Midterm: Running an Object Detection Model

Shakira Kashif

Christian Mpabuka

Devin Prikryl-Martin

Houston Community College

ITAI 1378 Computer Vision

Professor Patricia McManus

December 2, 2024

Midterm: Running an Object Detection Model

Our group, because of our varied schedules, struggled to find times where we could all meet synchronously. We communicated asynchronously and decided the best way to tackle the project as a group was to try to each run a model and come together with our results to try to use what we learned individually to create a really excellent model. Unfortunately, we all struggled with the technical side of the project and failed to produce even one model that has a precision better than 17%. The model attached is the most functional of the three.

Shakira: During this assignment, I encountered several challenges while attempting to load and process the Oxford-IIIT Pet dataset for model training. Although I made progress in understanding the dataset's structure and requirements, I ultimately could not run the model due to unresolved issues related to data preprocessing and time constraints. This essay reflects on the failed experiments, challenges faced, problem-solving approaches, unresolved issues, and key learnings from the assignment.

Failed Experiments

My initial attempts to load the dataset involved specifying version 3.*.*, assuming it would work with TensorFlow Datasets. However, this led to an AssertionError because the dataset only supports version 4.0.0. After addressing this, I encountered another issue when using the as_supervised=True argument, expecting the dataset to return (input, label) pairs. This resulted in a NotImplementedError because the dataset is stored in the ArrayRecord format, which is incompatible with this approach.

Further complications arose when I attempted to split the dataset manually into training and testing subsets. While the TensorFlow Datasets API provides methods like .take() and .skip() for such operations, ensuring proper pairing and compatibility with a model's input format became increasingly complex. These issues collectively hindered my progress toward running a model on the dataset.

Challenges Faced

One major challenge was understanding the implications of dataset versioning. Initially, I was unaware that the specified version was unsupported, leading to wasted time troubleshooting errors. Another significant hurdle was dealing with the ArrayRecord format, which posed unique challenges in loading the dataset. Unlike other TensorFlow datasets, this format does not natively support the as_supervised argument, requiring a more hands-on approach to data extraction and preprocessing.

Preprocessing the dataset also proved to be a time-intensive task. The absence of automated (input, label) pairing meant that I needed to manually extract and prepare the data. This required additional steps, such as resizing images, batching, and converting labels into a usable format. Without a predefined preprocessing pipeline, I found it difficult to adapt to these requirements in the limited time available.

Problem-Solving Approaches

To address the versioning issue, I removed the version specification and allowed TensorFlow Datasets to default to the latest version, which resolved the AssertionError. I also explored the TensorFlow Datasets documentation extensively to better understand

how to handle the ArrayRecord format. This led to the realization that I needed to load the dataset without as_supervised and process it manually.
For splitting the dataset, I used TensorFlow's .take() and .skip() methods based on the number of examples provided in the dataset metadata. I also inspected sample records to understand the dataset's structure and determine how to extract images and labels. These steps helped me make partial progress in preparing the data for model training.
Unresolved Issues and Why I Couldn't Run the Model
Despite my efforts, I could not complete a preprocessing pipeline to extract images and labels, resize the images, and prepare batches for model training. The primary issue was the incompatibility of the ArrayRecord format with standard supervised loading methods, which disrupted my initial workflow. Creating a custom pipeline for preprocessing and pairing data required advanced knowledge of TensorFlow's dataset manipulation capabilities, which I couldn't fully implement within the assignment's timeframe.
Time constraints also played a significant role in my inability to run the model. Preprocessing data is often as resource-intensive as training the model itself, and I underestimated the time required to address the unique challenges posed by the dataset's format.
Learnings and Takeaways
This assignment underscored the importance of understanding dataset formats and their implications on model training. Thoroughly reviewing documentation before implementation could have saved time and prevented many of the errors I encountered. Additionally, this experience highlighted the critical role of preprocessing in machine learning workflows. Preparing data pipelines often takes as much effort as model training, requiring careful planning and sufficient time allocation.
In conclusion, while I faced numerous obstacles and could not run the model, the assignment provided valuable lessons in handling dataset challenges, adapting to unforeseen issues, and emphasizing the importance of thorough preparation. These experiences will guide me in future projects as I continue to develop my skills in machine learning and data processing.

Devin: My journey trying to alter this object detection model so that it would run well and produce decent results has been quite the learning experience. It definitely reinforces the idea that one has to fail spectacularly first in order to succeed. Because I am a novice programmer, I relied heavily on Claude, ChatGPT and Gemini to help me troubleshoot and problem solve errors in the code, but sometimes I ran into technical issues getting the models to run that even the GPTs couldn't solve. Typically in this scenario I would attempt to troubleshoot one cell of code with suggested replacements for so long that it would start to create bugs in the earlier code. On several occasions I would discern that the model had strayed so far from the original with no progress to show for it, that it would be better to start again from scratch. This lead to at least 7 iterations of models with varying dataset and model combinations, all of which failed at some critical point and could not be salvaged. I initially tried to change the dataset to the Oxford-IIIT Pet and the EfficientDet model, a decision I made based on the recommendation of Claude, because it is a smaller

dataset, already available through TensorFlow, has high quality images and only 37 breed categories within a single domain (or category), pets. It is considered a good dataset for beginners because the data is high quality, well labeled and generally well formatted. This makes it easier to parse for beginners. This change yielded another model with 0 true positives, just like the model from the lab that I began with. I tinkered with that model for several hours and only ever achieved 10 true positives out of 560 images. I then looked into what might be a good model to replace the SSDMobileNet model to improve results. I started with the GPT recommendation, EfficientDet. I chose this model because it has certain strengths I thought might be useful, such as improved feature extraction compared to SSD, the fact that it is superior at detecting small objects, and the fact that it is still lightweight enough for colab. I attempted several versions of this combination of dataset and model, and spent probably 12 hours trying to first, alter the original code to produce results and then, trying to build new code without an existing framework, in the hopes that it was the layout of the model and not its components that were causing the model to fail. I attempted different preprocessing methods as well, suggested to me by my AI helpers. At some point I hit a wall with that model, where it would get snagged on one improperly named data variable, I would try to use GPT to fix it, which would produce a different error. Then I would try to fix the second error, and that solution would cause the first error to happen again. Between that issue, and the constantly disconnecting runtime (I ended up having to purchase compute) I was at my wits end. This happened cyclically until I gave up on that model and returned to the base model in lab 9, to begin again from the drawing board. This lead me to my first true breakthrough after several days and many hours of attempts. I let go of the Oxford IIIT dataset and looked into what might be a better dataset for specifically SSDMobileNet. That's when I remembered that this pretrained model was trained on the COCO dataset, so I went about trying to use that dataset instead. At first, I was exasperated because the dataset was too large for the amount of disk space I had. I realized I needed to use a subset of COCO

Christian: **Dataset Selection:** PASCAL VOC 2007This dataset contains the data from the PASCAL Visual Object Classes Challenge 2007, a.k.a. VOC 2007. It includes 9963 images, each containing a set of objects from 20 different classes, for a total of 24640 annotated objects.
PASCAL VOC 2007: is widely used for benchmarking object detection models, and contains 20 object categories, providing a balanced mix of complexity and size. Available via TensorFlow Datasets for ease of integration.

During this assignment, I faced several challenges. First, there were issues with downloading and preparing the PASCAL VOC 2007 dataset; at times, the download failed, or the files were corrupted. I resolved this by ensuring a stable internet connection and updating the TensorFlow Datasets library. While preprocessing the data, I encountered errors related to accessing annotation keys and the overall data structure. Thankfully, I was able to address these issues with the help of Gemini and ChatGPT. Additionally, my model exhibited signs of overfitting, which I tackled by applying regularization techniques such as data augmentation and early stopping. Another challenge involved inconsistent inference times, particularly when trying to optimize for real-time performance. I experimented with different model architectures and applied quantization to strike a balance between speed and accuracy. At one point, my code stopped working entirely, forcing me

to start over. As the model's performance plateaued, I adjusted the learning rates, tried out different optimizers, and conducted hyperparameter tuning to improve convergence. I also faced difficulties when training the model on all 20 object categories, which resulted in poor performance for some classes. By focusing on a smaller subset of relevant classes, I was able to enhance accuracy. Finally, I encountered issues with maximum suppression while filtering bounding boxes, but these problems were resolved by carefully managing tensor shapes before applying the NMS technique.