AUTONOMOUS ANOMALY DETECTION AND CLASSIFICATION USING NEURAL NETWORKS

BY SIDDARTH IJJU

MENTORED BY DR. KEITH HARRISON, CHERRY CREEK HIGH SCHOOL

Abstract

The purpose of this project is to create a novel implementation of the concepts of anomaly detection and object classification to assist search-and-rescue (SAR) teams in rescue missions. SAR teams continually suffer from a lack of resources and volunteers, which decrease their ability to efficiently search the hundreds of miles of trails in the potential search area. My project aims to address this issue by creating an autonomous system that would be able to detect objects and anomalies on the trail using object classification and anomaly detection. This system would be used in conjunction with a quadrotor or other autonomous vehicle and would enormously increase efficiency in search missions. The research done in this project can also be applied in other areas such as cybersecurity and medical imaging.

Anomaly detection is the ability of a machine to learn what is normal for a system, and then detect outliers from the norm for that system in the future. Object classification is the ability of a machine to learn features differentiating between objects and then use this information to classify objects in a new image. In this project, a 2-layer convolutional auto-encoder was constructed and tested against a set of images with an accuracy of 87.5% for detecting anomalies. In conjunction, a pre-existing object classifier called MobileNet was re-trained to classify common hiker items with an accuracy of 93%. An autonomous implementation of the model was also created. The contributions of this project include performing real time object classification and anomaly detection that do not require Wi-Fi or cellular service.

Table of Contents

Introduction

Background

The advent of the 20th century brought humans their first taste of robotics and Artificial Intelligence. One of the major concepts of Artificial Intelligence (AI) concerns anomaly detection, or the ability of a machine to learn features that define what is normal for a system, and then detect outliers from the norm for that system in the future. Some examples in our daily lives include “military surveillance for enemy activities, intrusion detection in cyber security, fraud detection for credit cards, insurance or health care and fault detection in safety critical systems.” (Singh and Upadhyaya, 2012). Another major concept in AI is that of object detection and classification: the ability of a machine to learn about different objects from extensive training data (in image format) and use this information to classify objects in a new image. Object detection applications “have expanded into video surveillance, medical image analysis, facial detection and recognition, and many others.” (Hirano et al.).

The goal of my project was to combine these two major concepts of machine learning in a novel implementation that could potentially provide enormous real-world benefits. This combined implementation would theoretically take an image as input, identify it as anomalous or not, and then classify the objects within the image for further analysis.

A sample application of this problem, and the main inspiration for this project, came from the issues search-and-rescue (SAR) teams have in their missions. Typical SAR teams have about 6 to 8 volunteers that will engage in an initial search for the subject. This phase is where there is a need for more volunteers and speed because the likelihood of finding a subject goes down with every hour gone by from the start of a search. In many cases, the terrain is difficult for human volunteers to conduct a quick search, and requires additional skills and time. A model that could allow a quadcopter or other unmanned aerial vehicle to quickly search for and detect objects near and on trails would provide a huge time benefit to these teams as they could then allocate human resources to more difficult to search terrains. However, the applications of this implementation are not limited to the above.

Neural Networks

In recent years, Convolutional Neural Networks (CNN) have enjoyed great popularity as a means for image classification/categorization since Krizhevsky et al. (2012) won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012 competition. Their work has spurred the use of neural networks for anomaly detection and object classification in one functioning model.

The general idea of a neural network is to model the process in which the human brain itself learns to recognize objects and concepts, albeit in a more primitive and mathematically oriented way. Neural networks often consist of hundreds, thousands, or even millions of nodes, each of which performs a simple mathematical function, in crude approximation to neurons in the brain. These nodes are often arranged into layers to perform their function based on input from nodes from the previous layer (Figure 1).

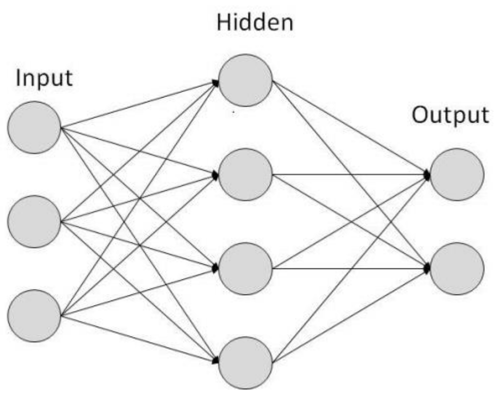


Figure 1. A simple example of a neural network. The various nodes are represented by circles and are organized by layer, with the first layer containing 3 nodes labeled as the Input layer and so on. The connections between nodes represent where the results of one node go to another as input.

A neural network takes as its input an n-dimensional vector x where n represents the number of input features in the data. For example, if we were creating a neural network to classify cats vs. humans, the features in x could be the number of legs, number of ears, whether the input has fur or not, etc., and there would be n such features.

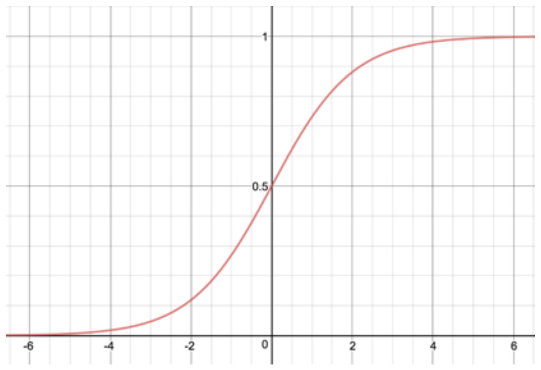
Each node in a neural network begins with a weight w and a constant bias value b, which is usually either 0 or a random number,

$$\mathrm{z} = \mathrm{w} * \mathrm{x} + \mathrm{b}$$

where z is shorthand for the output of the specific node.

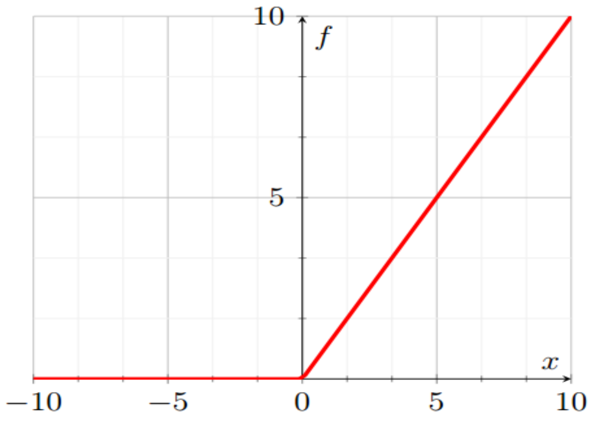
As training of the neural network commences, each node receives the result of the previous layer in the network. To calculate node outputs, the node uses a defined activation function (a term borrowed from the "activation" of neurons) to calculate its output. Commonly used functions include the sigmoid function:

$$\mathrm{sigmoid}(\mathrm{z}) = \frac{\mathrm{e}^{\mathrm{z}}}{\mathrm{e}^{\mathrm{z}} + 1}$$



Or the Rectified Linear Unit (ReLU) equation:

$$\text{relu(z)} = \max(\text{z}, 0)$$



This process is repeated for each node in a neural network until the output vector y from the final layer is produced. This process can be made easier by treating the nodes and their parameters as vectors that contain all the values for that layer. Using this approach, the calculations for a layer can be reduced dramatically. Continuing, we will use the notation W_l to represent the vectorization of the w values for all the nodes in the lth layer of the neural network, and similarly B_l for b and Z_l for z.

We now delve into the math required for the actual learning done by a neural network. To do this, we must define two concepts: gradient descent and loss functions. Loss functions are the functions that compute the loss of an iteration of a neural network. In other words, it computes how accurate the current neural network by calculating the difference between the model's set of predictions and the actual answers. Gradient descent is the algorithm used to attempt to minimize the cost function, which is the combined loss functions of the neural network over all training examples (Figure 2).

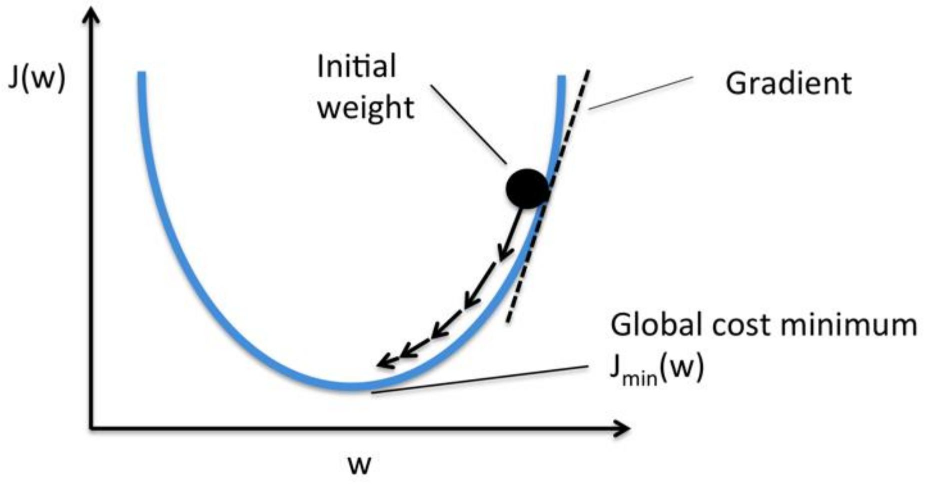


Figure 2. A visual representation of gradient descent. The variable w in the image corresponds to the variable z in this paper. The algorithm tries to find the global cost minimum through hundreds of trials from the initial loss (referred to as weight in image). The algorithm uses the gradient function (like the mathematical derivative) to find the global minimum.

Minimizing the cost function has an effect of making the neural network more accurate (i.e. able to correctly identify cats vs. humans at a higher percentage).

A loss function can be expressed as:

$$\mathrm{L}(A_l, \mathrm{y}) = -(\mathrm{y} * \log(A_l) + (1 - \mathrm{y}) * \log(1 - \mathrm{A_l}))$$

where A_l is the vectorization of the outputs of the lth layer of a neural network made on a single training example, and y is the correct classification the network should have predicted for that example.

This loss function is then used in the cost function, defined as:

$$\mathrm{J(W,B)} = \frac{\sum_{\mathrm{i=1}}^{\mathrm{m}} \mathrm{L}\left(A_{last}\ , \mathrm{y^i}\right)}{\mathrm{m}}$$

where m is the number of training examples, A_{last} is the vectorization of the output of the last layer of the neural network (the final prediction), and the superscripted y refers to the prediction and label for that training example.

Now we can define the gradient descent algorithm as:

Repeat for every layer l:

$$\mathrm{W_l = W_l - \alpha \left(\frac{dJ(W,b)}{dW}\right)}$$

$$\mathrm{B_l = B_l - \alpha \left(\frac{dJ(W,b)}{db}\right)}$$

Where a is the learning rate (how quickly the neural network changes what it has learned with each training example), and $\frac{\mathrm{dJ(W,b)}}{\mathrm{dW}}$ and $\frac{\mathrm{dJ(W,b)}}{\mathrm{db}}$ are the derivatives of the cost function with respect to the weight and bias values, respectively.

Now we must introduce the backpropagation algorithm. The backpropagation algorithm is the algorithm that is used to compute the $\frac{\mathrm{dJ(W,b)}}{\mathrm{dW}}$ and $\frac{\mathrm{dJ(W,b)}}{\mathrm{db}}$ terms in the above gradient descent algorithm. In simpler terms, the backpropagation algorithm computes the gradient of the cost function to update the parameters for the nodes in each layer.

The backpropagation algorithm treads backwards from the last layer of the neural network to the first to produce the derivatives desired. The algorithm can then be defined as follows:

The Back-Propagation algorithm can be defined as below:

For every layer in the neural network, from the last to the first, we have:

$dZ_l = A_l - Y$ (where dZ_l is the derivative of the contents of the layer l's nodes)

$dW_l = \frac{dZ_l * A_{l-1}}{m}$ (where $dW_l$is the derivative of the W_l vector in layer l)

$dB_l = \frac{\sum_{i=0}^{n} dZ_l}{m}$ (where $dB_l$is the derivative of the B_l vector in layer l)

$dA_{l-1} = W_l * dZ_l$

Deep Auto-encoders

An important part of my project is the deep auto-encoder, a neural network whose hidden layers are progressively smaller than the input and output layers (Figure 3).

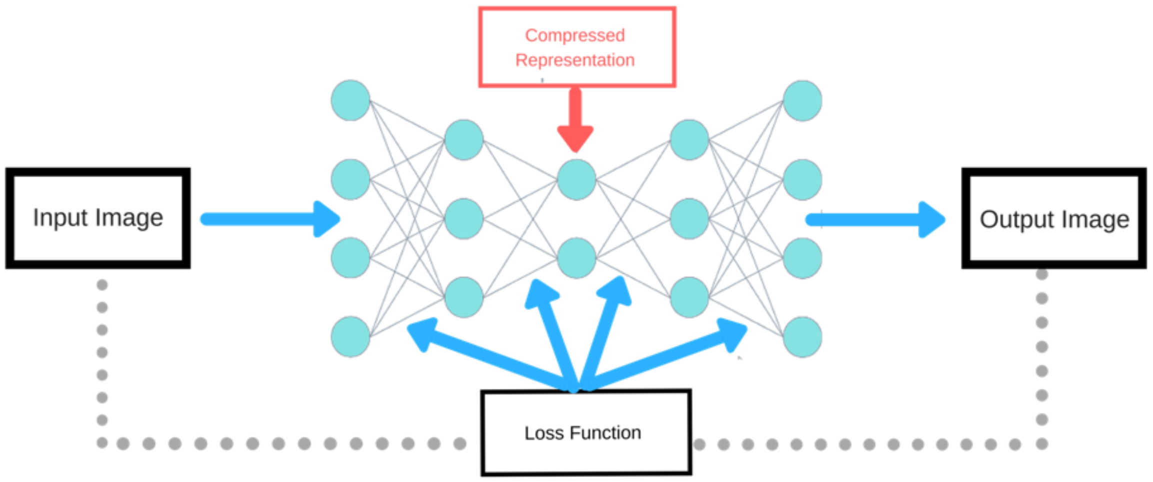
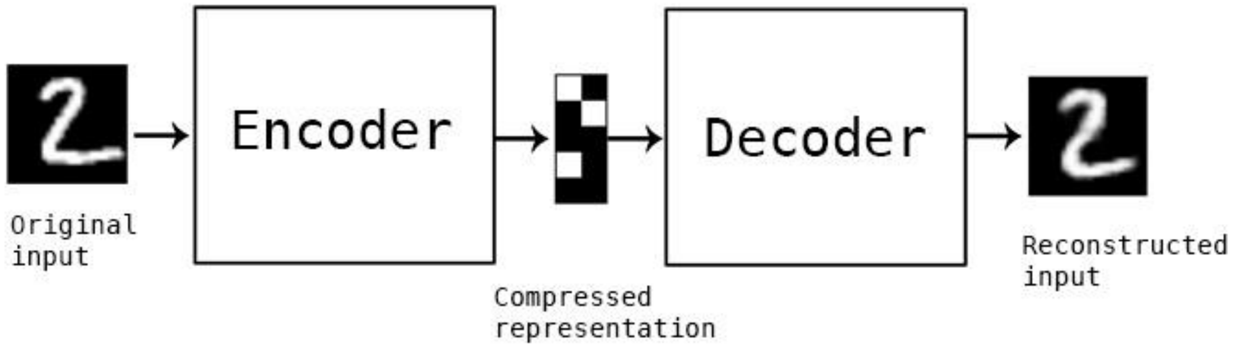


Figure 3. Deep Auto-encoder. The figure shows the general layout of the nodes and layers of a conventional deep auto-encoder

As hinted in Figure 3, auto-encoders are primarily used to compress data. The function of the auto-encoder is to learn how to compress an input image then decode the compressed image back into the input image. An example of this can be seen in the below image, taken from Chollet (2016).



The reason an auto-encoder is vital to the function of my project is twofold. First, using an auto-encoder is necessary for the storage of image data on a microprocessor for a quadcopter due to the small storage space available. Second, auto-encoders are a relatively new method of detecting anomalies in images, which is a main component of my project. Since using an auto-encoder fulfills two objectives of my project, it was the clear choice.

The loss function used for auto-encoders is significantly different from the loss function previously mentioned. Since an auto-encoder deals with image data, the loss function must be able to approximate the amount of difference between an auto-encoder picture prediction and the actual image pixel-by-pixel.

Initially, the Binary Cross Entropy (BCE) cost function was used. BCE can be expressed as

$$\text{BCE} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

where $\hat{y}$ is the prediction for a given image (into two categories, hence the binary modifier) and y is the correct result. However, this function did not represent the goals of an auto-encoder very well, due to its innate discreteness and inability to accurately measure differences between images.

Therefore, another cost function was used for this project, which was the Mean Squared Error (MSE) function. This function was chosen for its simplicity and accuracy in other auto-encoder experiments.

MSE can be expressed as,

$$\mathrm{MSE} = \frac{1}{n} * \sum_{i=1}^{n} \left(\hat{y}^{i} - y^{i}\right)^{2}$$

where $\hat{y}^{i}$ is the prediction given by the auto-encoder for the i^{th} training image, y^{i} is the original value of the i^{th} training image, and n is the number of training examples.

Solution

The primary idea for my solution begins with building and training a deep auto-encoder on images of hiking trails I shot in the field. Then, I would retrain an existing object classification network (Google's MobileNet) based on the original images I collected. Next, I would compile both models into one functioning "super" model. Finally, I would implement the model on a Raspberry Pi Zero as part of a Google AIY Vision Kit.

The benefits of an approach like this are significant. First, the model would not require any Wi-Fi or cellular service to run, meaning it would be completely autonomous and self-sufficient. Secondly, this model allows for a wide range of uses, as the model can simply be retrained for different scenarios and applications like the original.

Research Question

Can a system be developed to autonomously detect and classify anomalies using neural networks? This system would be able to both detect anomalies on a trail and classify the anomaly on the trail. For the anomaly detection aspect of the system, an auto-encoder will be trained on image data collected of non-anomalous trails. For the classification aspect, a MobileNet model will be retrained to classify anomalies in four categories (water bottles, hats, backpacks, and shoes).

Goals and Parameters

The parameters I will be using to judge the success of my project are:

1. Detection of anomaly from image with an accuracy > 70%
2. Detection of objects in anomalous image with an accuracy > 80%

Methods

The materials I used for this project were; a Nikon camera for taking pictures of trails, a Raspberry Pi Camera V2 for the autonomous implementation of my models, a Raspberry Pi Zero W and Google AIY Vision Kit for the actual physical implementations of my models, and a computer with Python capabilities for creating my models.

The first step I did was to gather my training dataset for my models. This was done by first taking non-anomalous pictures of trails from the field, making sure to center the trail in the image and to have a consistent angle which the image is taken from. I then took anomalous pictures from the field, each containing objects in some category of anomaly (boots, backpacks, trail mixes, etc.). I then sorted and labeled the dataset for anomalous and non-anomalous categories, setting aside 70% of the non-anomalous images for training and using the anomalous images and the remaining 30% of the non-anomalous images for testing.

I then created and trained my auto-encoder by using the training images to build and train a decoder and encoder model as part of an auto-encoder. As part of this process I had to calculate MSE for each epoch of training as well as use Numpy's polyfit module to fit a line to the SSIM error from the trained and loaded auto-encoder.

I then created an error analysis program to run through bias vector values to determine best line of fit for auto-encoder accuracy. After the anomaly detection model is trained and tested, I performed object detection on the image to identify the anomaly in it. This process was done by retraining final layers of MobileNet model using the anomalous images as categories.

After the models were completed, I then compiled both the object classification script and the anomaly detection script into one Python script for the implementation on the Raspberry Pi. I then set up and created a Raspberry Pi environment and constructed the Google AIY Vision Kit that will run the model. The final steps of my project were to import and implement my models on the Raspberry Pi using the script compiled previously and test my prediction model using the Google Vision's Camera.

Results and Discussion

Stage 1

As mentioned in the Results section, Stage 1 primarily revolved around the construction of the auto-encoder. However, in this section I will elaborate more on data presented in the Results section, what it means, and the process of achieving the data shown.

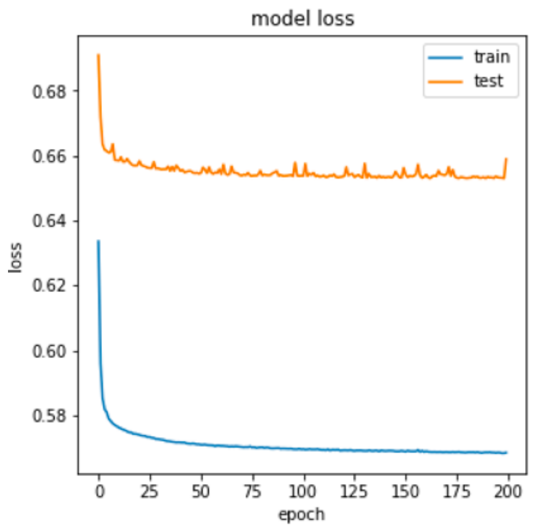


Figure 4: Value of Loss Function Over Epoch for Trial #1

Figure 4 shows the progression of the loss function over the training of the auto-encoder during Trial #1. The gradual flattening of the training curve shows how the auto-encoder gradually learned the correct weights and biases to minimize the loss function given to it (originally binary cross-entropy). The reason this model only achieved an accuracy of 22% was because of a shortage of data. Only 200 pieces of data were passed through for the training of this model, and that likely wasn't enough data to allow the model to sufficiently learn patterns in non-anomalous image.

This reasoning can be seen below in Figure 5. The image and its decoded representation are wildly different and inaccurate because the auto-encoder did not have enough data to learn any meaningful patterns (such as a straight, dull-colored section down the middle of the images).

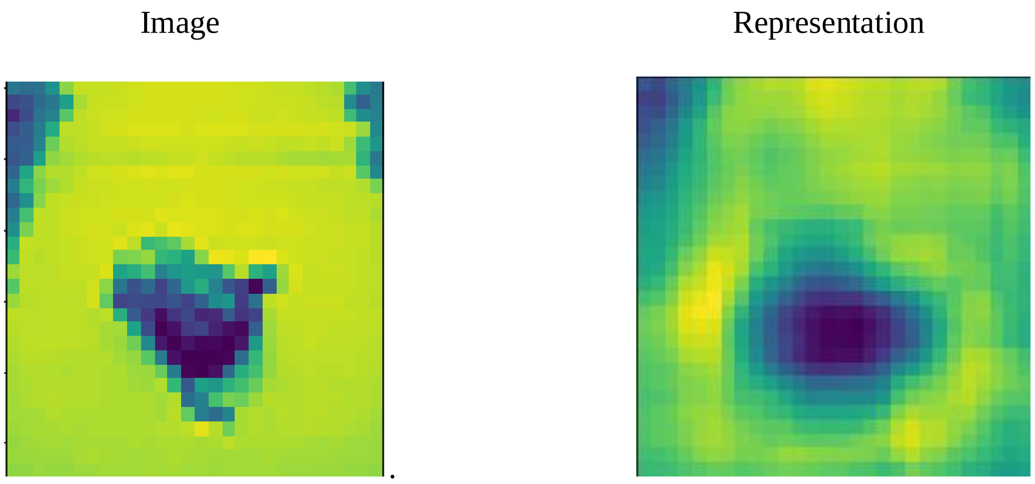


Figure 5: Image and AE Representation for Trial #1

Figure 6 describes the loss of the second version of the auto-encoder. Between Trial #1 and Trial #2, more data (roughly 100 extra training images) were added to the training dataset to determine whether it was the amount of data that affected the accuracy.

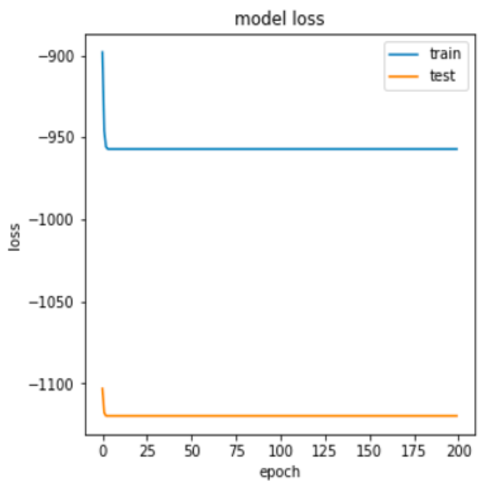


Figure 6: Value of Loss Function Over Epoch for Trial #2

However, the accuracy of the second model only increased to 23%. This indicated that the model had other issues than the amount of data given to it, but still did perform better if given more data.

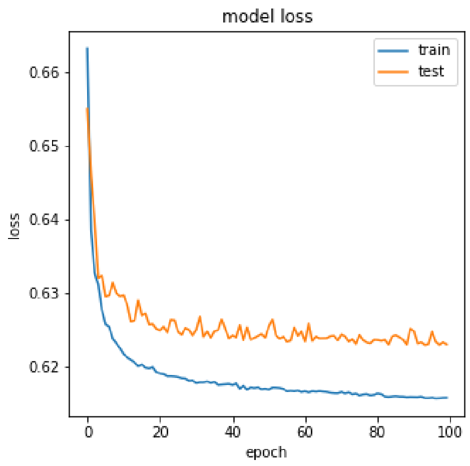
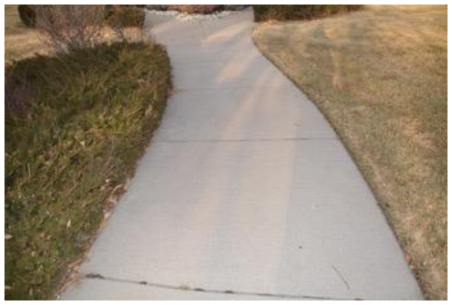


Figure 7: Value of Loss Function Over Epoch for Trial #3

Between Trials #2 and #3, two issues were identified with the model. The first was that the training dataset was not very specific to the problem (i.e. the auto-encoder was not able to identify a pattern with the images passed to it). The second issue was that the testing dataset was small and possibly error-prone (it initially only had 28 images). The training dataset was thereby fixed by adding new pictures and trimming out some that were not good for the model to learn from, as seen in Figure 8:

Example of good image:



Example of bad image:

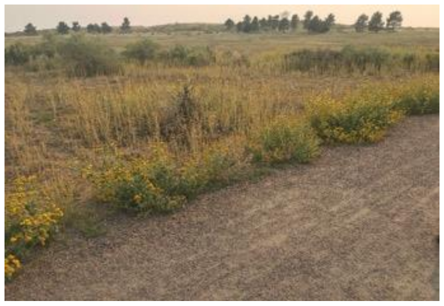


Figure 8: Examples of good and bad images for pathway detection

Figure 7 shows that the auto-encoder returned to a slower loss, and thereby slower learning, which was precisely the result desired from the changes made. Figure 7 also shows some of the inconsistencies in the loss of the testing dataset, validating the capabilities of the auto-encoder approach to the problem. This result can also be derived from Figure 9.

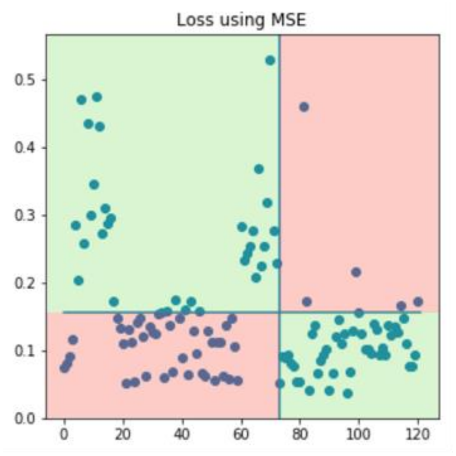


Figure 9: Distribution of MSE Loss over Testing Dataset

In Figure 9, all points to the right of the line are part of the non-anomalous testing dataset, and all points to the left are part of the anomalous testing dataset. As can be clearly seen, there are many more images above the average MSE (Mean Squared Error) line (the horizontal line) on the anomalous side than on the non-anomalous side. However, the changes made between Trials #2 and Trials #3 clearly worked to great effect, as the accuracy of the model in Trial #3 increased to 40%.

Between Trial #3 and Trial #4, the cost function used by the auto-encoder was changed from Binary Cross-Entropy to MSE. The reason for the shift was because MSE was a closer approximation of differences in images than Binary Cross-Entropy.

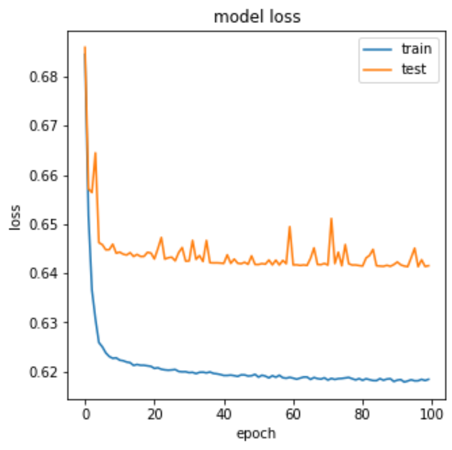


Figure 10: Value of Loss Function Over Epoch for Trial #4

In accordance with the prediction that using MSE instead of Binary Cross-Entropy would increase the accuracy, that accuracy of the model in Trial 4 increased to 51%. Figure 10 also shows that an increased sensitivity to anomalies was achieved in this trial, as evidenced by the variation of loss in the test dataset.

The next changes made to the auto-encoder were largely inspired from the Lyudchik (2016) research on outliers in the MNIST digits dataset. Lyudchik found that auto-encoder accuracy could be improved by decreasing the number of layers in the auto-encoder to prevent the auto-encoder from learning features it should not. In Lyudchik (2016), the researcher attempts to create an outlier detection system for the number 7. However, the auto-encoder was unable to identify 7 as an anomaly because it had learned the features of a 7 from the features of a 1. In the case of this experiment, the auto-encoder could have been learning ways to represent objects it had never seen before (backpacks, hats, shoes, etc.) from the areas in each image of grass and shrubbery.

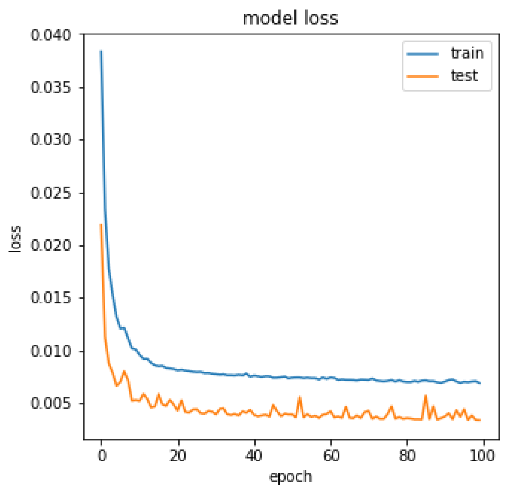


Figure 11: Value of Loss Function Over Epoch for Trial #5

Figure 11 shows that after changes were made to decrease the number of layers for the auto-encoder, the convergence for the loss on the training dataset took longer than in previous trials, and that the testing dataset had increased in anomaly sensitivity, indicating that the changes had indeed increased the accuracy of the model. The accuracy for the model in Trial #5 corroborated the predictions made, with an accuracy of 60%.

Between Trial #5 and Trial #6, over 200 more images were added to the training dataset, and approximately 150 more images were added to the testing dataset. In addition, the auto-encoder network was modified to allow color images to be passed through the network. Up until Trial #6, the model only had capability to take gray-scaled images as input.

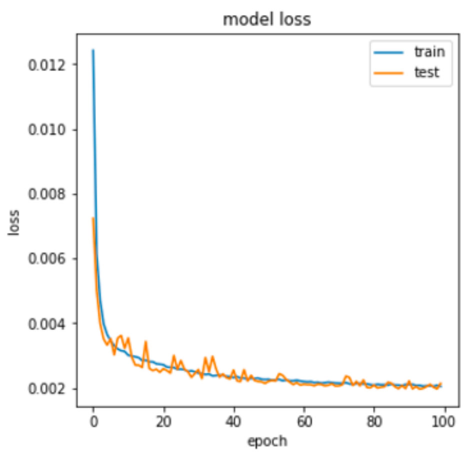


Figure 12: Value of Loss Function Over Epoch for Trial #6

As seen in Figure 12, this model converged in a much shorter timeframe (100 epochs vs. 200 epochs previously) than previous models. The testing dataset also lined up much closer to the training dataset than before, which indicates a higher accuracy and less overfitting in the auto-encoder. The results of this model corroborated the predictions and changes made, with an accuracy of 74%, surpassing the first goal mentioned in the Introduction to this paper.

However, this is not the final model. After adding more data to both the training and testing datasets, we arrive at the final model.

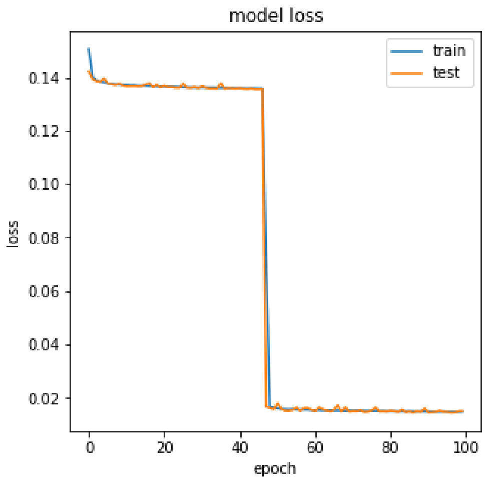


Figure 13: Value of Loss Function Over Epoch for Final Trial

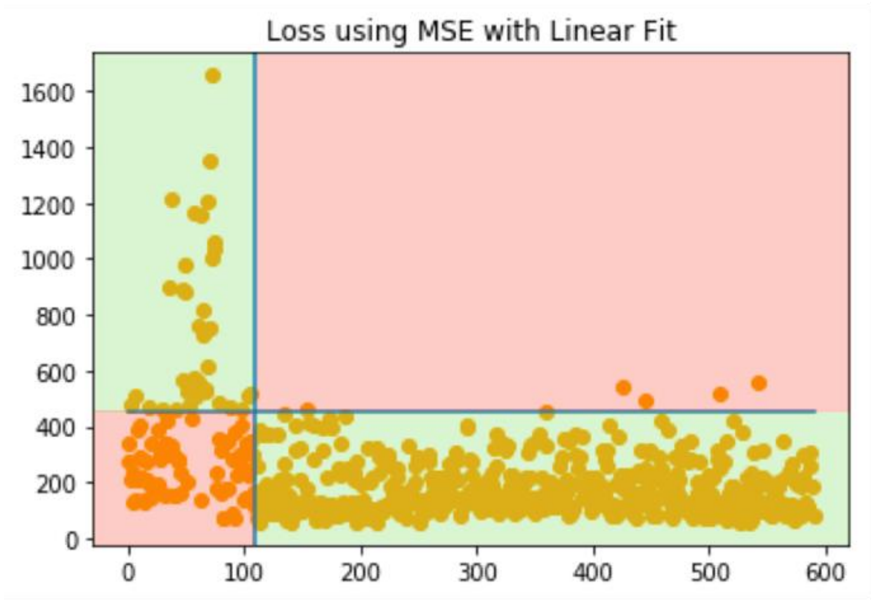
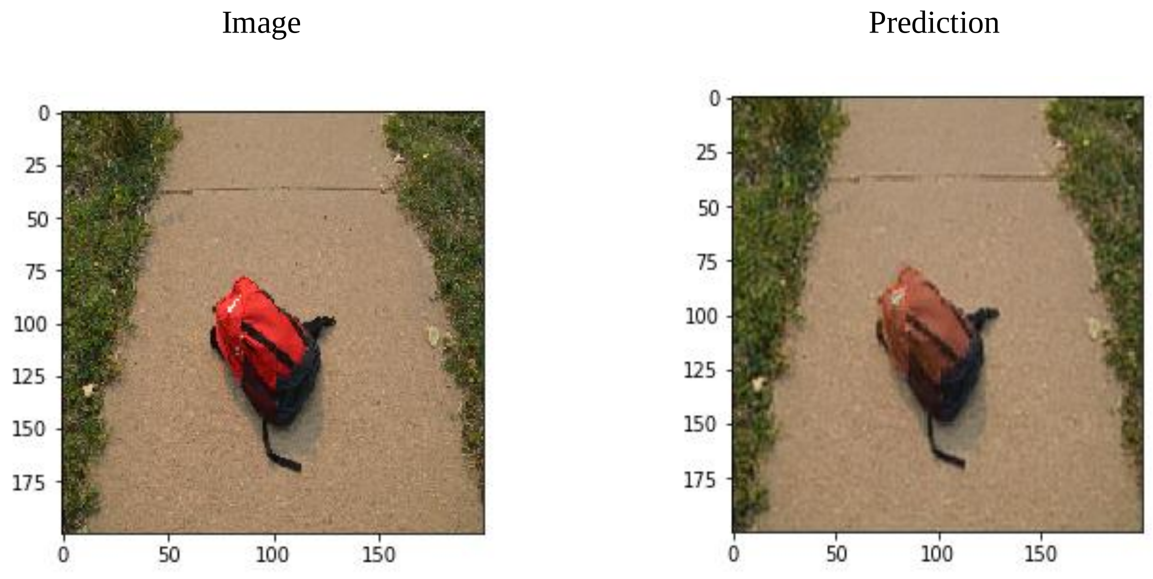


Figure 14: Distribution of MSE Loss over Testing Dataset

As can be seen in Figure 13, the testing and training dataset were closer in-line than ever before, belaying an increase in accuracy. Also, as seen in Figure 14, almost all the non-anomaly data points are under the line, whereas a significant portion of the anomalous points are above the line. The accuracy achieved from making these changes was 87.5%.

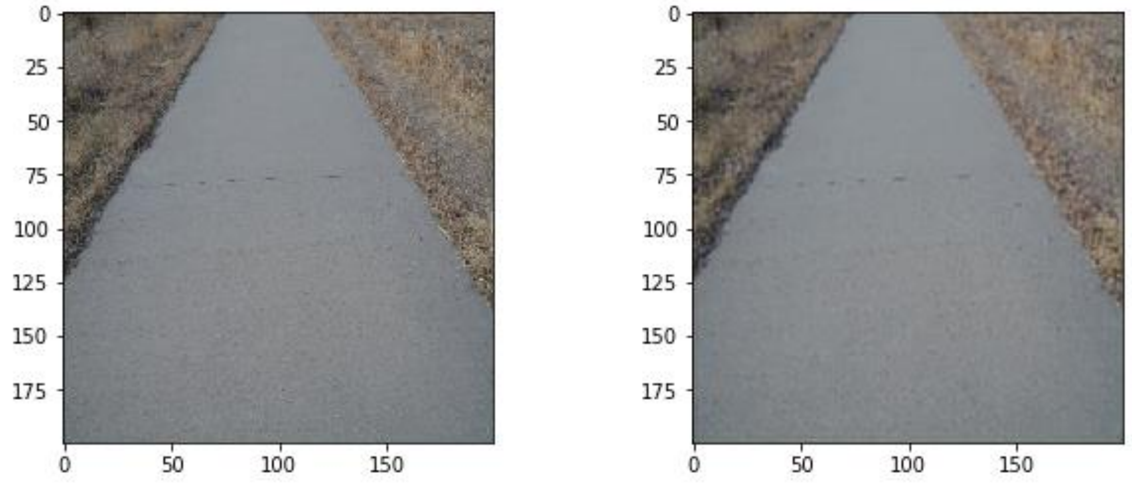
Figures 14 and 15 show sample classifications of anomalous and non-anomalous images.



Classification: Anomalous

Figure 14: Sample Classification of Anomalous Image

Image Prediction



Classification: Not Anomalous

Figure 15: Sample Classification of Non-Anomalous Image

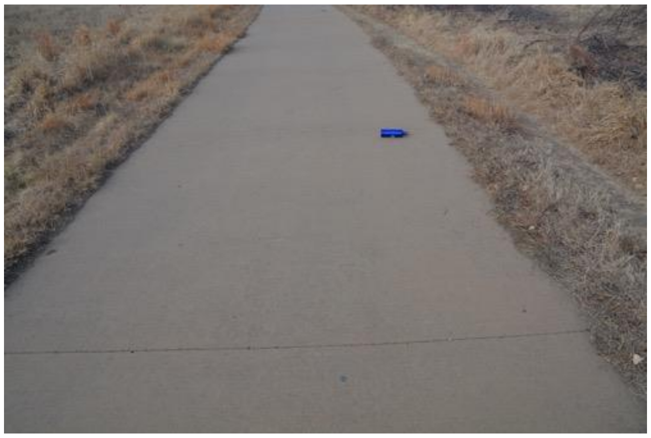
Final Accuracy: 87.5%

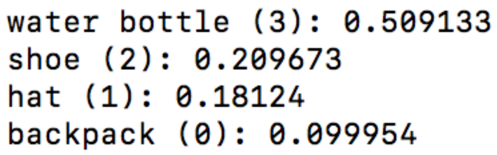
Stage 2

The second stage of this project revolved around implementing an object detection network by retraining Google's MobileNet platform for the 4 categories of anomalies in the project. These 4 categories were backpacks, shoes, hats, and water bottles. In this section, I will discuss my reasoning for my choices of the MobileNet platform and of the categories for my anomalies.

This stage of my project began by deliberating between the different types of pre-trained object classification networks already existing. These include GoogleNet, AlexNet, MobileNet, and InceptionV3. Of these 4, MobileNet had the second highest accuracy based on the dataset it was trained on (roughly 90% on thousands of different objects). However, the reason I chose MobileNet instead of the most accurate, InceptionV3, was because MobileNet took up considerably less space compared to InceptionV3, but still with a respectable accuracy. This aspect of MobileNet was good for real-time image classifying, which is what the goal of this project was.

The reason I chose the 4 categories of objects for anomalies is twofold. First, I chose common objects that hikers would likely take on the trail to best simulate possible items that would be lost and could lead to the hiker themselves. Second, I was limited by the amount of time and data I could possibly collect in the field, so I decided to limit myself to 4 categories to not run into data collection problems. A sample object classification can be seen in Figure 16. The final accuracy achieved was 93%.





```
water bottle (3): 0.509133
shoe (2): 0.209673
hat (1): 0.18124
backpack (0): 0.099954
```

Figure 16: Image of Water Bottle on Trial and MobileNet Classification Probabilities

Final Accuracy: 93%

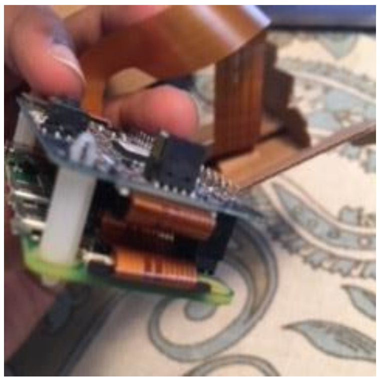
Stage 3

Stage 3 of my project revolved around implementing my model autonomously on a low-power device. I was able to construct the Google Vision Kit and import my combined super model onto the kit for autonomous use.

Google Vision Kit

Below are pictures taken from the construction of the kit.

VisionBonnet Cable VisionBonnet Board Raspberry Pi Zero



I was able to freeze my models into computational graphs, export them to the Google Vision and Raspberry Pi, export my Python scripts to the Google Vision, take images with the Google Vision, and pass images taken through my models on a computer.

Conclusion

This project reported on the possibilities of creating a novel implementation of transfer learning by combining anomaly detection and object classification into one functioning "super" model. The anomaly detection part of this project was implemented using a deep auto-encoder with two layers, which achieved an 87.5% final testing accuracy. The auto-encoder model continually suffered from lack of quality data and lack of data in general, but the introduction of color into the model proved to be able to compensate for the amount of data that was collected. The object classification part of my project was hindered by the lack of data from the field, but this was solved by obtaining online images (Google). The re-trained MobileNet model achieved 93% accuracy on the testing dataset. Both the results from the project surpass the goals set forth in the Introduction of the paper, allowing it to be classified as a success.

However, machine learning projects must always analyze what possible improvements and ideas can be made in the future. Aside from the obvious improvements of increasing the accuracy of the object classification and anomaly detection models, several improvements exist. First, the model could implement an image component separation scheme, like the scheme self-driving cars use to distinguish between cars and passengers. Another possible improvement could be to decrease the time needed for the model to conduct transfer learning. On the topic of transfer learning, another improvement could be to condense the two models into one neural network. Another interesting idea moving forward is to use a Generational Adversarial Network (GAN). GANs are neural networks that have the capabilities to construct an entire dataset from a few images by varying different qualities of the image to create more data. GANs could therefore be used to construct a more extensive dataset on which to train the main model.

Works Cited

"Artificial Intelligence Neural Networks" [Digital image] (n.d.). Retrieved Google search: artificial intelligence neural networks

Chollet, Francois (2016) "Building Auto-encoders in Keras." The Keras Blog

"Deep Auto-encoder" [Digital image] (n.d.) Retrieved from https://medium.com/@curiousily/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd

Hirano, Yutaka et al. (n.d.). "Industry and Object Recognition: Applications, Applied Research and Challenges."

Krizhevsky, Alex et al. (2012). "ImageNet Classification with Deep Convolutional Neural Networks." Advances in Neural Information Processing Systems

Lyudchik, Olga (2016). "Outlier Detection using Autoencoders." CERN Non-Member State Summer Student Report

Singh, Karanjit; Upadhyaya, Shuchita (2012). "Outlier Detection: Applications and Techniques." IJCSI International Journal of Computer Science Issues, 9