old data is not needed.

The final piece of hardware incorporated into this system was the TPU. A USB Accelerator TPU from Coral was purchased. The TPU allows for a Raspberry Pi to efficiently conduct machine learning processes, without the TPU, the Raspberry Pi would not be able to train, retrain, or deploy Machine Learning (ML) models. This has to do with the structure of a ML model, explained more below in *Software*. For now, the ML model structure can be compared to Red-Green-Blue counts for the colors of pixels. In this example, each pixel can be considered to be on an XY plane (flat). The Red, Green, and Blue counts are stacked upon the XY plane in the Z-axis, thus providing a third dimension to each pixel and forming a matrix. This can be visualized below in the 6-by-6-by-3 RGB matrix. The Raspberry Pi itself is a CPU, built to do a wide range of tasks, and thus is not capable of performing matrix operations efficiently on its own. However, GPUs are designed specifically to perform matrix operations well, supplementing the CPU. TPUs can be considered to be further optimized, small scale GPUs, designed specifically for ML model computations. Since CNN's require millions of such calculations, and we are doing this on a low-powered device, the specialized hardware of the TPU is required for efficient implementation.

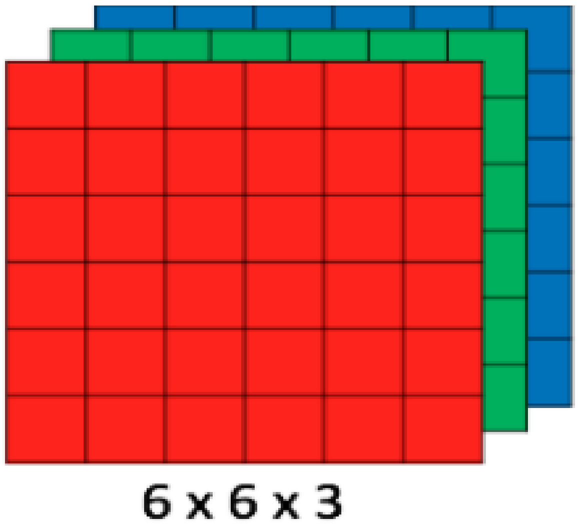

Figure 1: A 3D matrix of colored pixels

(Prabhu)

Data:

The data we are analyzing is composed of 15 different species, each organized into separate folders: Agouti, Coati, Cottontail, Coyote, Dog, Jaguar, Margay, Ocelot, Oncilla, Paca, Peccary, Puma, Tamandua, Tapir, and Tayra. These simplified folder names allow the retraining program to automatically grab the labels from each of these folders. There is a total of approximately 26,000 files with a combined size of 24 GB. The images were taken both by infrared cameras at night and normal cameras during the day (figures 2a & 2b).

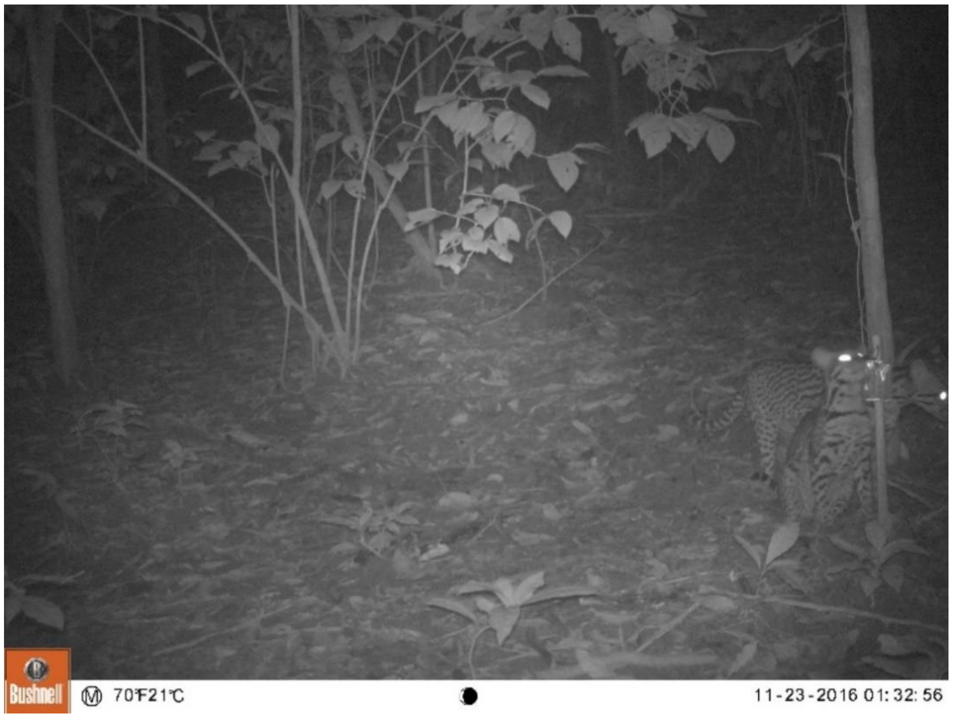

Figure 2a: Night Capture of Ocelots

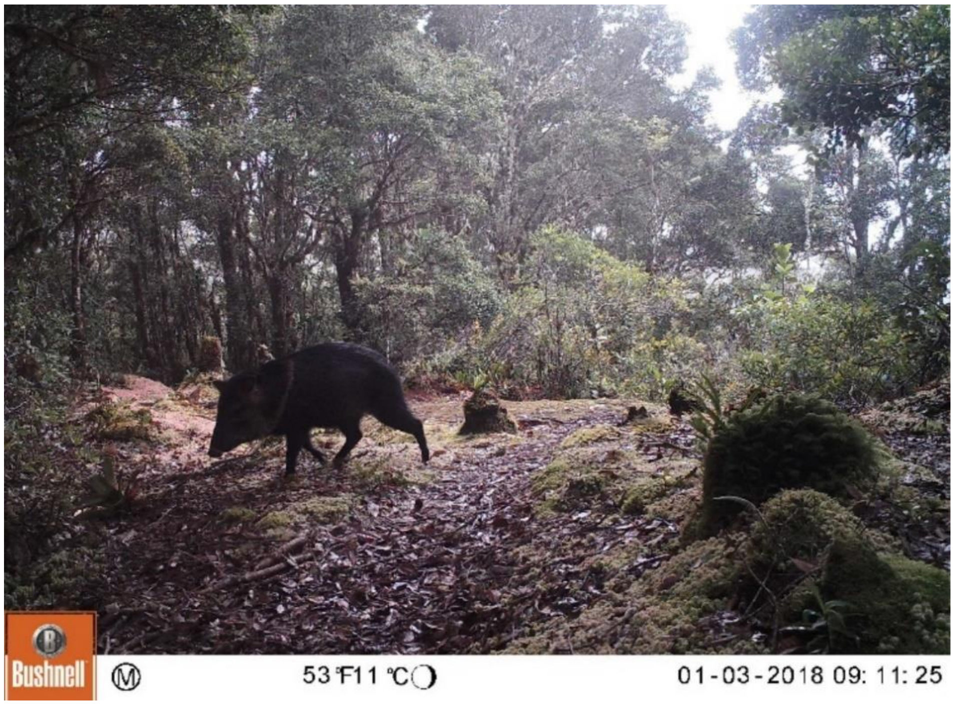

Figure 2b: Daytime Capture of Peccary

The data we are using has already been labelled and organized by the researchers. This makes our problem a Supervised Learning problem. This means that our data is already given labels and our model tries to learn from those pre-determined labels, rather than working with unlabeled data. For our problem, it is important to take the data labels given to the images as perfectly true, as Supervised Learning algorithms take in the labels and sets them as our ground truth. The ground truth is the model's way of saying that all these predetermined labels are considered to be 100% accurate, and thus is the entire basis for which our model retrains, learns, and classifies. However, over the course of this project we found that the labels provided were not 100% accurate, resulting in a false ground truth for our model's retraining. Likely this error is due to how the files were stored and captured, not directly due to mislabeling by humans.

There are a couple ways that the current data is mislabeled, which will affect the performance of the transfer learning and CNN. The first involves images that do not actually contain the animal they were labelled as. For instance, in Figure 3a. we can see an Agouti—the small rodent-like animal in the right corner—along with some deer. In Figure 3b, we have a photo captured directly after that one. In this photo however, the Agouti is nowhere to be seen, all we have left are the deer. The first photo is quite valuable, as we can not only see the species in question, but also the other animals it is interacting with. The second photo, however, is detrimental to the retraining of our CNN because its learning is based off of that photo containing an Agouti.

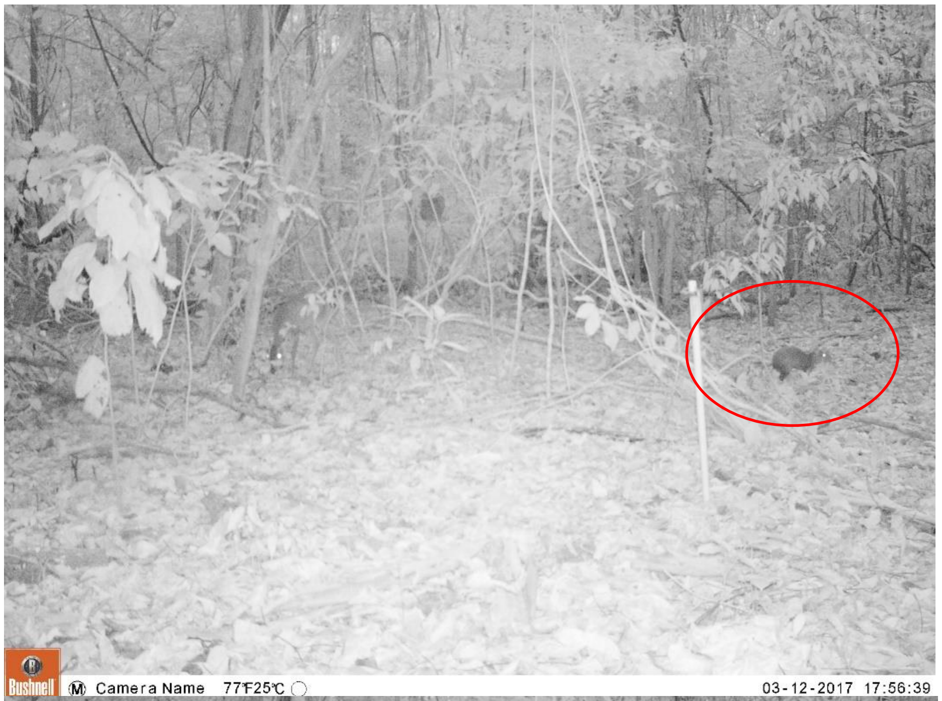

Figure 3a: An Agouti and some deer

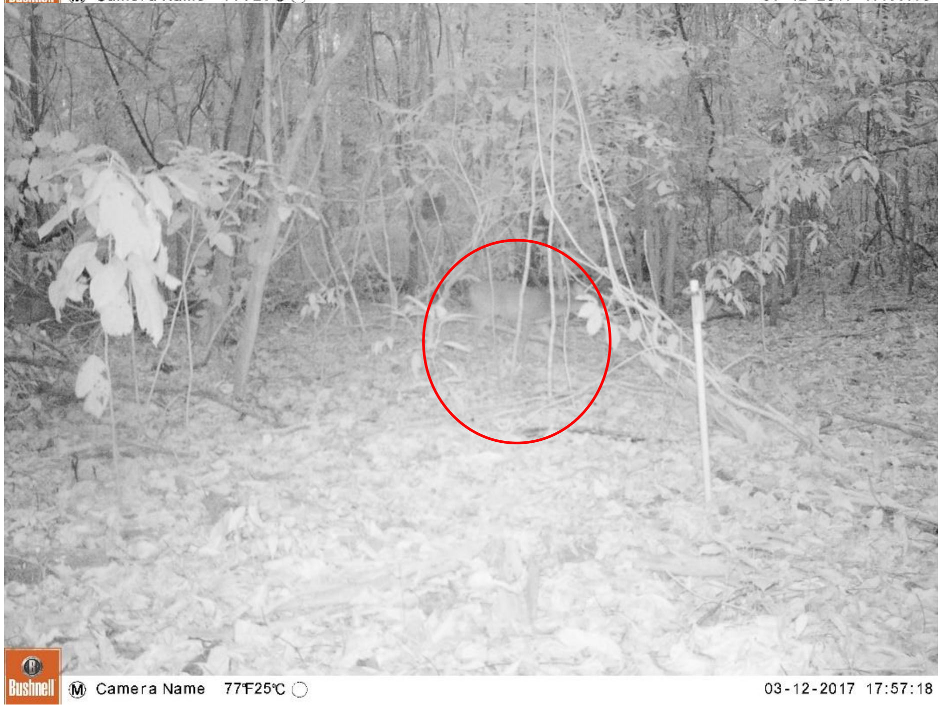

Figure 3b: A deer labelled as "Agouti," with no Agouti in frame

The other mistake involves images that do not actually have the species in frame. This typically occurs when an animal is on the edge of the frame in the previous picture. The way the cameras are set up involves photos being taken in bursts of 3. When labelling, the series of 3 images is frequently classified by researchers as whatever species appeared in one of the frames An example of this can be seen below, where in Figure 4a, a Puma is just exiting the frame and triggers the motion sensor, resulting in Figure 4b where no animal is actually present in frame.

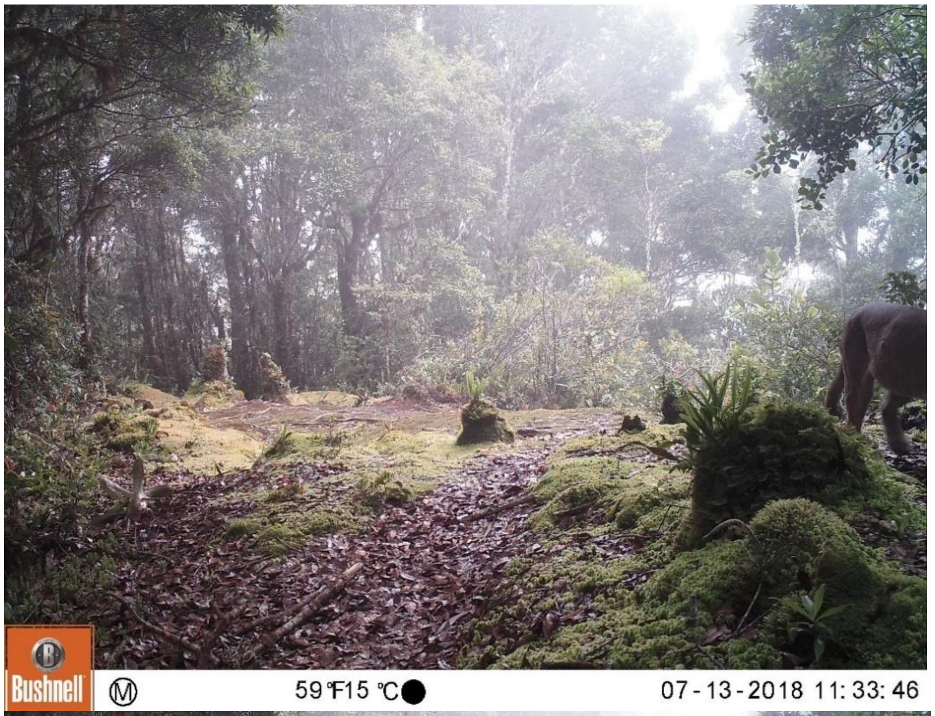

Figure 4a: Puma on right side of frame

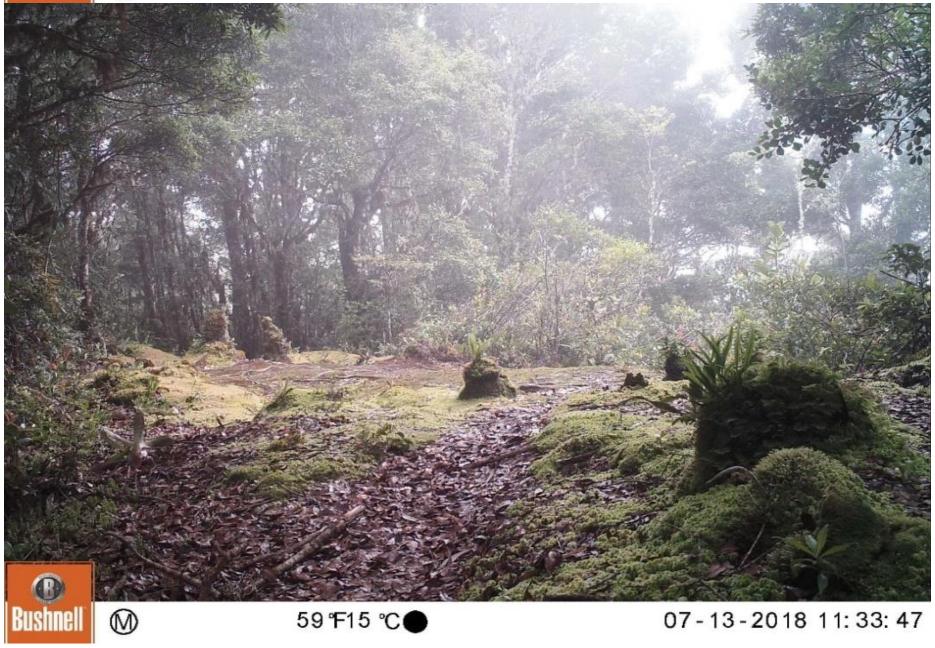

Figure 4b: No animal in frame

All of these shortcomings in the data have the potential to disrupt the transfer learning process, as the CNN will attempt to find patterns or associations within the bad data as well. Thoroughly and properly cleaning the data would provide great assistance to the model, allowing it to make accurate and reliable decisions/predictions on clean data. Unfortunately, this was not possible in the time of this project, resulting in reduced training accuracy for the CNN. A future step for the continuation of this project would involve cleaning the data first, then enacting the same procedures explained in this paper.

Software:

The Raspberry Pi was deployed with a final Raspbian version of Version ID 10, codenamed “Buster”, and a final Debian version of 10.2. Debian is also known as Debian GNU/Linux, and is a universal, open-source operating system.

Python3 was installed on the Raspberry Pi, with a final version of 3.7.3. The 3.7.3 version was used instead of Python2 versions because of the architecture of the retraining modules we used.

TensorFlow version 1.14.0 was installed and deployed on the Raspberry Pi as well. TensorFlow is a highly used Python module, developed specifically to assist in machine learning processes. Newer versions of TensorFlow are available, such as Tensorflow2.0, but many users are having issues with downloading V2.0 remotely, so it is recommended to stay on the older versions for now.

The Python module used was downloaded from GitHub from the Raspberry Pi’s command line. The module itself can be found at https://github.com/tensorflow/hub/blob/master/examples/image_retraining/retrain.py where it also specifies all possible inputs and outputs for the code. This enables custom placement of the outputs such as a graph, logs, and saved model, as well as changing parts of how the code executes (epochs, number of bottleneck files, which CNN to use, etc.). The retrain.py module found from the link above is technically deprecated in favor of newer modules that work on Tensorflow2. However, as previously mentioned, Tensorflow2 currently has issues with remote installation.

Transfer Learning

A typical end-to-end machine learning project will develop a model from the ground up. This takes a large amount of time and technical expertise, as how the data is inputted, split, convoluted, and tested all have to be developed manually. Then features, distinct characteristics that can be used to define each category, have to be identified by the model and given specific values in accordance with how those features match each category. Transfer learning seeks to streamlines this process by taking a pretrained model and applying it to a new dataset. At its essence, transfer learning is “a machine learning technique where a model trained on one task is re-purposed on a second related task” (Brownlee par. 6). Transfer learning has been shown to increase the “success rate of training from scratch to 75% was increased to 98% with transfer learning” (Şeker par.1) when applied to a CNN.

These pretrained models are developed as general image classifiers. They already come with a set of features that can be used to identify images, and vary based on speed, accuracy, and size of dataset. For example, one of the pretrained models we are using, the Inception V3 model, is “highly accurate, but comparatively large and slow” (retrain.py). The other model we are using, the MobileNetV1, is specifically developed to “maximize accuracy while being mindful of restricted resources for an on-device or embedded application” (Google AI Blog). MobileNets have been deployed in classification problems with high success rates: Islam et al. claim to have “put forth a MobileNet model which gives an amazing accuracy of up to 100%” when identifying local birds in Bangladesh, while Rabano et al. developed a MobileNet to classify common trash into different categories, with a final accuracy of 89.34% on their optimized model. Thus, we can see that not only have MobileNets been developed for our resource-limited problem, but have also experienced high degrees of success when applied to classification problems.

Transfer learning takes these predeveloped features and attempts to apply them to a new dataset. The same features that could be used to identify flowers such as certain colors, shapes, or sizes, can also be used to identify large mammals. Our brains work the same way as these models. When identifying a stool, we first notice the seat and the legs. Then, to distinguish the stool from a chair, we look at how long those legs are. However, a seat can apply to a whole range of objects: recliners, couches, car seat, etc. Similarly, a pretrained model has features that apply to some dataset, but these features can also be applied to entirely new problems. The transfer learning process involves changing the weights of those features, so that they can be used on the new datasets. The total architecture of CNN's is explained later in *Convolutional Neural Networks*, but transfer learning alters the last layer of a CNN. This layer is referred to as the Softmax layer when in a multi-class classification problem, and is the layer that actually assigns a label to the input image (dshahid380).

Convolutional Neural Networks (CNNs)

CNN image classifiers take in images as inputs, perform mathematical algorithms on them, and then classify the image as a category. For this project, the categories are defined by our image directories. CNNs will analyze both individual pixels and groups of pixels, looking for defining features. When using transfer learning, predetermined features are used, and their respective weighting and influence is changed to match the new dataset.

A Neural Network is composed of many layers. Each one of these layers performs mathematical operations, based on what the previous layers passed in. The layers are comprised of a series of nodes, each of which is given an input, performs calculations, and then feeds an output to the next layer, which can be seen in figure 5. The first layer, the convolutional layer, extracts features from the image. It does this by convoluting the images, looking for edges, sharpening, and blurring the image. Convoluting the images involves taking regions of pixels and averaging their values. Doing this simplifies the image by reducing the total number of pixels in the image. This helps the model extract features from the image through these computations. The values gained from the presence or absence of these features is then passed into a pooling field, where all the varying weights from each of the nodes from the convolutional layer are accumulated. Then, these pooled values get pushed back through another convolutional layer. This process repeats multiple times, until the outputs get flattened and fed into a series of fully connected layers that, in turn, feed values to the output layer, which classifies the image in question as one of our possible categories. This can be visualized in figure 6. The entire first half of this process, the convolutional layers and pooling, is referred to as Feature Learning. The latter half: flattening, fully connected layer, and image classification, is referred to as Classification.

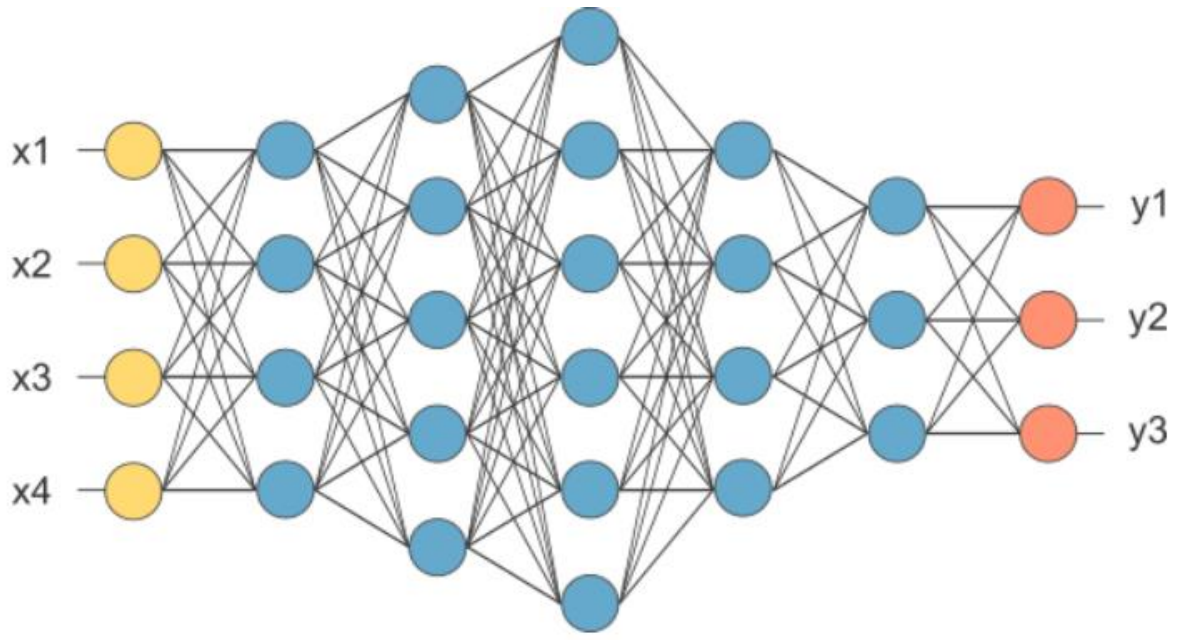

Figure 5: Example of node-structure of a CNN, where the Red Layer is the Softmax Layer (Prabhu)

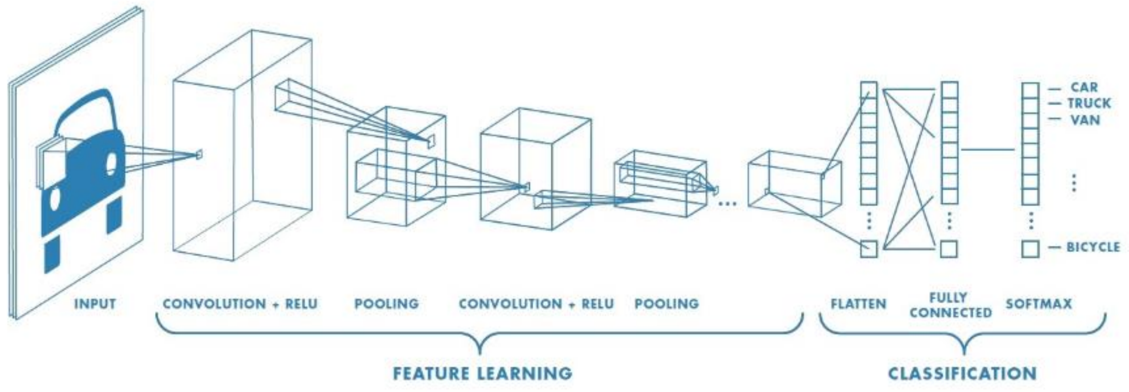

Figure 6: Visualization of layers of a CNN (Prabhu)

The process we are deploying is defined as Supervised Learning. This means that our labels are already known, so that the model can verify whether its predictions are correct or not. We will be dividing the data into three groups: the training set, testing set, and validation set. The majority of the training/retraining is done on the training set, while the validation set is used to sanity-check the model while it is being trained. The testing set is used for a final check on how well the model performs. Each one of these sets is kept separate, so no leaking of information is allowed. Additionally, the model runs through the data in epochs, splitting it into pieces so that the model can be incrementally sanity-checked and trained properly. Otherwise, it could make incorrect predictions on the whole dataset at once and be useless on the testing set. Typically, the data is split in an 8:1:1 fashion by the different sets. The accuracy of our retrained model will similarly be split by training, testing, and validation accuracies. In order to determine the overall accuracy of the model, I will calculate what I am going to call a Weighted Average. Here, since the training set possesses 8 times as much data as the validation and testing Sets, its accuracy will receive 8 times as much weight. For example, with an accuracy set of 80%:76%:82%, I would calculate my Weighted Average as:

$[0.8*(8) + (0.76) + (0.82)] / 10 = 79.8\%$

The Cross-Entropy Loss Function is also important in evaluating the effectiveness of a CNN. The way an image is labelled involves taking the highest probability calculated from the different categories (Uniqtech). For example, if our Dog category has a probability of .92, our CNN is 92% confident that the image passed in is a Dog. The Cross-Entropy Loss Function measures the divergence between our predicted probabilities and actual labels. Higher cross-entropy values indicate that the model is confident but wrong, indicating that it is apply its features incorrectly and misidentifying images. Lower values indicate that the model is not confident, showing that the model cannot differentiate between the categories well. Values closer to 0.5 indicate that the model is confident and correct, which is ideal (“A Gentle Introduction to Cross-Entropy for Machine Learning”).

Another piece of training the CNN is the distortion of the images. In order to gain more data and value from the images we already have, we will perform different operations on those images. These operations include rotating the image, flipping it, cropping it randomly etc. This allows features to be reinforced further by providing more images to analyze from a limited dataset.

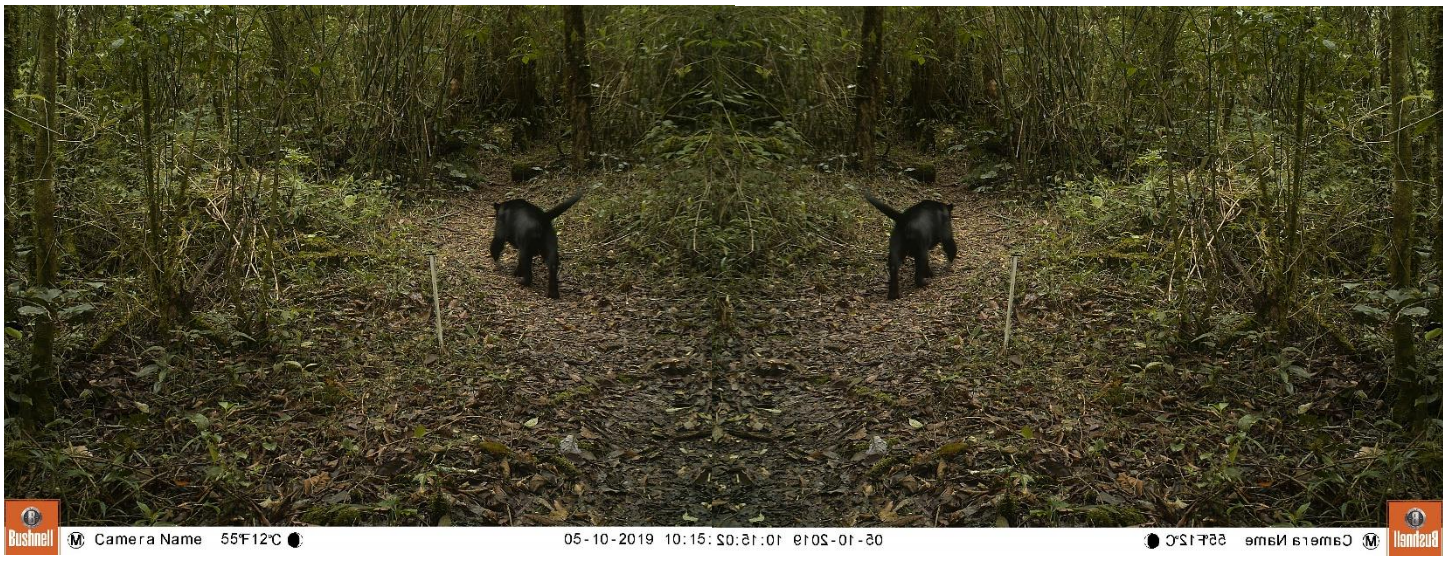

Figure 7: An image of a Jaguar flipped on the vertical axis, resulting in 2 images to analyze

Results

The Raspberry Pi was successfully deployed as an edge device. The installation of Raspbian and Debian was done successfully. The Coral Edge TPU was then successfully installed and tested, using guides from the Coral documentation. A remote camera was also integrated into the Raspberry Pi and used to capture a few images as a proof of concept (Fig. 8). The entire setup can be seen in Fig. 9.

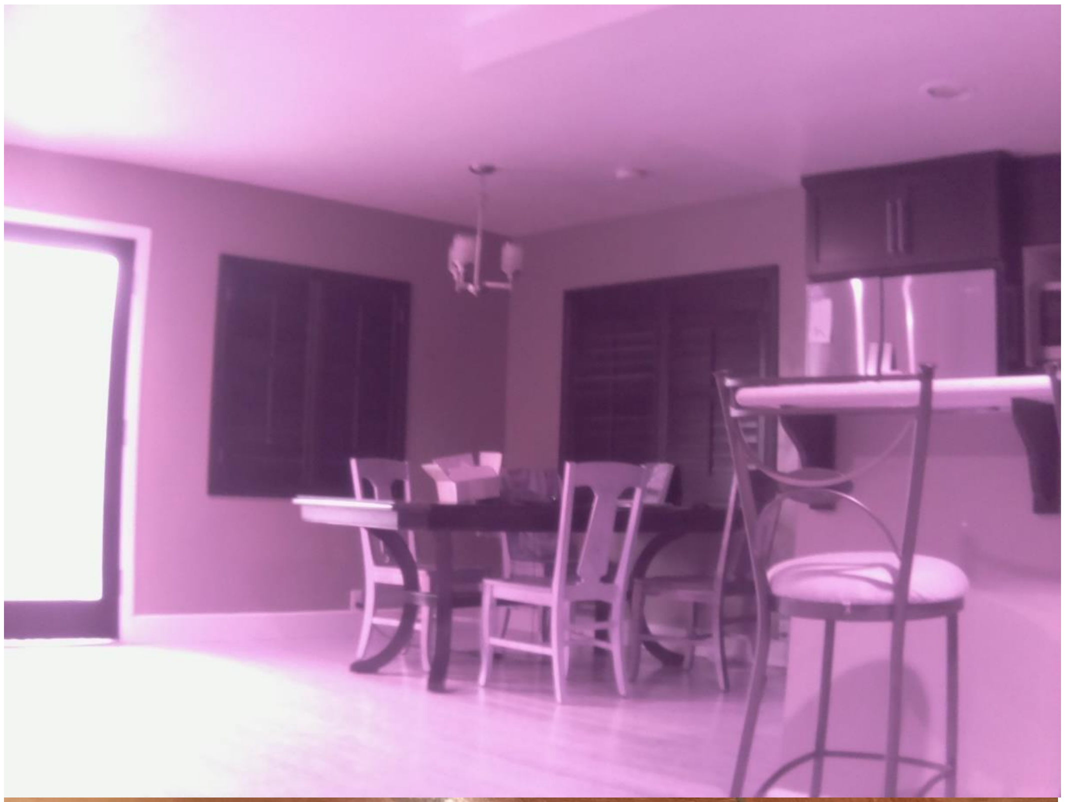

Figure 8: Image Captured by Raspberry Pi Module of my living room in low light

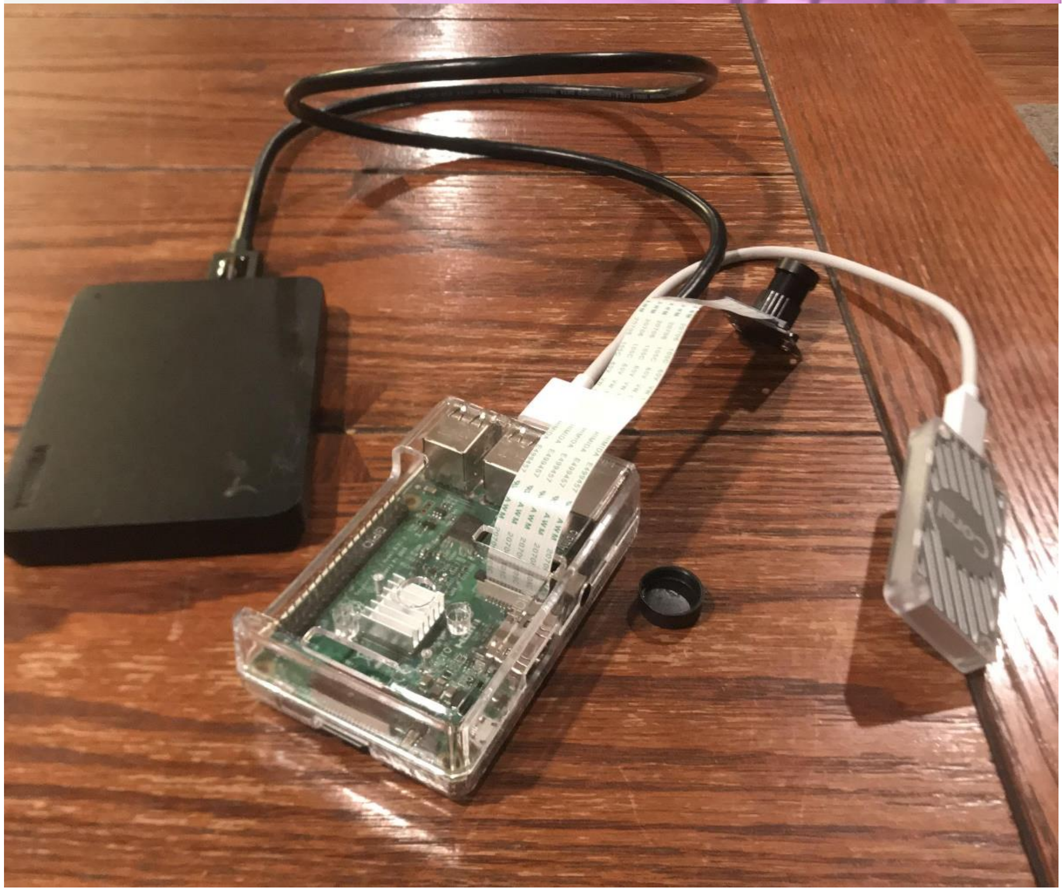

Figure 9: Hard drive on the left, Raspberry Pi with Camera in the middle, TPU on the right

After the initial setup of the edge device, transfer learning and a CNN were both successfully implemented on the same remote device. Downloading Python code directly from the TensorFlow Hub libraries on GitHub, saving them to the Raspberry Pi's hard drive, and executing from the command line in Raspbian was done successfully. Tensorflow1.14.0 was also installed on the Raspberry Pi. Multiple different CNN models were downloaded off of the TensorFlow Hub as well, with an Inception V3 and MobileNetV1 being used in the transfer learning process. The InceptionV3 model was used because the retraining python code we used had that model as its default. It is also a slower, more powerful model, providing an alternative to the faster MobileNetV1 model. The MobileNetV1 model was used due to its design being specifically for low-powered, resource restricted devices. It is of note that all of the transfer learning was done on the low-power remote device.

The retraining code used applies a cross-entropy loss function (defined in Methodology), automatically convolutes and distorts the images, and alters our Softmax layer. The MobileNetV1 model was deployed with 100 neuron layers and an image size of 224. Smaller image sizes result in less detail, but faster rates. Similarly, less neuron layers result in faster retraining rates, but lower accuracy. We deployed the maximum values taken by the python code for both of these values: 224 for image size and 100 for the number of layers.

These two models were successfully and efficiently retrained on our edge device. The defaults were kept from the retrain.py code from GitHub, using a batch size of 100, as well as a validation percentage of 10 and a testing percentage of 10. Because of this, the accuracy measurements are split into three categories: training accuracy (accuracy on the training set), validation accuracy (accuracy on the validation set), and testing accuracy (accuracy on the testing set). The MobileNetV1 model was able to retrain in ~6 hours, with a final test accuracy of 71.7%. Additional metrics from some of the last epochs include: training accuracy of 86%, validation accuracy of 74%, and a cross-entropy of 0.639. This results in our weighted average—where the training accuracy is considered to hold 8 times as much weight as the other two accuracies—of 83.4%. The Inception V3 model is considered slower and more accurate, and retrained in ~18 hours. Surprisingly, it actually had a lower final test accuracy of 70.8. Additionally, it had a final training accuracy of 78%, validation accuracy of 65%, and cross-entropy of 0.725. With a weighted average of 75.9%, this means that the faster and less power-intensive MobileNetV1 model actually performed better than the more-powerful, slower InceptionV3 model. The cross-entropy of 0.639 also points towards the MobileNetV1 being more confident in its predictions than the InceptionV3 counterpart. This proves that transfer learning can be done efficiently and effectively on a low-power edge device, with assistance from an Edge TPU.

Issues:

One of the largest problems came from the use of deprecated modules. Unfortunately, the new modules proved unreliable in their documentation and did not perform the way they claimed to. This has to do with what was previously mentioned in the *Software* section of the Methodology: Tensorflow2.0 cannot be downloaded remotely onto an edge device and many modules do not function properly. Additionally, at the 2020 New Year, TensorFlow displaced the older modules in favor of their newer ones. Previously, the module we were using had a whole tutorial/guide page, explaining how the Python module worked and how to modify it for my own purposes. Because that page was removed from existence, we were forced to use the limited documentation present in the actual downloadable form of the module on GitHub.

These deprecated modules also prevented us from properly analyzing the retrained models. Certain pieces of the code have been completely replaced in the TensorFlow library, with no reliable alternatives found after hours of searching. Thus, a few graphs cannot be saved properly due to the absence of specific TensorFlow modules within the Python code. It is notable, however, that this has absolutely no impact on the actual performance of the transfer learning on the CNN, just an impact on what tools are available to analyze the retraining deeper.

Additionally, the data fed to the retraining module was not properly cleaned. We did not have the time nor expertise to sift through the thousands of images and remove the ones with misidentified species or without any species at all in them. We are confident that this had an impact on the performance of the transfer learning, and that if that data was cleaned in the future and the same processes that we have done were executed again, better and more accurate performance could be gotten out of these CNNs.

Another problem that arose was from overheating of the Raspberry Pi. Even with the heat sink installed, if we ran the retraining module multiple times in a row, without giving the Raspberry Pi a break by turning it off for a period of time, the Raspberry Pi would crash during the transfer learning sequence. In reality, this would not be a major problem, as you would ideally only retrain a model once, instead of manipulating the parameters over and over again in a short amount of time.

The last problem we encountered was the development of a confusion matrix. This would other researchers determine which species were commonly confused or misidentified by the CNNs, allowing for further customization/analysis. Unfortunately, the TensorFlow modules for creating and saving these confusion matrices could not be coaxed into working properly.

Conclusion/Discussion

The question that drove this project—Can we use transfer learning to implement a CNN to do large mammal classification on an edge device? —was answered. An edge device with machine learning capacities was deployed, a CNN deployed on that edge device was retrained on a new dataset using transfer learning, and the CNN was shown to be able to classify mammal images with below average confidence in comparison to other research projects. As previously mentioned under *Transfer Learning* in Methodology, Şeker managed to improve a model from 75% to 98% and Rabano et al. obtained a final test accuracy of 89.34% in their image classification with a MobileNet. With a final testing accuracy of approximately 70-71% for both of our models, our pretrained networks performed less than ideally, however given the presence of incorrect labels in our dataset, this may not be a model limitation, but due to the data itself.

There is definitely room for improvement, especially with the actual classification part of the algorithm. Cleaning up the data, as mentioned in the *Data* part of Methodology, would greatly improve performance and accuracy. As evidenced by accuracies obtained by other research projects, the limitations on our model's accuracy are not derived from the nature of the problem itself. Instead, these limitations are due to mislabelled data and nonfunctional modules. I believe that future researchers can produce a highly accurate CNN capable of tagging new images once the data is cleaned properly and modules are updated. Additionally, further analysis of the confusion matrix, output graphs, and TensorFlow Logs would allow the researchers to determine which species the model is confident at predicting. By providing researchers with this information, once the CNNs are actually deployed on new data, they could be adapted to tag any images from the less-confident species, prompting manual analyzation.

The limitations from both time and faulty data restricted the overall success of this project. However, although the final models may not have been as accurate as I would have liked, the main goals of this research were all satisfied: we successfully performed transfer learning on an edge device and retrained a CNN to classify large mammals in Costa Rica.

Bibliography

Afkham, Heydar Madoubi, et al. "Joint Visual Vocabulary for Animal Classification." *2008 19th International Conference on Pattern Recognition.* Tampa, FL. 2008, pp.1-4.

Brownlee, Jason. "A Gentle Introduction to Transfer Learning for Deep Learning." *Machine Learning Mastery*. 20 December 2017. Web. https://machinelearningmastery.com/transfer-learning-for-deep-learning/

---. "A Gentle Introduction to Cross-Entropy for Machine Learning." *Machine Learning Mastery.* 21 October 2019. Web. https://machinelearningmastery.com/cross-entropy-for-machine-learning/

Cao, Zheng, et al. "Marine Animal Classification Using combined CNN and Hand-Designed Image Features." *OCEANS 2015—MTS/IEEE Washington.* Washington, DC. 2015, pp. 1-6. https://ieeexplore.ieee.org/abstract/document/7404375

Cass, Stephen. "Taking AI to the Edge: Google's TPU now comes in a Maker-Friendly Package." *IEEE Spectrum*, vol. 56, no. 5, 2019, pp. 16-17. https://ieeexplore.ieee.org/abstract/document/8701189

Dshahid380. *Convolutional Neural Network*. TowardsDataScience. 24 Feb. 2019. Web. https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Islam, Md. Romyull, et al. "MobileNet Model for Classifying Local Birds of Bangladesh from Image Content Using Convolutional Neural Network." *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT),* Kanpur, India, 2019, pp. 1-4. https://ieeexplore.ieee.org/document/8944403

Lawrence, S, et al. "Face Recognition: A Convolutional Neural Network Approach." *IEEE Transactions on Neural Networks,* vol. 8, no.1, 1997, pp. 98-113. https://ieeexplore.ieee.org/abstract/document/554195

Long, Changchun, et al. "Edge Computing Framework for Cooperative Video Processing in Multimedia IoT Systems." *IEEE Transactions on Multimedia*, vol. 20, no. 5, 2018, pp. 1126-1139. https://ieeexplore.ieee.org/abstract/document/8070982

"Machine Learning: What it is and Why it Matters." *SAS Insights.* SAS Institute. Web. https://www.sas.com/en_us/insights/analytics/machine-learning.html

MobileNets: Open-Source Models for Efficient On-Device Vision. *Google AI Blog.* https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

Prabhu. *Understanding of Convolutional Neural Network (CNN) – Deep Learning.* Medium. 4 Mar. 2018. Web. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Rabano, Stephenn L., et al. "Common Garbage Classification Using MobileNet." *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM).* Baguio City, Philippines, 2018, pp. 1-4.

Şeker, Abdulkadir. "Evaluation of Fabric Defect Detection Based on Transfer Learning with Pre-trained AlexNet." *2018 International Conference on Artificial Intelligence and Data Processing (IDAP),* Malatya, Turkey, 2018, pp. 1-4. https://ieeexplore.ieee.org/document/8620888

Shao, Ling, et al. "Transfer Learning for Visual Categorization: A Survey." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 5, 2015, pp. 1019-1034. https://ieeexplore.ieee.org/document/6847217

Ujjwalkarn. *An Intuitive Explanation of Convolutional Neural Networks.* The Data Science Blog. 11 Aug. 2016. Web. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Uniqtech. *Understand the Softmax Function in Minutes.* Data Science Bootcamp. 30 Jan 2018.

Web. https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

Vilros Basic Starter Kit: https://vilros.com/products/raspberry-pi-3-model-b-basic-kit

Wang, Xiaokang, et al. “A Cloud-Edge Computing Framework for Cyber-Physical-Social Services.” *IEEE Communications Magazine*, vol. 55, no. 11, 2017, pp. 80-85. https://ieeexplore.ieee.org/abstract/document/8114554