Machine Learning for Random Number Generation

Dylan Rae | 30020151

ENEL 525

Prepared for Dr. Leung & King Ma

December 17, 2021

Table of Contents

Table of Figures

Introduction

Random numbers are a necessary input for the operation of many technological innovations we use every day. With machine learning innovation and interest at an all-time high, this paper explores how a trained machine learning model can be used to generate *true* random numbers.

Background

As computational technology has progressed, the need for truly random numbers in computer science grew. Algorithms for generating random numbers were quickly designed, however these algorithms required a seed input to start initiate the generation. Seeds were often used from sources such as the system tie recorded by the operating system. These types of random numbers were known as *pseudo-random* as the process seeding the number generator is not random. Soon, computer science required *true random* numbers for fields such as machine learning and cryptography. The seeds for the number generators used in these applications extracted data from a random process found in nature. Commonly used random processes include radioactive decay, Brownian motion, and atmospheric noise. The National Institute for Standards and Technology (NIST) maintains and lists standardized testing procedures for evaluating the effectiveness of random number generators for high security applications such as cryptography [1].

Motivation

Current popular random process used for *true random* number generation are expensive and inaccessible for many organizations. Researchers are exploring other types of natural phenomena that are suitable for *true random* number generation [2]. Lava lamps are vases that display the movement of a gel when it is exposed to heat and allowed to cool after rising. The moved of the gel is an unpredictable and high entropy process. By extracting data from this random process, an inexpensive true random number generator could be constructed.

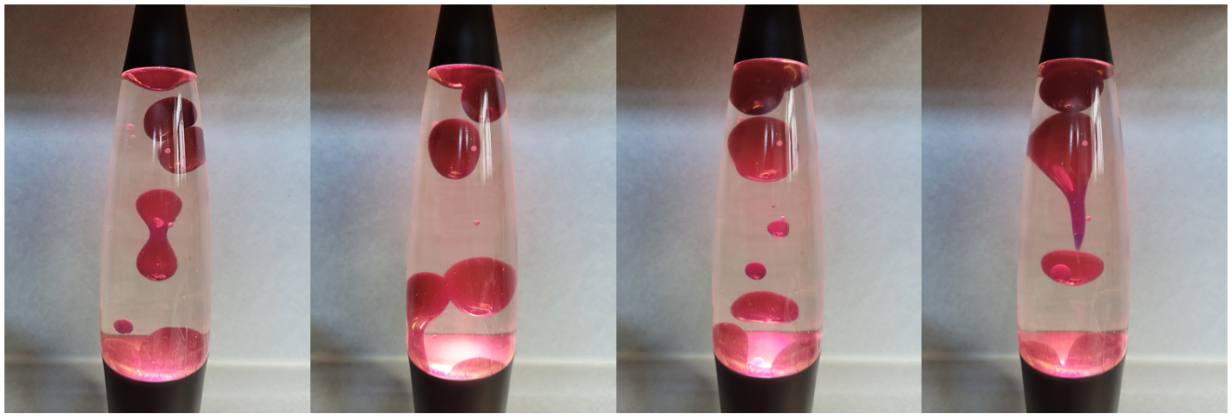


Figure 1: Snap Shots of Lava Lamp Illustrating Random Process

Methodology

To extract the data from the lava lamp that describes the random process, the following methodology will be used. First, pictures will be gathered from the lava lamp and these pictures will be annotated. Next these pictures will undergo data preprocessing to be prepared for input to an object detection machine learning model. This model will be trained and continually evaluated until the model's error is sufficiently low. Finally, the model with be tested and used to seed a random number generator. By evaluating the histogram produced by a normally distributed random number generator, the effectiveness of the model at generating *true random* numbers will be determined.

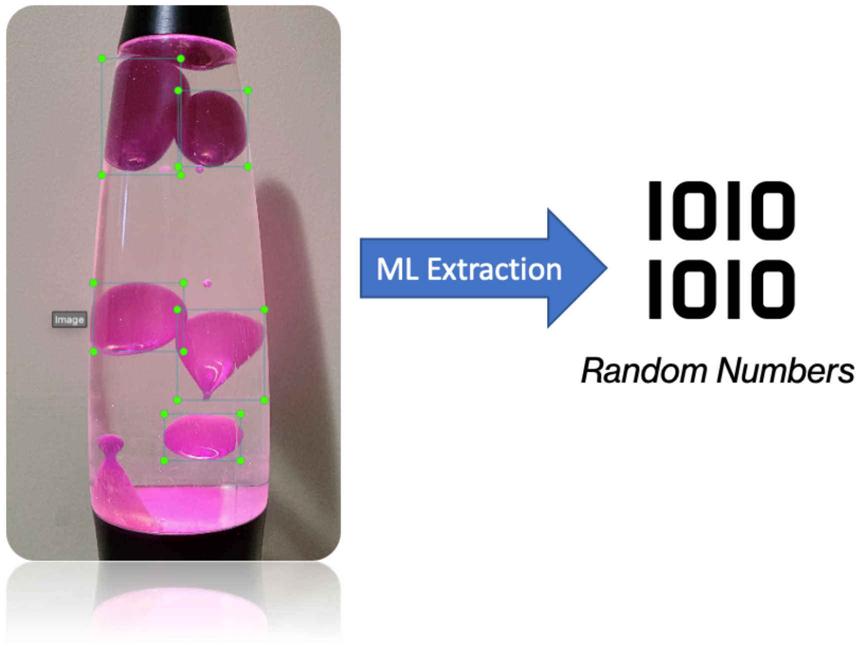


Figure 2: Random Data Extraction

Data Gathering

Pictures of the lava lamp were taken at 5 second intervals to ensure enough time had passed for the lava lamp to change positions. Pictures were taken with the lava lamp in two separate locations to help ensure the dataset could train the model to function when the lava lamp was in different locations. To further help with the flexibility of the applicability of the model, pictures were taken from a variety of angles and positions. In total 164 images were taken providing 699 samples in total (average of 4.3 samples/image).

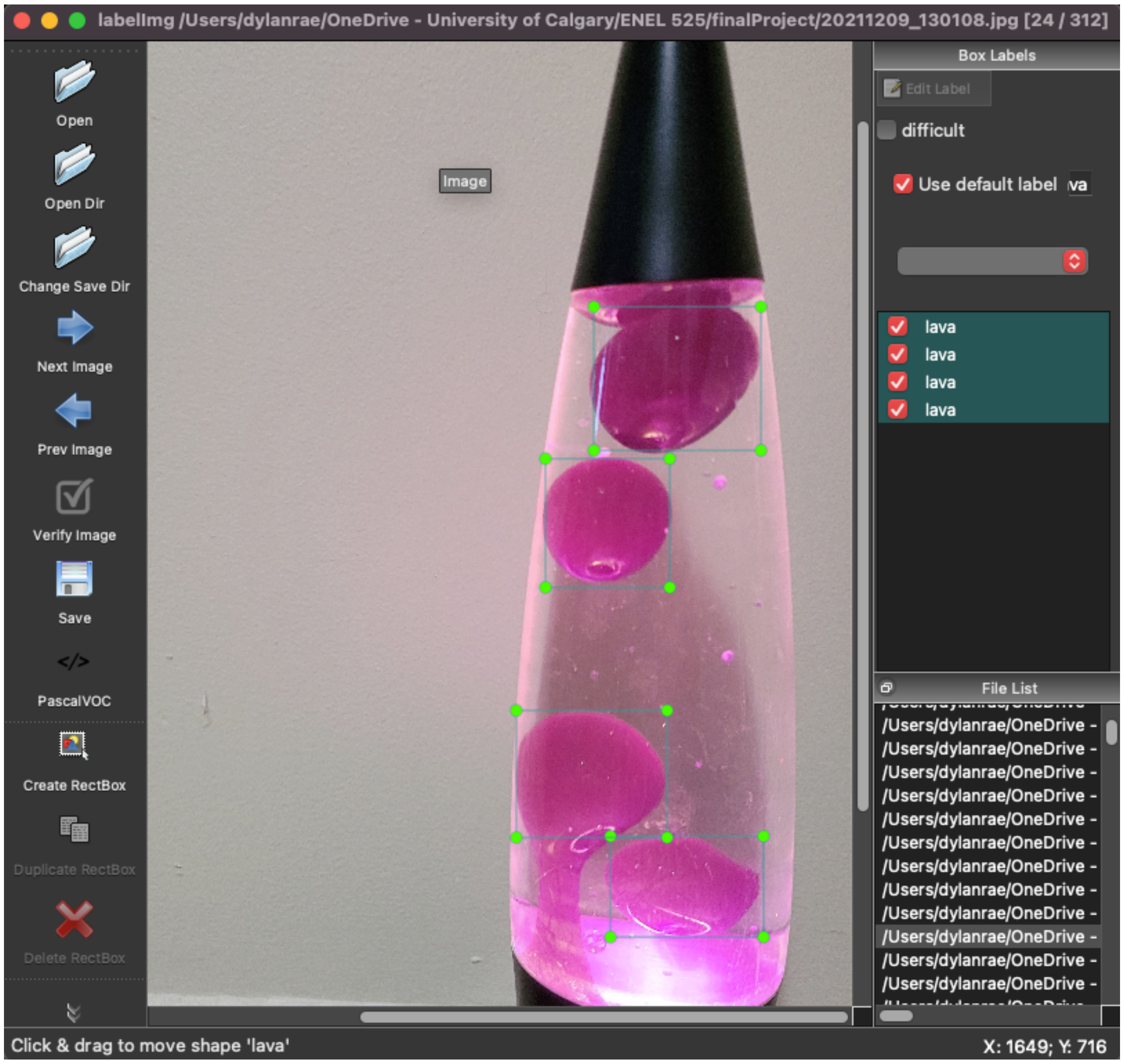


Figure 3: Image Annotation

A python program called LabelImg (see above figure) was used to annotate these images. The resulting annotation output was saved in xml format for later processing.

Data Preprocessing

The xml file from the annotation stage provided a convenient format to store annotation data from a specific image. The xml annotation data described the coordinates of each bounding box (x_1, y_1, x_2, y_2) and the name of the class describing the object within the bounding box ("lava"). This data was loaded into a pandas DataFrame using he xml module from python to parse the xml. The records of the DataFrame were randomly shuffled to evenly distribute the various backgrounds and image angles throughout the dataset. Once complete, this DataFrame was exported to a csv file for convenient and compact storage.

Upon reviewing the csv file, it was determined that the annotation software produced some invalid data samples (bounding box coordinates outside the range of the picture). Thus, a python script was used to check every sample's data for validity and removed any records that did not meet the requirements from the csv. The final step to finish preparing the data was to convert the csv to a TensorFlow record file. This is a data format based on NumPy arrays and is a standardized layout for data being fed into TensorFlow models.

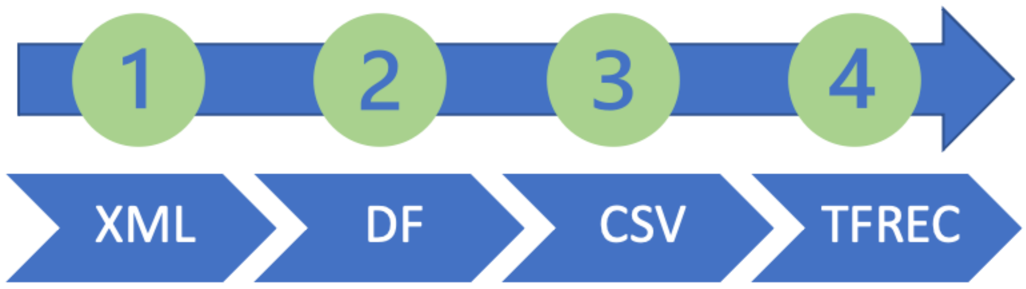


Figure 4: Data Preprocessing Steps

Model Selection

Object detection machine learning models are complex to design due to the multiple data inputs that need to be fed into the network and the multiple targets the network must predict. The inference and training should be performed as quickly as possible; however, the complex design makes this challenging. TensorFlow has many prebuilt models publicly available that have been researched. Many of these models are available pre-trained on a standardized data set either for immediate use or to shorten training time to learn detection for new objects. It should be noted that these object detection models are only compatible with TensorFlow versions 1.x and Python versions 3.7 and below.

There are numerous notable network styles to consider, and active research is being performed to design more efficient and accurate object detection networks. The three main models considered were Convolutional Neural Networks (CNN), Single Shot Detectors (SSD), and Spatial Pyramid Poolers (SPP). The below table summarizes the main benefits and drawbacks of these models. Other models should be evaluated in future studies.

	CNN	SSD	SPP
Benefits	- Deeply studied - Easier to understand and self-architect	- Best detector for speed and accuracy for feature map extraction and convolutional filter applications [3] - Plentiful documentation	- Well suited to arbitrary image sizes and scales - Avoids repeatedly computing convolutional features
Drawbacks	- Training is unpredictable and long - Training is multiphase - Network predictions are comparatively slow at inference time		- Poor documentation - Few practical examples of implementation
Models	R-CNN, Fast-R-CNN, Faster-R-CNN, Mask-R-CNN	MobileNet, ResNet, SpaghettiNet	SPPNet

Figure 5: Object Detection Model Summary

An SSD was used as this model had plentiful documentation was available online and reported the fastest training times. Specifically, the *ssd_mobilenet_v2* model was selected and a pre-trained checkpoint form the common objects in context (COCO) dataset was utilized. The below figure depicts the network architecture of a similar SSD model [3].

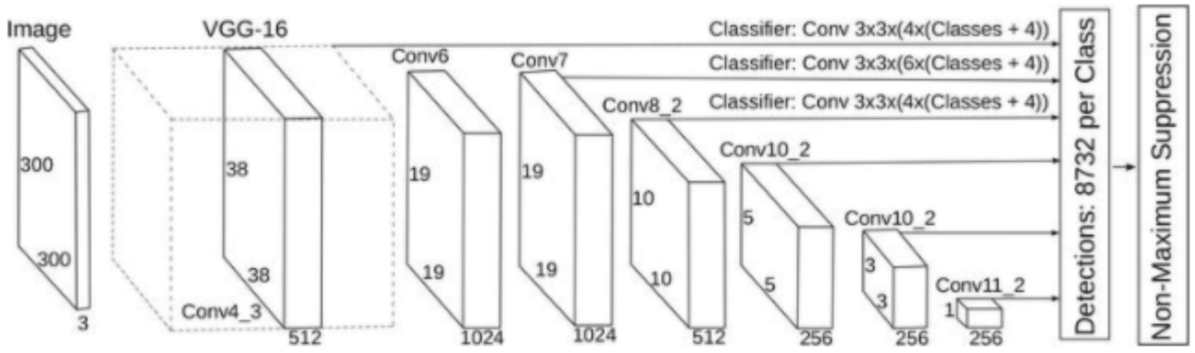


Image
300
300
3
VGG-16
38
38
Conv4_3
512
Conv6
19
19
1024
Conv7
19
19
1024
Conv8_2
10
10
512
Conv10_2
5
5
256
Conv10_2
3
3
256
Conv11_2
1
256
Classifier: Conv 3x3x(4x(Classes + 4))
Classifier: Conv 3x3x(6x(Classes + 4))
Classifier: Conv 3x3x(4x(Classes + 4))
Detections: 8732 per Class
Non-Maximum Suppression

Figure 6: SSD Network Architecture

Training and Evaluation

For conducting training on prebuilt TensorFlow models, TensorFlow provides a training script that simultaneously trains and evaluates the model. The model was trained with the recommended learning rate of 0.004 and an exponential learning decay factor of 0.95. The input was data was split into 150 images for training and 14 images for evaluation. The TensorFlow script performed 100 steps of training, then ran the model against the evaluation set to monitor progress. The overall total loss equation is depicted below. It is a function of the error in the confidence of the predicted bounding box location, and a function of location loss. The location loss describes how far the predicted bounding box is from the target ground truth. The training script also output the current intersection over union (IoU) performance

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)$$

Figure 7: MobileNet SSD Total Loss Function

The initial total loss started out at 95. For the first 24 hours of training, the total loss decayed linearly to 13. During the final 12 hours the error decayed exponentially from 13 to a final total loss of 10.54. Some IoU values rapidly rose to 1, while others struggled to get above 0.05. The ideal total loss by the end of training was less than 1, with values below 2 also being accepted. Due to time and computational constraints, the model was not able to be trained to this accuracy. Having a larger data set would be beneficial for improving the final trained state of the model.

Model Performance

To analyze the performance of the model, inference was performed on the entire data set. The detection confidence scores were below the target of >0.9 at an average 0.34. While disappointing, this was expected with the high total loss and IoU errors at the end of model training. The top and bottom 3 detection confidence scores were recorded and the images were displayed with the ground truth and predicted bounding boxes. The figure below depicts the highest performing image. Inference time also came in lower than expected with an average inference time of 1.29 seconds per image.

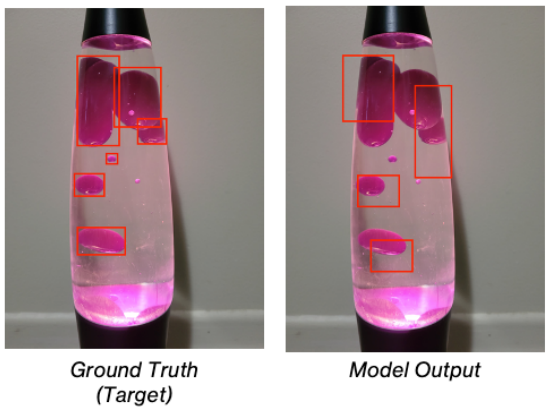


Figure 8: Trained Model Performance

Random Number Generation

To generate random numbers, data about from the random process displayed by the lava lamp needs to be extracted. With the trained model, the number generator can now be tested. The below equation was used to generate a seed for a random number generator from the data output after inference. The equation calculates the center of each blob of lava, and then calculates the weighted average position by multiplying each pair of coordinates by the size (bounding box area) of the lava blob. Finally, the x and y coordinates were multiplied together to get a single seed number.

$$Random\ Seed = \left[\sum_{i=0}^{n} \frac{(x_{2,i} - x_{1,i})}{2} * area\right] * \left[\sum_{i=0}^{n} \frac{(y_{2,i} - y_{1,i})}{2} * area\right]$$

Figure 9: Random Number Seed Extraction Equation

The seed generation was tested by generating numbers in the ranges [0, 10], [0, 100], and [0, 1000]. The below histograms depict the distributions of the generation. Ideally the plots should be uniform and rectangular, however I predict there are two factors preventing this result. First, the model is not accurate enough to successfully extract the random data from the lava lamp. This likely results in a bias being introduced that disqualifies the result from being considered a *true random* number generator. Second, the data sample is not large enough to see the true distribution of random numbers being generated as the sample sizes tends to infinity.

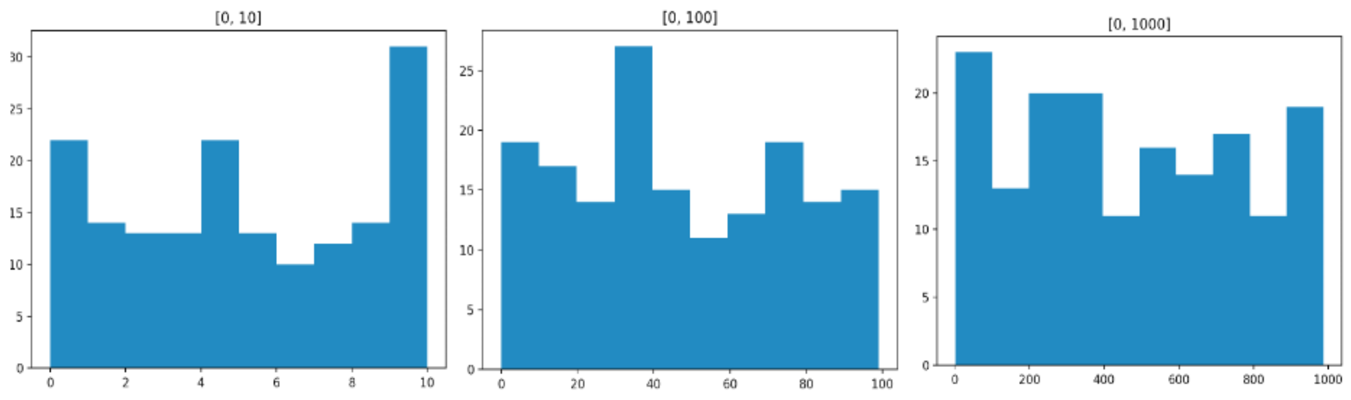


Figure 10: Random Number Generation Distribution

Future Work

The most important next step is to improve the accuracy of the object detection model. First, the process for picturing gathering will be automated to collect a much larger data set of images. These new images will also need to be annotated. Using the new data set, multiple instances of the new model will be trained. These new instances will experiment with different configuration such as no starting checkpoint, varying learning rates, and different variations of prebuilt TensorFlow object detection models. Training these models on google collab will be explored to improve the training time currently being performed on a PC.

The improved model will then be tested for its effectiveness at extracting random data which will be used to determine if the numbers being generated are *true random* numbers. NIST maintains standardized testing procedures for evaluating the effectiveness of random number generators for high security applications such as cryptography [1]. The dataset will be evaluated to determine its efficacy for *true random* number generation.

References

[1] E. Barker and L. Bassham, "Random Bit Generation," NIST, 9 Jul 2014. [Online]. Available: https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software. [Accessed 14 Dec 2021].

[2] R. Katyal, A. Mishra and A. Baluni, "True Random Number Generator using Fish Tank Image," *International Journal of Computer Applications,* vol. 78, no. 16, 2013.

[3] L. W. e. al., "SSD: Single Shot MultiBox Detector," *European Conference on Computer Vision,* vol. 9905, pp. 21-37, 2016.

Appendix

Presentation Slides

Machine Learning for Random Number Generation

Dylan Rae

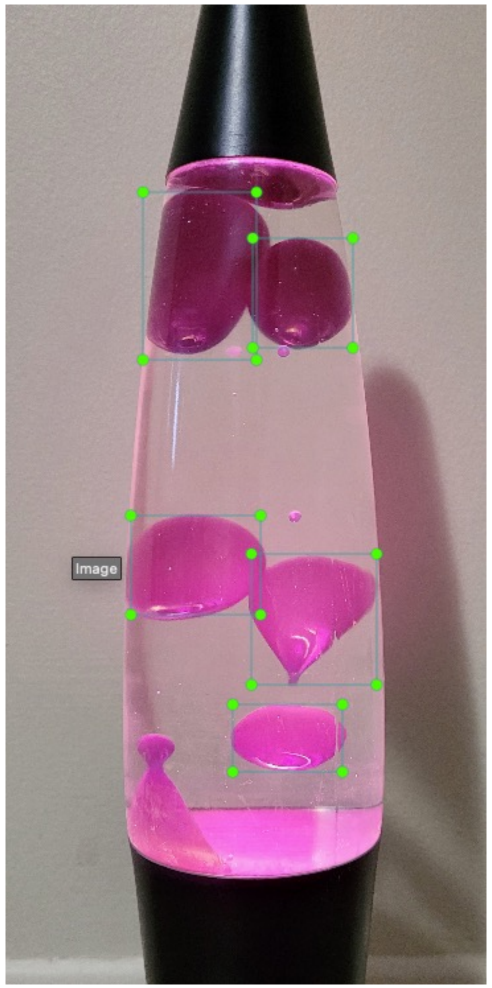
30020151

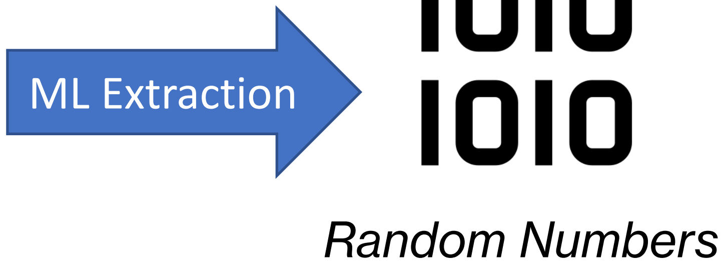
Background

- Random numbers are needed in many areas of computer science
  - Cryptography
  - Machine Learning
- Typically, a computer only generates *pseudo-random* numbers
- By extracting data from a source of entropy (random process), a *true random* number can be generated
  - Radioactive decay
  - Brownian motion
  - Atmospheric noise
- Researchers are exploring other natural phenomena that random data can be generated from (see paper here)

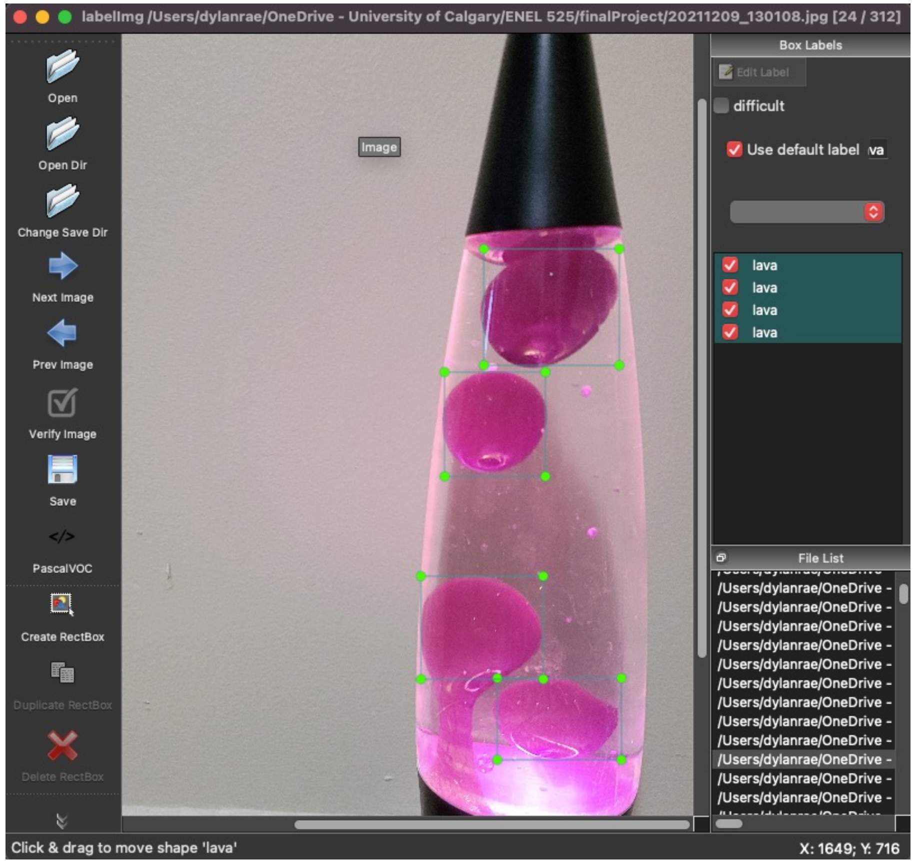
Problem & Motivation

- Extracting data from sources of entropy mentioned previously such these is complex and expensive
- **Problem**: Use entropy produced by a lava lamp to generate true random numbers
- The positions and sizes of the "lava" in the lava lamp will be random
- By detecting the position and sizes of the "lava", this data can be fed into a random number generator





Step 1: Data Gathering & Annotation



labelImg /Users/dylanrae/OneDrive - University of Calgary/ENEL 525/finalProject/20211209_130108.jpg [24 / 312]

/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -
/Users/dylanrae/OneDrive -

Click & drag to move shape 'lava'   X: 1649; Y: 716

- Pictures were taken with 2 different backgrounds at 5 second intervals
- Variety of angles and positioning used
- Program called ***labelImg*** was used to annotate and save in xml format

164 *images*

699 *samples*

Step 2: Data Preparation

1 XML
- Stores bounding box sizes and position
- Xml module makes it easy to extract data

2 DF
- Xml data loaded into a DataFrame

3 CSV
- Exported to csv for easy storage and viewing
- Wrote a data validator to remove invalid entries

4 TFREC
- Data format based on NumPy arrays
- Standardized for feeding into TensorFlow models

Step **3**: Model Selection

CNN
Convolutional Neural Network

- Slow training for object detection
- Uses selective search algorithm and bounding box regression
- Models are R-CNN, Fast-R-CNN, Faster-R-CNN, Mask-R-CNN

SSD
Single Shot Detector

- Good for detecting images of varying sizes
- Fast training and inference
- Plentiful documentation
- Models are Inception, MobileNet, ResNet, SpaghettiNet

SPP
Spatial Pyramid Pooling

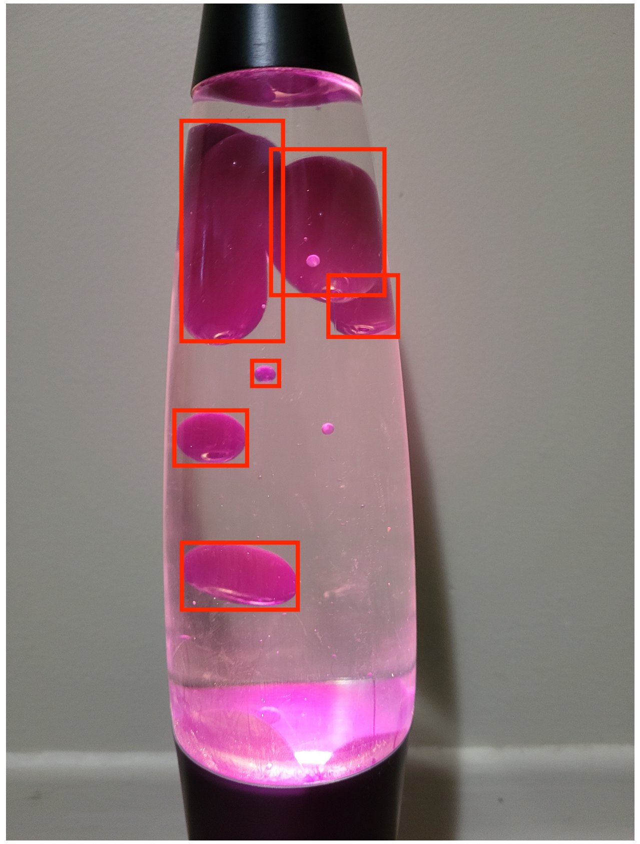
- Good for arbitrary image size and scale
- Avoids repeatedly computing convolutional features
- Popular model is SPPNet

Step **4**: Training and Evaluation

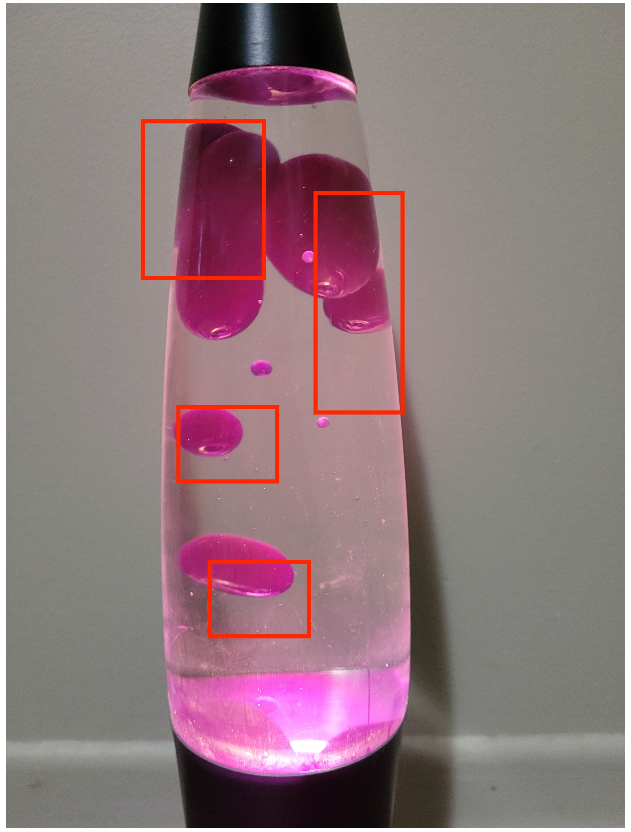
- Trained with a model called *ssd_mobilenet_v2*
- Started with checkpoint of model after being pretrained on the *COCO* data set (common objects in context)
- Used the recommended learning rate of *0.004* with an exponential decay factor of 0.95
- Training data was 150 images, evaluation set was 14
- Using a TensorFlow provided script model was trained through 100 steps, then used an evaluation set to monitor progress
- First 24 hours linear decay from 95 → 13, next 12 hours exponential decay very slow
- Total of 11508 steps with final total loss of 10.54 (total loss compares error from class and position detection)
- Some IoU values struggled to get close to 1 (how well the predicted bounding box matches ground truth)

Step **5**: Model Performance

- Very low detection confidence scores (average of 0.34)
- Occasionally can detect correctly, but errors need to be lower and confidence scores need to be >0.9
- Takes approx. 1.29 secs to perform an inference on an image



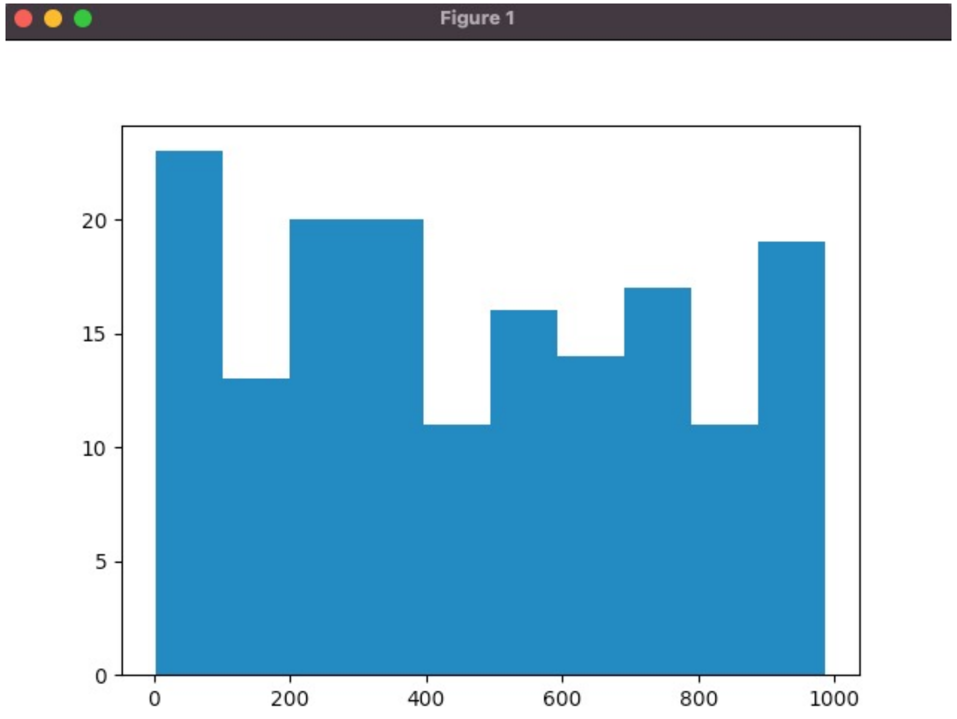
Ground Truth (Target)



Model Output

Random Number Generation

- Used python random number and seeded positions of boxes to a generator
- Generated numbers between 0→ 1000



Next Steps

- Automated process for data gathering
- Annotate this data to produce a much larger data set
- Train multiple instances of model using google collab
  - From scratch (no checkpoint)
  - Varying learning rate
  - Other pre-trained checkpoints
- Test according to NIST standards outlining requirements for true random number generation available here