

UCLA

UCLA Electronic Theses and Dissertations

Title

Active Learning for Object Detection on Website Pages

Permalink

<https://escholarship.org/uc/item/6fz5r7q0>

Author

Ward, Anders Seth

Publication Date

2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Active Learning for Object Detection on Website Pages

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Anders Seth Ward

2023

© Copyright by
Anders Seth Ward
2023

ABSTRACT OF THE THESIS

Active Learning for Object Detection on Website Pages

by

Anders Seth Ward

Master of Applied Statistics

University of California, Los Angeles, 2023

Professor Yingnian Wu, Chair

In this paper we conduct transfer learning on the Faster R-CNN ResNet-50 FPN V2 object detection model from *Benchmarking Detection Transfer Learning with Vision Transformers* [12] to a new problem space: detecting what parts of a webpage can be clicked on. The model predicts the bounding boxes and class labels for what part of a website image can be clicked on, and what the result will be. We found that training the feature extraction layers of the model was detrimental to the learning (354% higher loss after 10 epochs), but that training the RPN layers was not detrimental. We additionally found that the most effective resizing for website imagery was to 224 x 224 pixels. Since the meta trainer we developed did not produce usable data, we experiment on using the data diversity approach in active learning to select the most representative images to train on. Compared to our control, the model trained on the active learning dataset scored 12.5% better on the validation data. However, with such a small dataset of 10 labeled images, the model was unable to generalize and did not predict any of the 15 objects on a labeled image outside the training set.

The thesis of Anders Seth Ward is approved.

Michael Tsiang

Frederic R. Paik Schoenberg

Yingnian Wu, Committee Chair

University of California, Los Angeles

2023

To my parents, Douglas Ward and Ginger Ward

TABLE OF CONTENTS

1	Introduction	1
1.1	Object Detection	1
1.2	Active Learning	4
1.3	Learning to Navigate the Web	5
2	Methods	8
2.1	Meta Trainer	8
2.2	Faster R-CNN derived classifier	12
2.3	Initial Training	13
2.4	Active Learning	14
3	Analysis	19
3.1	Meta Trainer	19
3.2	Resizing the Image	23
3.3	What Layers to Train	24
3.4	Active Learning	26
4	Conclusion	29
5	References	31

LIST OF FIGURES

1.1	The Faster R-CNN model, with shared convolutional layers, a RPN, and the Fast R-CNN structure built in. Ren, 2016, arXiv, 1506.01497. Copyright 2016 by arXiv.	3
1.2	World of Bits paradigm. Shi, 2017, Proceedings of Machine Learning Research, 70. Copyright 2017 Proceedings of Machine Learning Research	5
2.1	Meta Trainer mapping for https://www.allrecipes.com/new-pizza-hut-bucket-hat-7375405 . Each dot represents a click and the associated class: green for url redirect and red for new tab opened. The background class is hidden. . .	10
2.2	Input image after default transformation including pixel normalization and image resizing to 1248 x 800.	13
2.3	Screenshot closest to the 0th cluster centroid. Sourced from www.allrecipes.com	16
2.4	The exact 0th cluster centroid	17
2.5	Screenshot closest to the 1st cluster centroid. Sourced from www.twitch.tv .	17
2.6	The exact 1st cluster centroid	17
2.7	Screenshot closest to the 2nd cluster centroid. Sourced from www.amazon.com	18
2.8	The exact 2nd cluster centroid	18
3.1	Max Pooling KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels are decent, but there are missing labels towards the bottom of the image. The notifications popup in this screenshot demonstrates the some of the variation the meta trainer experienced on the same url. Sourced from www.allrecipes.com	21

3.2	Max Pooling KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels over predict objectness of the image, and do not center the class labels towards class locations. Sourced from www.allrecipes.com	22
3.3	Smooth KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels inconsistently predict objects in the center of the image, and do not correctly predict the orientation of objects. Sourced from www.allrecipes.com	23
3.4	Training loss over 15 epochs with different initial image resizing transformations.	24
3.5	Training loss over 10 epochs with different model parameters subject to training. For all models, the RoI heads are trained.	25
3.6	Best model predictions on the Wikipedia validation image. This model was trained for 11 epochs on the active learning dataset, after 30 epochs of training on the initial dataset. Sourced from www.wikipedia.org	27

LIST OF TABLES

2.1	The websites screenshots used for the active learning component	15
3.1	Learning Dataset Summary - Part 1	26
3.2	Learning Dataset Summary - Part 2	26
3.3	Model Component Analysis	28

CHAPTER 1

Introduction

1.1 Object Detection

Using object detection models to inform reinforcement learning (RL) agents is often applied in the field of autonomous vehicles [9]. In that use case, the object detection model simplifies the video feed from the vehicle’s cameras in order to better inform the RL agent. Reinforcement learning agents overwhelmed by large state and action spaces often fail to converge because trial and error is ineffective in these scenarios, due to the low likelihood of a reinforcement learning model stumbling upon the correct action sequence [13]. That means the learning process is usually computationally expensive [13]. Due to the influence of each action on the state space, these RL agents are presented with too many interdependent actions to find the comparatively sparse rewards. Therefore the object detection model is crucial for autonomous vehicles to understand the current state space.

Object detection models have seen a rapid growth in terms of predictive performance and speed [5]. These models take images as input and provide class-labeled bounding boxes for each class present in the input image. Besides predictive accuracy, speed is perhaps the most important metric for an object detection model [9]. The use cases that have propelled object detection research forward such as autonomous vehicles, surveillance systems, and robotics, all require that the model make real time predictions [9]. One of the first models to achieve the required speed and accuracy for these use cases was Faster R-CNN [5]. Faster R-CNN is the third model in a line that begins with R-CNN. R-CNN was developed in 2014 by researchers at Berkeley [6], who named it after its fundamental structure: region-based convolutional neural network. R-CNN generates 2,000 region proposals, extracts a feature

vector for each region proposal, and then uses a pre-trained SVM to classify the regions based on those feature vectors [6]. One of the main shortcomings of R-CNN is that it is a collection of 3 different components that cannot be trained end to end [7]. Additionally, the Selective Search algorithm for the region proposals is time expensive and cannot be trained to a specific object detection task [7]. The region proposals are fed independently into the CNN component, and then cached for the SVM, making it both time and space expensive [7]. Next came Fast R-CNN, which as the name suggests, greatly improved the speed of the model, and also improved the predictive accuracy of the model [7]. The researchers achieved this by coalescing the R-CNN structure into a single stage model that utilized a novel region of interest (RoI) pooling layer to share convolutional layer computations across all region proposals [7]. After the convolutional feature map is generated by the initial layers, the RoI pooling layer is applied to turn the proposals into fixed length feature vectors. The RoI pooling layer applies max pooling to each sub-grid in the original feature map, based on the region proposals it is given. Since the region proposals might be of varying sizes, the RoI pooling simplifies their information into fixed length vectors, and uses only the one feature map to do it [7].

Faster R-CNN improved on its predecessor mainly by introducing the Region Proposal Network (RPN) [5]. The region proposals had been the bottleneck of the Fast R-CNN model [5], and the new RPN is directly integrated into the detection pipeline, and does not use the selective search algorithm to find region proposals. The RPN can learn region proposals, and so is more generalizable to different tasks if needed [5]. The RPN is a fully convolutional network that generates a set of object proposal boxes, each with an objectness score. For computational speed, these convolutional layers are shared with the Fast R-CNN part of the network. The region proposals are generated by taking the output of the last shared convolutional layer, dividing it into $n \times n$ sliding spatial windows, mapping it to a lower dimensional space, and then two fully connected regression models give box location and classification [5]. Each region proposal is related to k anchors: reference boxes that parameterize the sliding window's center, scale, and aspect ratio. This approach ensures translational invariance [5], bolstering the network's generalizability and accuracy [5].

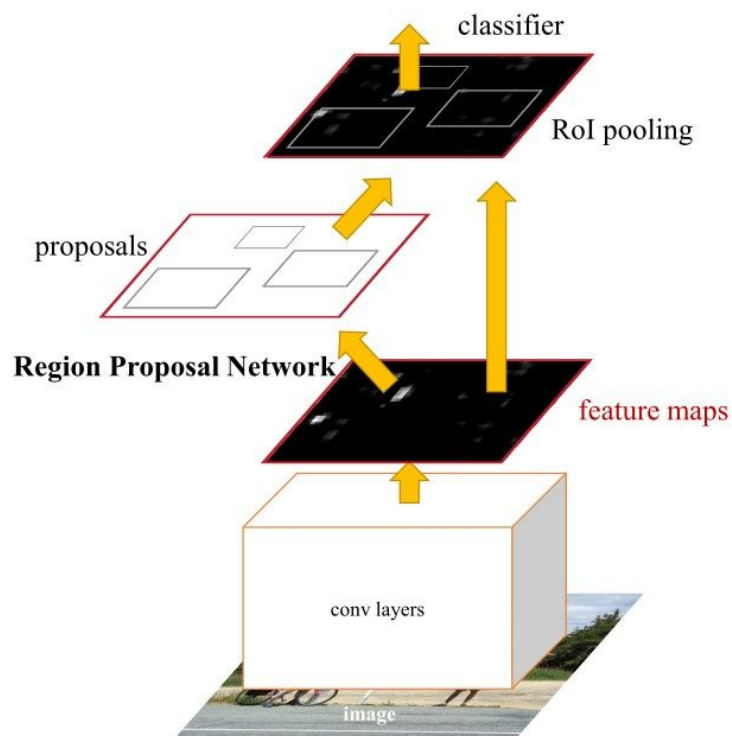


Figure 1.1: The Faster R-CNN model, with shared convolutional layers, a RPN, and the Fast R-CNN structure built in. Ren, 2016, arXiv, 1506.01497. Copyright 2016 by arXiv.

Faster R-CNN, which can operate at 5fps [12], has since been superseded by YOLO and SSD, which achieve 40 fps and 22fps respectively [12]. These models will not be discussed in this paper, but the main improvement can be summarized as being a single pass object detection framework, rather than a multi-stage framework.

1.2 Active Learning

One of the biggest challenges in object detection is getting enough labelled data. The deep learning paradigm central to object detection requires vast datasets, and in the case of object detection, each image might include multiple class labels and box coordinates. When labelling data is this time-expensive, the problem naturally lends itself to active learning. Active learning is a strategy to reduce the amount of labelled training data necessary for the model to train. It works by highlighting which unlabeled training examples would yield the most benefit to the model [1]. Data diversity and model uncertainty are the two main methods for estimating the most useful training examples for labelling [10]. Data diversity is the process of finding the most representative images, usually through a k-means algorithm. After clustering, we can select a sample of representative images from each cluster, and annotate those [10]. This saves the task of annotating every image in a cluster. Model uncertainty is the approach of estimating which images the object detection model will perform worst on [10]. One approach is to have the model to predict on unlabeled images, and then select the images with two different classes having a similar score, indicating that the prediction is close to a decision boundary [10]. This is called 1-vs-2 sampling. Another approach is to make altered copies of unlabeled images, and then select the images with the highest variance in model predictions across altered copies. Yet another approach is to learn to predict either the detection loss or the change in gradient from learning [10]. In this paper we explore a data diversity approach by selecting the images closest to the centroids of the k-means clustering.

1.3 Learning to Navigate the Web

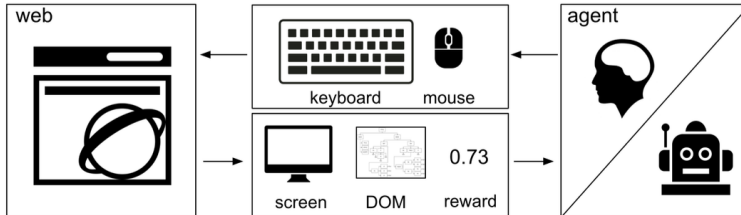


Figure 1.2: World of Bits paradigm. Shi, 2017, Proceedings of Machine Learning Research, 70. Copyright 2017 Proceedings of Machine Learning Research

Another paradigm in which reinforcement learning agents are presented with large state and action spaces is internet tasks. In the World of Bits platform [8], reinforcement learning agents are trained to perform tasks on the internet based on natural language input. In these environments, the RL agent is fed the HTML DOM [8]. This is the underlying code for displaying websites visually to humans. Gür et al presents the following RL agent task: “Book a flight from WTK to LON on 21-Oct-2016” [13], the RL agent is confronted with an extremely large state and action space. Although this task appears simple to a human, the number of possible web elements to click on, the information presented by the airline website, and the long series of steps required to complete the purchase present a challenge for the RL agent. Previous work on this topic had used *expert demonstrations*, wherein a human is recorded completing the task, and then the RL agent is initially trained to copy the solution [13]. This avoids the first step in the learning process wherein the AI never embarks on the correct path to the solution. Due to the low likelihood of the RL agent stumbling upon the correct action sequence, an expert example speeds up the training tremendously [13]. The model is then generalized by treating the cities, input text, and dates as variables [13]. These researchers explored an alternative approach to expert demonstrations, as it is nearly impossible to create an expert demonstration for each task that the RL agent might be presented in a natural language setting. Additionally, it has been shown that expert demonstrations do not guarantee convergence for more complicated tasks [13]. Therefore, Gur et al developed curriculum-DQN, a method of decomposing the natural language instructions into

simpler sub-instructions. That way the RL agent can successively learn simple tasks with more available rewards and smaller state spaces [13]. They also developed “meta-trainer”, which annotates navigation sessions, expert or not, with generated instructions. Therefore it is able to create expert demonstrations from unlabeled data, even if that data was generated by the RL agent [13].

This paper explores the task of simplifying the state space of a webpage for an RL agent, with an alternate approach. We use an object detection model to transform the image of a webpage into a list of class labeled boxes. Similarly to how an autonomous vehicle uses object detection to see the road and obstacles, the RL agent would see submission forms and redirect links. By decomposing the image into class labeled boxes, the RL agent is able to make decisions in a smaller state and action space. The state space is smaller because the 50,176 pixel image has been simplified to the class labeled boxes, and the action space is smaller because the RL agent would ostensibly be limited to acting on those labeled objects. This approach has the added benefit of using the same information a human would have, the image of the webpage, as opposed to the raw HTML of the webpage. Additionally, using the visual image of the webpage has the advantage of being generalizable to all computer usage, not just web based tasks.

This is not a novel approach, in *World of Bits* [8] the experimenters tried using a CNN to transform an image of the webpage into a feature vector. They did not use object detection, intuiting that the feature vector would suffice. In this paper, the computational cost of processing the image through an object detection model, versus the computational cost of processing the HTML into thought vectors, is not explored. The solution explored here would not be as generalizable to natural language tasks, such as buying an airline ticket, but might be sufficient in simpler tasks. The focus of the paper is on the Faster R-CNN object detection model’s ability to parse the image of a webpage for useful object classes. The efficacy of an RL agent using the object detection model is not explored.

Since labelling images of webpages is time intensive for a human, this paper additionally explores an unsupervised learning method, and a semi-supervised learning method. The unsupervised method is a meta-trainer, which annotates images of webpages with their

class labels through iterative exploration by a deterministic agent. The deterministic agent exhaustively explores the same url, recording what outcome is achieved by clicking where. Unfortunately the meta trainer was unable to provide usable data for training, due to the limitations of relying on Selenium to record the state space. For that reason we move on to explore active learning methods that prove to be more effective.

For the semi-supervised method, we explore the data diversity paradigm in active learning. We use k-means clustering to select the webpage images that provide the most new information to the model. From there, the webpage images are annotated for training.

CHAPTER 2

Methods

2.1 Meta Trainer

For the purpose of producing annotated image data, we develop a meta trainer, which exhaustively searches a website page for interactive elements. It does so by clicking a location on the webpage, and recording the change in state. The background class is taken as a section of the webpage that, when clicked, produces no result. All clicking will be performed as a left-click. Scrolling, dragging, right-clicking, and keyboard presses are not explored.

The meta trainer is built on top of two Python modules: Selenium and PyAutoGUI. Selenium is an automatable web-browser across multiple environments, including Python, Javascript, Java, and Ruby. In Python it is single threaded, meaning it executes commands in a sequential manner without involving multiple CPUs. This is in contrast to how Google Chrome normally functions when a human uses it, and greatly slows down how quickly the meta trainer can iterate. Selenium is capable of emulating Google Chrome by using ChromeDriver, which was the web driver chosen for this experiment. ChromeDriver is an implementation of WebDriver, an open source platform that implements its own W3C WebDriver standard protocol. That way, Selenium is able to interact with Google Chrome as if it were controlled by a user. Selenium has wide uses in the fields of data scraping on the internet and automated website browsing. PyAutoGUI is used to programmatically control the computers mouse and keyboard using Python, and allows the code to perform most inputs a human would be able to perform using their mouse and keyboard.

We used these two packages to orchestrate the opening of a ChromeDriver, opening the chosen webpage, clicking a specific location on that webpage, and interpreting the change in

the state of the web environment. A change in state might include: redirecting to a different url and webpage, changing the existing webpage, or opening a new tab, such as in the case of an advertisement redirect. Through this method we are able to observe the cause and effect of a click deterministically, and build a rough mapping of location to outcome. This is a computationally expensive method, as for every grid space on a webpage it requires loading a webpage, clicking, recording the result, and if the webpage has changed, reloading the original webpage.

The procedure for the meta trainer is as follows. First, instantiate a ChromeDriver and set various parameters to obfuscate the fact that it is not being controlled by a human. This is done to prevent the website from initiating a captcha test. Next, a window is opened to a starting url. For training, four different allrecpies.com urls were used in equal proportion. Then we record information on the state of the webpage including: window size, url, all the text that can be scraped from the HTML, the number of tabs open, and a screenshot of the webpage. The meta trainer will then make a query: by clicking a random location on the webpage we find if that location is enclosed by an interactive web element, such as a “log in” button, that would redirect the user to a different webpage, open up a new tab, or else change the contents of the current page. After the click has occurred, we again record the state of the webpage. If it has not changed, then clearly that location is part of the background class.

From observation of the data and meta trainer, it becomes clear that there are issues with reliability. Originally, a change in the text on the webpage was recorded as one of the parameters to define a class. However, this was found to be inconsistent, as the Selenium ChromeDriver was unable to accurately record the text on the screen, often recording text invisible to the user, that was changing as a result of network calls to the website after the website had finished loading. There were many different Selenium HTML queries attempted, and we ultimately settled on using BeautifulSoup to find all HTML elements with text, try filter out those labeled invisible, and then record that text. However, no solution to the problem of network calls occurring after the webpage had finished loading was found, leading to false positives on the background class. For this reason, only two parameters are recorded

for state change. The first is whether the url changed, and the second is whether the number of tabs changed. This results in two object classes: a redirect to a new url, and a new tab opening with a new url. Since we could not accurately record the text on the webpage changing, a click that alters the webpage without redirecting to a new url is considered the background class. Both a new tab opening and a redirect on the original tab does not occur. The redirect to a new url will always occur on the new tab, leaving the original tab as is.

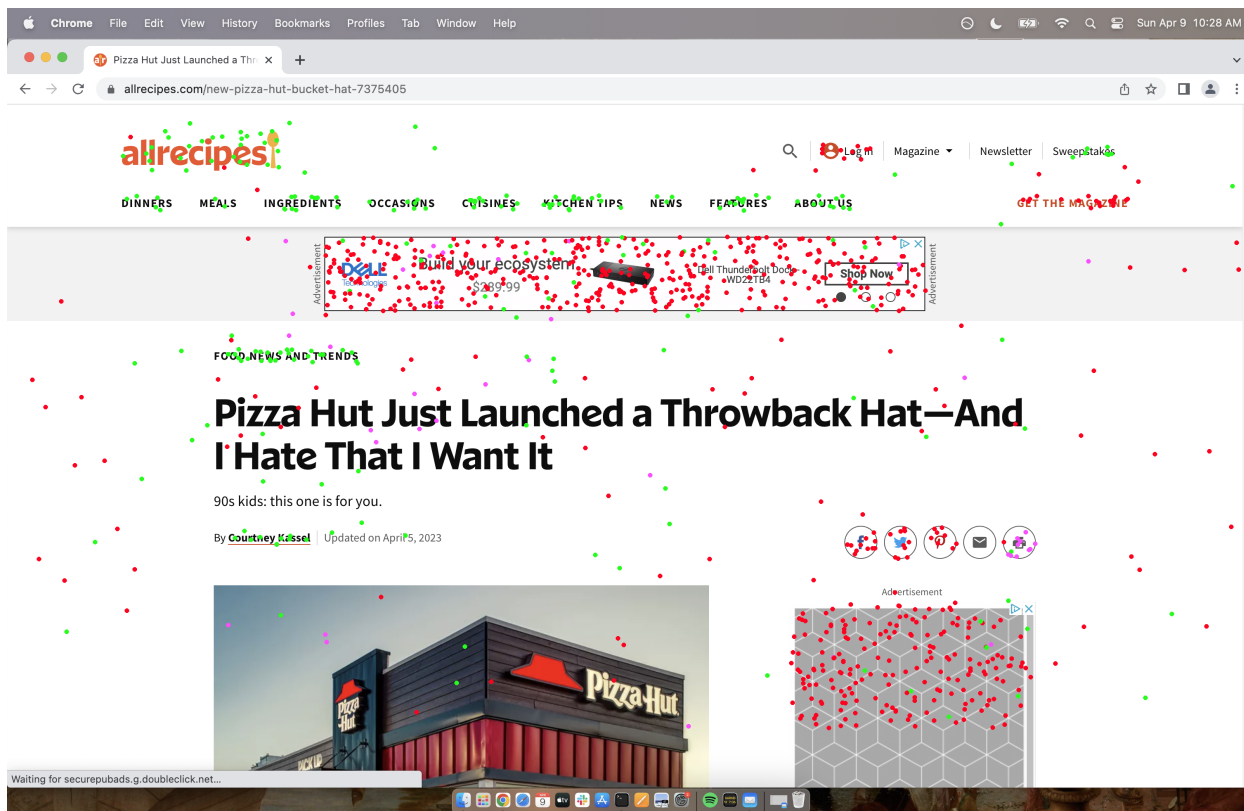


Figure 2.1: Meta Trainer mapping for <https://www.allrecipes.com/new-pizza-hut-bucket-hat-7375405>. Each dot represents a click and the associated class: green for url redirect and red for new tab opened. The background class is hidden.

By recording the location of the click, and the resulting class (redirect, redirect to new tab, background), the meta trainer constructs a map of click location to state change. This map is specific to the url, and to the way the webpage loaded. If the webpage loaded with a large ad, then the map will be different than a webpage loaded with a small ad. For this reason the starting state spaces are matched with both url and image similarity. Every time

the url changes we reset the webpage to the starting url, in order to focus the efforts of the meta trainer on a small set of urls, due to the fact that it has to exhaustively search the webpage for interactive web elements. We choose to reset the entire script every 250 iterations in order to avoid the WebDriver getting stuck in a non-informative state, such as a minimized window. Although Selenium provides a method to maximize the existing window to full, the window handle is sometimes lost in the flow of the script. A bash script was used to orchestrate the continuous resetting of the Python environment. This approach was chosen because the interaction between Selenium and PyAutoGUI did not result in consistent behavior.

Once we have a mapping of click location to result, we can try several methods to generate weakly annotated data. The desired result is a set of class-labeled bounding boxes and a representative screenshot. Since both classes are nearly always present on a webpage, simply listing the classes in the image, as is the common practice, is not sufficient. There were three approaches tested to generate bounding boxes from the click location data. In all methods, the image is divided into a grid, and each element of the grid is labelled according to its class. From these labelled grids we can generate a set of bounding boxes. Many grid sizes were explored for each method in order to determine an optimal setup.

We tried picking the most common class for a grid space, using a KDE to record density scores for both object classes in each grid space, and using a KDE to pick the densest class for each grid space. The first method we explored was to choose the most common class for each grid space. The downside to this method is that the grid spaces need to be larger in order to accommodate the relative sparsity of click data at each location. The KDE methods involved generating a Kernel Density Estimation (KDE) of the two object classes. The KDE was generated using the SciPy library in Python using a Gaussian Kernel. The input would be a single classes recorded click locations, which is a set of x,y coordinates. The Scott's rule bandwidth was used for every KDE in order to normalize the density across varying total clicks recorded, because for different urls there is a large variation in the number of objects witnessed. Creating a KDE for the background class was explored, but found to be ineffective. It is more effective to place the background class in any grid with a low density

of the object classes. Once we have the KDE for both object classes, we have a choice in how to interpret the two KDEs. The smoothing method involves keeping both class scores to yield a labelled grid space like $(0.7 \times 10^{-6}, 0.4 \times 10^{-6})$ where the density of both classes is recorded. The max pooling method picks the class with the largest density to yield a grid space like $(1, 0)$. In these approaches we independently optimize a threshold below which the background class is chosen for that grid space.

2.2 Faster R-CNN derived classifier

We used the Faster R-CNN ResNet-50 FPN V2 object detection model from *Benchmarking Detection Transfer Learning with Vision Transformers* [12]. The Faster R-CNN ResNet-50 FPN V2 uses the ResNet-50 backbone network for feature extraction, as opposed to the original Faster R-CNN model which had used the VGG16 backbone. This results in performance increases while still maintaining the same conceptual design.

The model is sourced from TorchVision, the computer vision section of the PyTorch library, which is a popular end-to-end machine learning framework in Python, known for its applications in deep learning. For the purposes of transfer learning, we used the pre-trained weights provided by TorchVision, which were trained on the Common Objects in Context (COCO) dataset [12]. The COCO dataset comprises over 200 thousand labeled images, and is a common resource for computer vision training. The outline of the network is as follows: a transformation of the image to a smaller size, along with normalization, then the ResNet-50 backbone, then the RPN layers, and finally the RoI heads.

From there, we reset the output layer box predictor weights, and resized it to fit our expected number of classes (3 with background). Since the box predictor is the last fully connected layer of the model, it is just two parallel fully connected layers, one for box locations and the other for class labels. This box predictor is given the cropped feature maps after one last convolutional layer and a fully connected output from the box head, so the image has already been fully interpreted by the RoI layers, the RPN layers, and the backbone layers, all trained on the COCO dataset. Therefore, by resetting these weights,

the network unlearns the classes from the COCO dataset, while keeping the desired feature extraction.

2.3 Initial Training

The default transformation of the input images for the Faster R-CNN ResNet-50 FPN V2 object detection model is composed of several steps. The input images are 3024×1964 pixels with 3 channels for RGB. The numerical values of the pixels range between 0 and 255, so the first step is to normalize the pixels to have a mean of 0.485, 0.456, and 0.406 respectively for the three channels, with a standard deviation of 0.229, 0.224, and 0.225. This effectively changes the colors. Next, the image is resized. For the default transformation of the model from Torchvision, the input image of 3024 by 1964 is rescaled to be 1248 by 800. This results in an image that looks like the following, which is what the model sees.

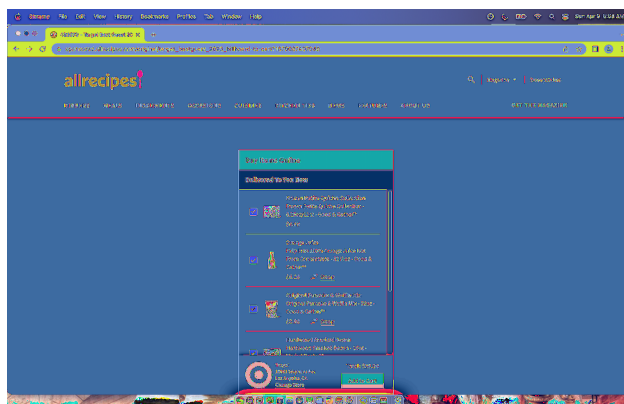


Figure 2.2: Input image after default transformation including pixel normalization and image resizing to 1248 x 800.

The pixel normalization was left as is, but various resizings of the input image were explored. The default resizing from 3024×1964 to 1333×865 was evaluated alongside resizing to 640×480 , resizing to 400×400 , and resizing to 244×244 . For each resizing, multiple approaches were attempted with regard to what layers were frozen. The best training performance was recorded for each resizing.

During the training process, we enable Stochastic Gradient Descent (SGD) optimization.

As we are conducting transfer learning, it is common to only train certain layers. Here we explore three transfer training methods.

- Freezing the ResNet-50 backbone and the RPN, so that only the fully connected classification layers are learning
- Freezing the ResNet-50 backbone, so that the RPN is also included in the learning
- Allowing the entire network to learn, including all feature extraction layers.

Since the website imagery is so different than the COCO dataset, training the RPN might be useful in reconfiguring how the object regions are proposed, and training the backbone might be helpful to focus more on visual features relevant to a website. In all transfer learning methods the RoI heads are subject to training. The RoI heads is a series of convolutional pooling layers, followed by a fully connected linear output layer, and finally the box predictor with its randomly initialized weights. The box predictor will benefit the most from the training, as it has randomly initialized weights, but for the rest of the available network, the training will transfer the learning from the COCO dataset to the problem space of website images outlined in this paper. We used the SGD optimizer from TorchVision with a 1×10^{-3} learning rate and 0.9 momentum. The initial training consisted of only 8 labelled images. All initial training was done on *allrecipes.com* for the purpose of testing the model’s ability to test on other website’s webpages without it having encountered those pages before. Further training is explored in the active learning section, where we found the most beneficial examples to annotate.

2.4 Active Learning

After the initial training with labelled data sourced from allrecipes.com, we proceed to incorporate screenshots of other websites, chosen by the active learning methodology of diversity sampling. The websites chosen were sourced from an article on the most visited websites of 2023: explodingtopics.com/blog/most-visited-websites. For some websites, more

than one screenshot was taken, including some screenshots that include a captcha, which was most likely triggered from the behavior of the python driven chrome instance upon visiting the website. These were included in the active learning sample because of the high likelihood an RL agent would encounter them in real use. Below is a table of the websites included.

Active Learning Screenshots
Amazon
Fandom
Grubhub
TikTok
Twitch
Wikipedia
YouTube

Table 2.1: The websites screenshots used for the active learning component

As mentioned in the introduction, screenshots for the above websites were chosen for annotation using the data diversity approach to active learning with K-Means clustering. The 3-channel RGB matrices of pixel values from the screenshots were flattened into one long vector, where the first width x height values were all one channel, followed by the next two channels. We used the K-Means clustering algorithm from the scikit-learn library in Python. The algorithm was initialized with $k = 4$ clusters, however, that resulted in a cluster of just one image, and a second cluster with only two images. The purpose of the data diversity approach is to select images most representative of a cluster, so a single value cluster does not yield as much meaning. Additionally, for the two image cluster, the images are equally close to the centroid by definition, resulting in a cluster where neither value can be considered the most representative. Therefore we settled on 3 clusters for the data.

Since one of the three clusters is entirely composed of allrecipes.com screenshots from the initial training, we omit that cluster from the active learning portion as it has already been trained on. It was necessary to include those examples in the clustering in order to differentiate the new data from data the model had already been trained on. Therefore the

active learning portion will consist of training on two selected images, the ones closest to the cluster center for clusters 2 and 3. A control model will be trained on the two images furthest from the 3rd cluster center, as that would be the worst choice of training data according to the data diversity methodology. Both the active learning model and control model will be derived from whichever object detection model scores the best in initial training. The two models will be compared using two validation images, one from cluster 2 and one from cluster 3, which are omitted from any training data.

Cluster 0 was entirely made up of the initial training allrecipes.com screenshots, which are already labelled. Cluster 1 is Twitch and Fandom, which both have dark and colorful UIs with many separately boxed elements. And cluster 2 is made up of white backgrounds and sparse UIs. The representative images closest to the centroid are presented below, alongside the exact centroids.

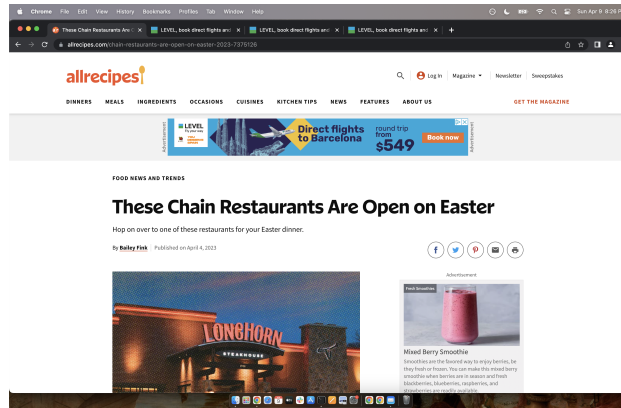
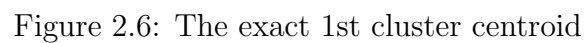


Figure 2.3: Screenshot closest to the 0th cluster centroid. Sourced from www.allrecipes.com



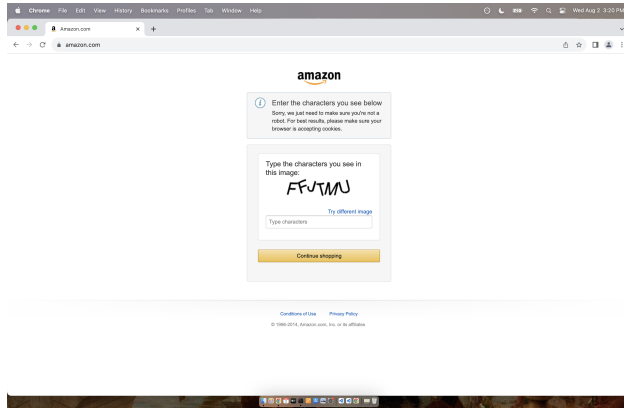


Figure 2.7: Screenshot closest to the 2nd cluster centroid. Sourced from www.amazon.com

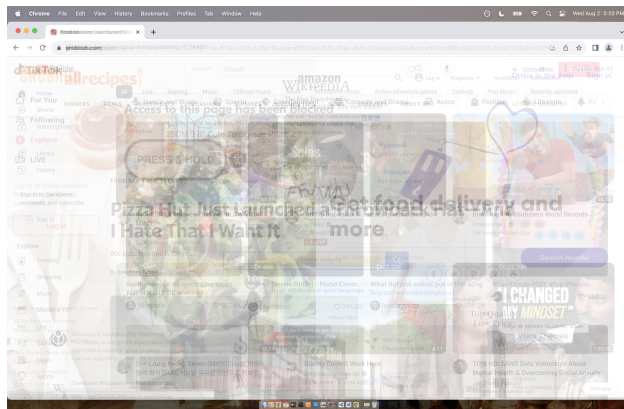


Figure 2.8: The exact 2nd cluster centroid

CHAPTER 3

Analysis

3.1 Meta Trainer

The meta trainer was unable to produce data for use in training. Given the amount of time that went into developing the meta trainer, evaluating its production, and synthesizing its output, that leaves us with very little labeled data to work off of. The inclusion of active learning in this paper begun after it became clear that the concept behind the meta trainer was flawed, and may never produce usable data. The fundamental problem with the meta trainer was the reliance on the Selenium ChromeDriver to understand the cause and effect relationships of the inputs given by the PyAutoGUI controller. Selenium was never designed to be used in conjunction with the actual mouse and keyboard. It has its own internal methods for clicking on elements, so when PyAutoGUI would perform a click using the computer itself, Selenium’s understanding of what had occurred was lost.

The result is that Selenium could not give accurate classifications. It often incorrectly determined that no redirect had occurred, when a redirect had occurred. It additionally could not differentiate between the tab that was visible to the screen of the computer, and the tab it was currently looking at in order to determine the state of the RL agent. Additionally, the data it recorded needed heavy cleaning because of the frequency with which it would lose the window handle to a window that had been closed by itself. The fundamental issue was that the meta trainer was using Selenium for purposes it had not been intended. It would have been more pragmatic to use an image classification model to classify the change in state between two images. An Optical Character Recognition (OCR) model could be used to determine the url of each image, and a classification model that is mostly looking at image

distance could ascertain whether the webpage had changed between two images.

To understand the problems with the meta trainer, it is most illustrative to look at examples. The following are images from the same url that have first been processed to remove clearly incorrect data. The images were first stratified by url, then by image distance from each other, as there are multiple configurations of the webpage for each url. An example of a differentiating factor for the same url is the size of the advertisement. Next, the KDE smoothing is applied to get a density estimate for each class individually. Finally, we stack the class density estimates and either perform max pooling or smooth pooling for each grid square as discussed in the methods. First we view an example where the meta trainer creates an example quite usable in weak learning. However, this was not a common result. Given that the meta trainer is creating these training examples through trial and error, it takes too much time in order to generate one example for every 10 examples that do not give usable information. Next are some examples of failed attempts, note that the same KDE parameters were used. See figures 3.1, 3.2, and 3.3.

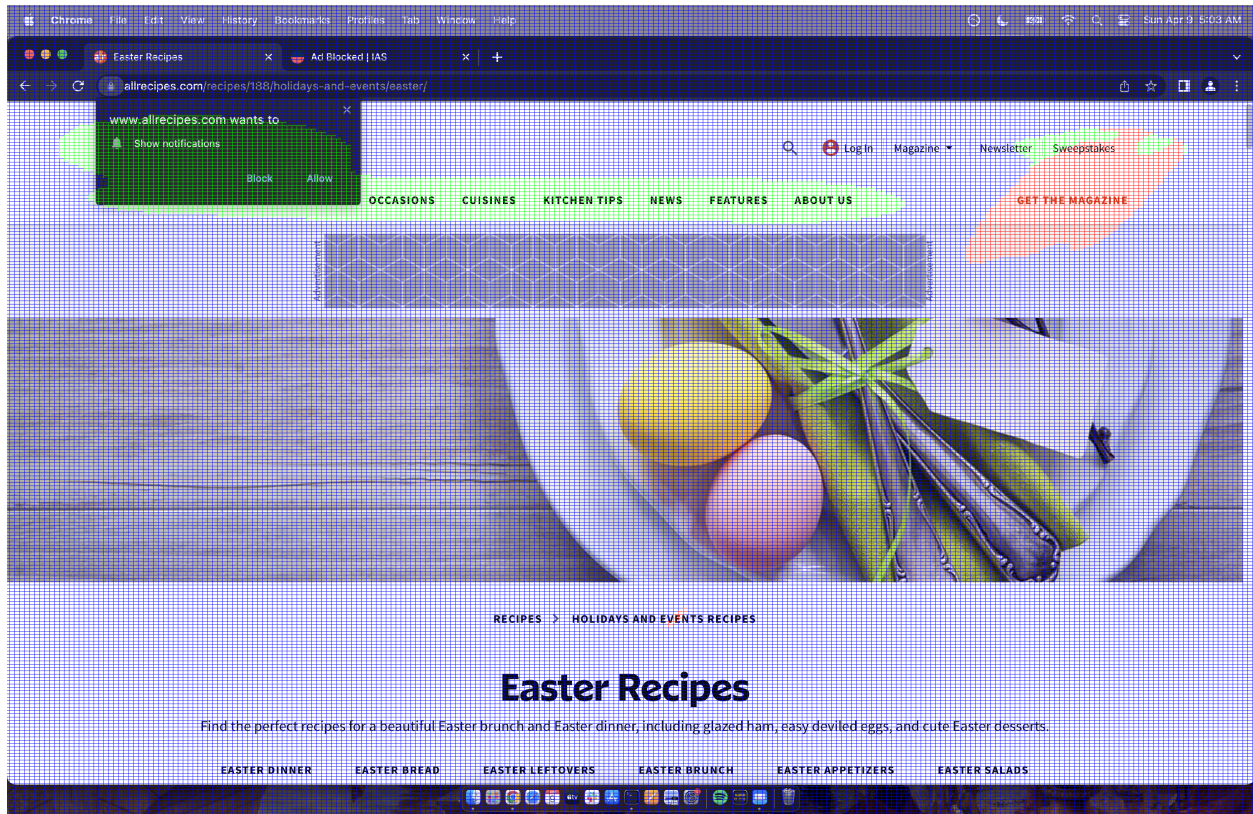


Figure 3.1: Max Pooling KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels are decent, but there are missing labels towards the bottom of the image. The notifications popup in this screenshot demonstrates the some of the variation the meta trainer experienced on the same url. Sourced from www.allrecipes.com

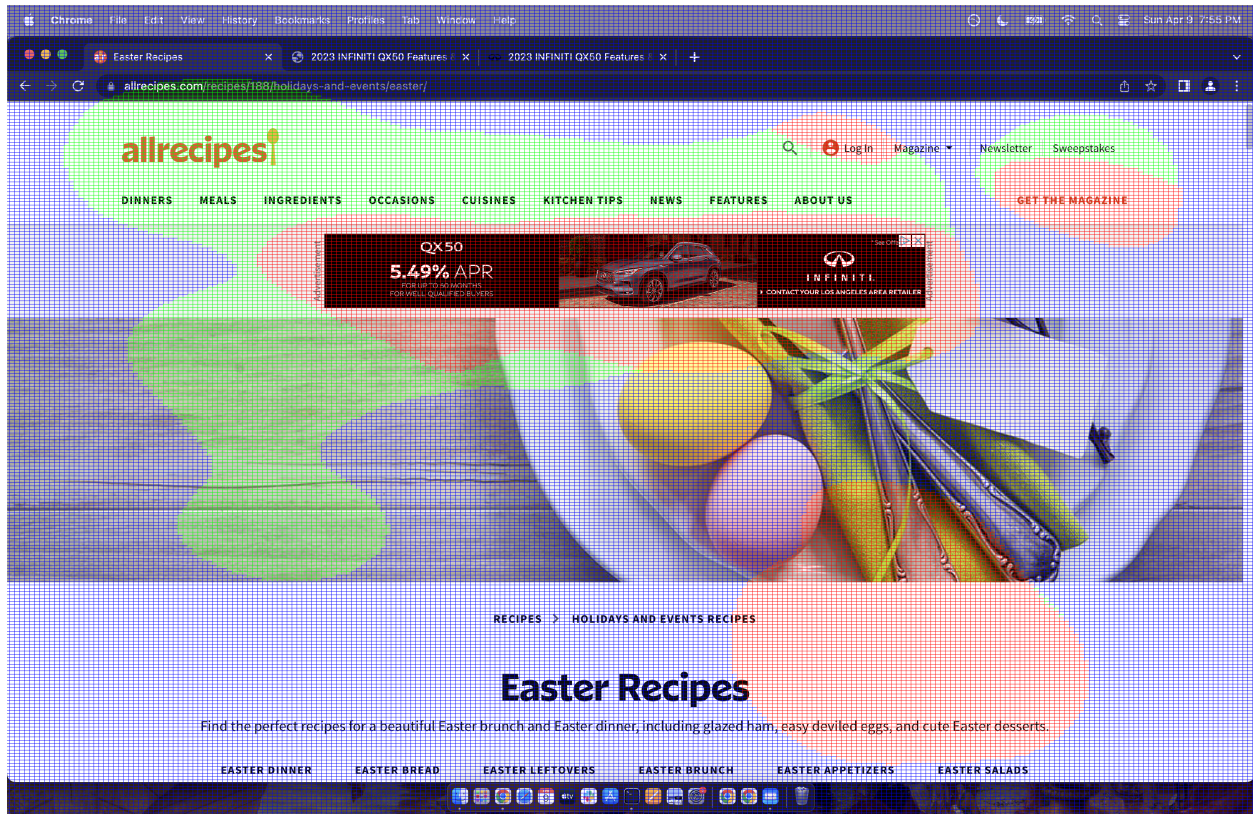


Figure 3.2: Max Pooling KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels over predict objectness of the image, and do not center the class labels towards class locations. Sourced from www.allrecipes.com

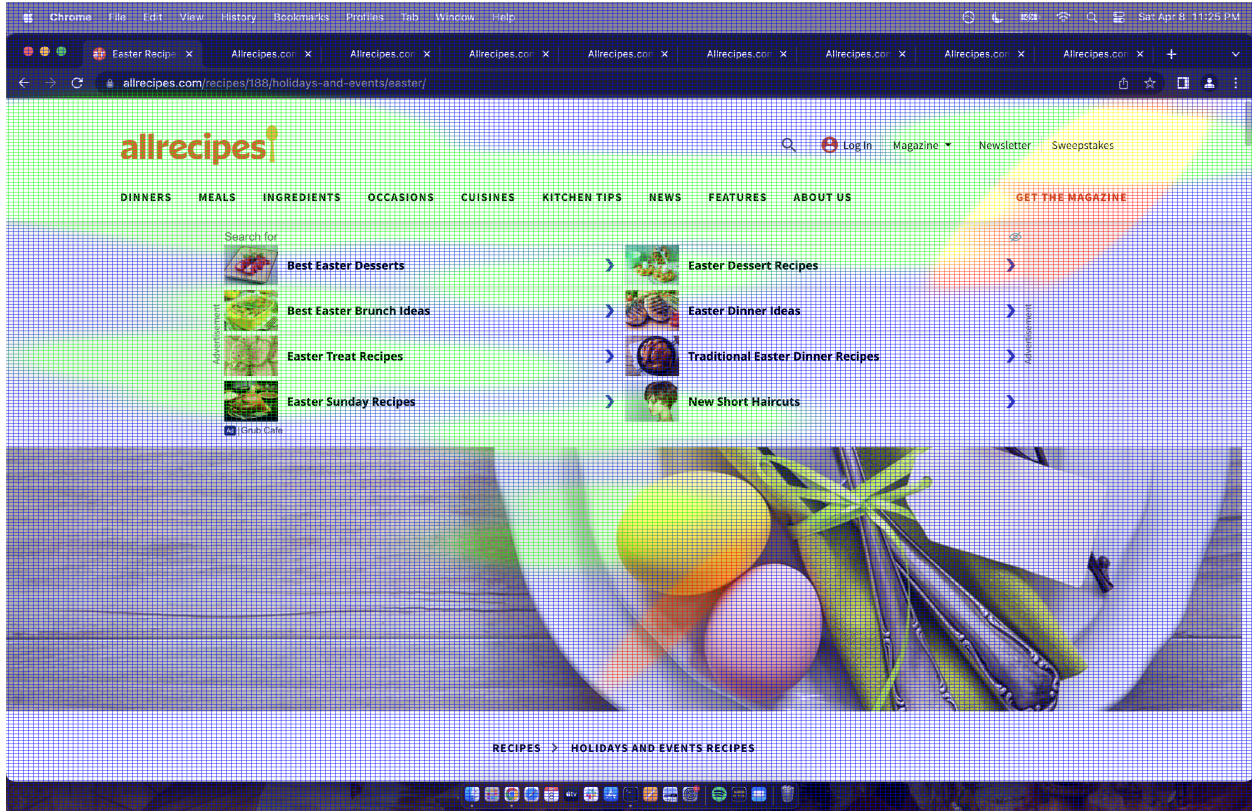


Figure 3.3: Smooth KDE estimation of classes. Blue is background, red is new tab and redirect, green is redirect. The labels inconsistently predict objects in the center of the image, and do not correctly predict the orientation of objects. Sourced from www.allrecipes.com

3.2 Resizing the Image

Despite the fact that the default transformation should be the most relevant to a pre-trained network, it was not in fact the most effective transformation of the input image. Given that the website images are quite different than the COCO dataset, it is possible that something about smaller images lend to better feature extraction and region proposal. The most effective resizing was to 224 x 224. Perhaps the ResNet-50 backbone is benefiting from this sizing, though it is unclear why the default transformation would not match 224 if that is the case. The Torchvision documentation does not mention the image size the model was trained on, but discussions in the Torchvision forum often recommend using a 224 x 224

transform, indicating there is some consensus that this resizing would yield the best results.

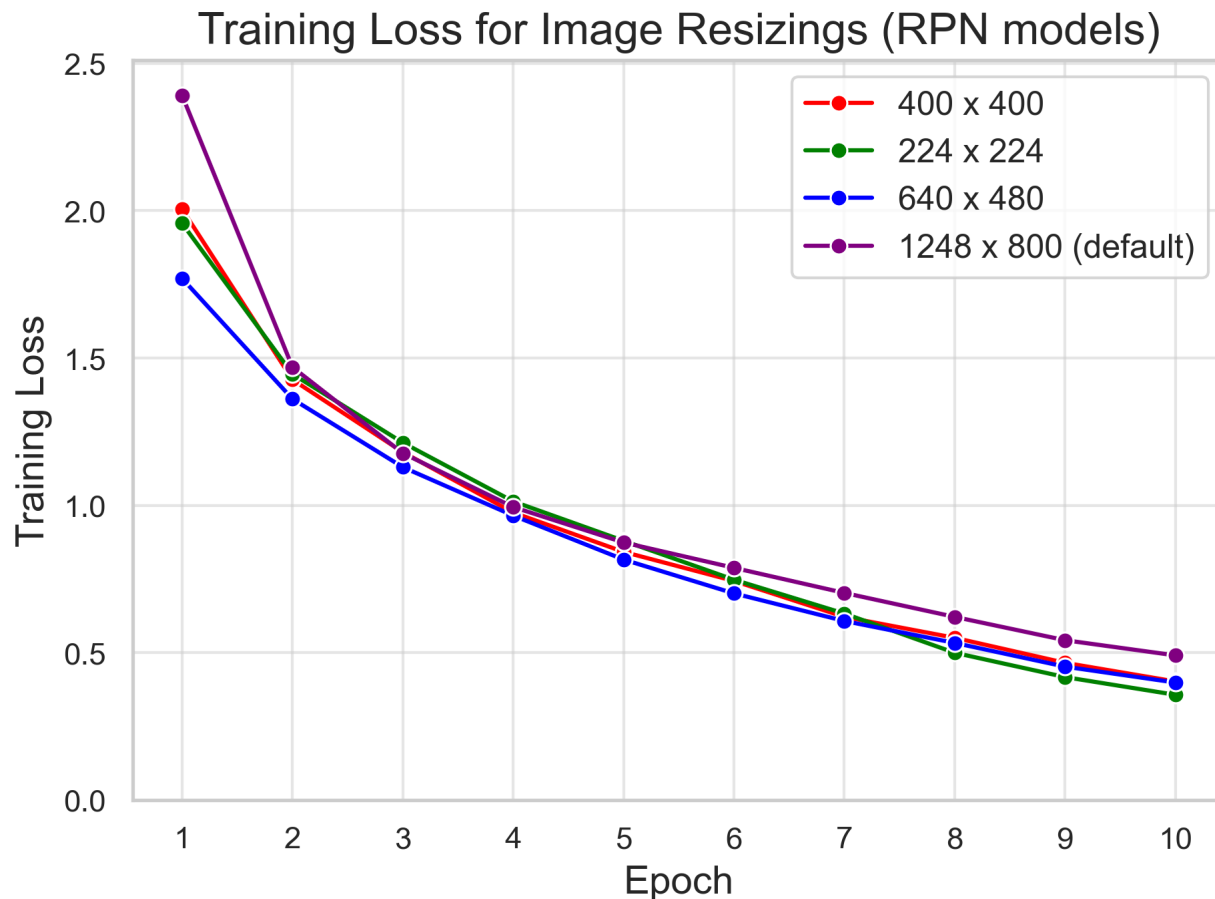


Figure 3.4: Training loss over 15 epochs with different initial image resizing transformations.

Since 224 x 224 is the best resizing in both training accuracy and computational cost, all further analysis assumes the model is trained and tested with this transformation.

3.3 What Layers to Train

The least effective transfer learning strategy was to not freeze any of the pre-trained layers. It appears that training the feature extraction layers is detrimental to model learning. The model appears to have been able to generalize the feature extraction from COCO objects to web element objects. From the initial training of 10 epochs, it is not clear whether the model benefits from training the RPN layers. However, it makes sense to train more layers

if possible, and since the objects in this data are quite different than the COCO objects, changing the region proposal layers makes logical sense.

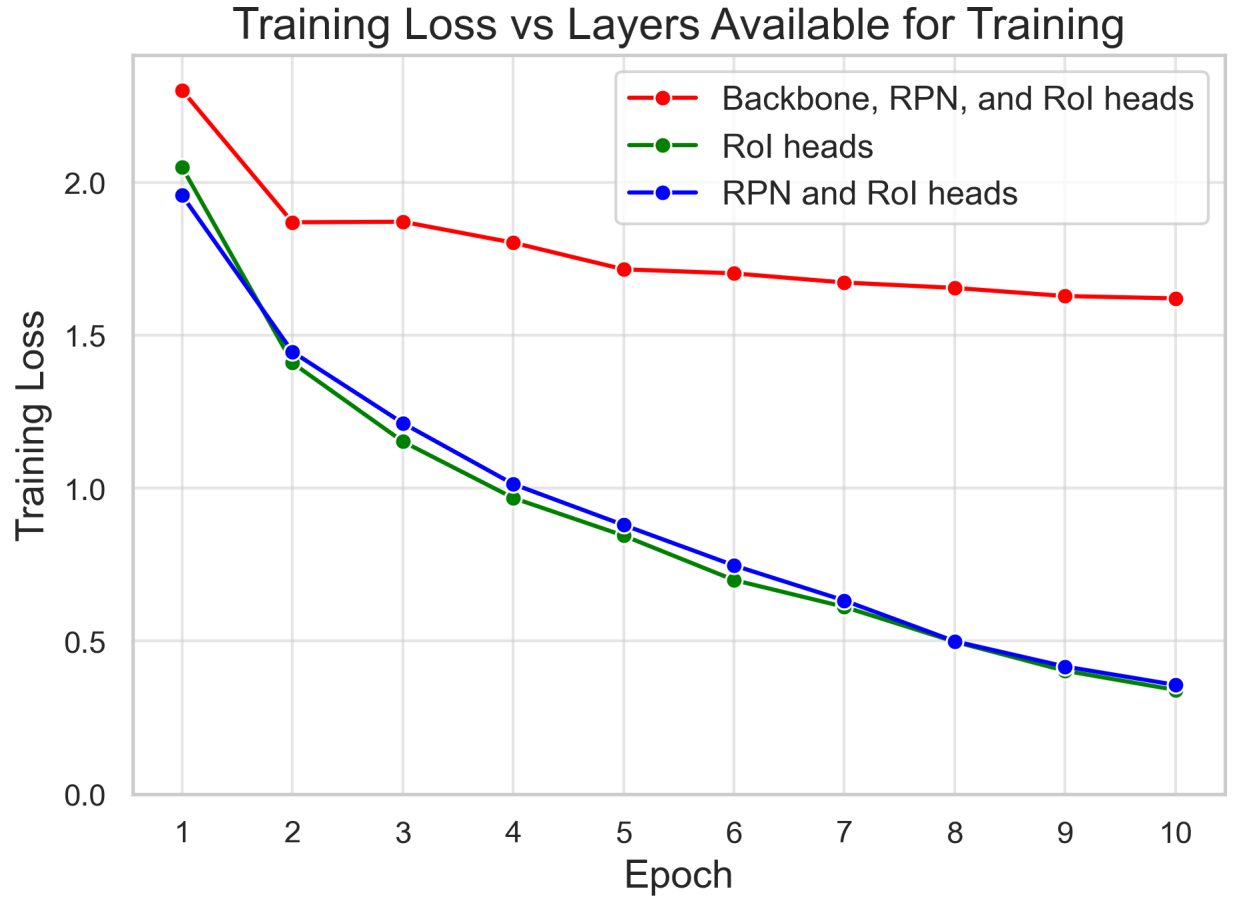


Figure 3.5: Training loss over 10 epochs with different model parameters subject to training. For all models, the RoI heads are trained.

Then it has been determined that the best model was trained with 224 x 224 resizing and had the RoI and RPN layers available for training. It is not clear how much training on the initial samples will lead to overfitting, so we explore multiple starting points for active learning. We start the active learning training with up to 50 epochs trained on the initial data. That way we can get the most information out of our limited samples without overfitting. Since there are so few examples to train on, the model will suffer regardless from overfitting and an inability to generalize. We can mitigate those issues by exploring a larger parameter space now that we have found the best model.

3.4 Active Learning

The diversity sampling determined that the Amazon captcha page and the Twitch home page were the most representative images to train on. We compare this active learning set against a model trained with the least representative samples found with the same method: the TikTok and YouTube homepages. We will refer to the latter as the control set. For both sets of training data, we use the same validation data: the Wikipedia homepage and the Fandom homepage. The models were trained for 20 epochs, and generally had their best validation score at around 10 epochs. The best score for the models, the epoch they scored it, and the number of epochs they were trained on the starting data is displayed below.

Table 3.1: Learning Dataset Summary - Part 1

Learning Dataset	Best Initial Epochs	Number of Epochs Trained
Active	30	20
Control	50	20

Table 3.2: Learning Dataset Summary - Part 2

Learning Dataset	Epoch with Best Validation	Best Validation Score	Initial Validation Score
Active	11	3.56	4.19
Control	1	4.07	4.07

For the active learning dataset, the optimal initial training was 30 epochs, whereas for the control set it was 50 epochs. The best active learning model reached its optimal validation score halfway through the 20 epoch training, whereas the control learner never improved upon its initial validation score. The models trained on the active learning data performed much better on the validation data. This is despite the fact that, qualitatively, the Fandom homepage, which is from the validation set, is very similar to the YouTube homepage, which is from the control set and not the active learning set. It may appear that the active learning training only scores slightly better on the validation, but the difference is the active learning dataset improved the validation score, while the control dataset did not. That means that given such a small dataset, the active learning examples gave the model the ability to score

at all on the validation set, whereas the models given the control data were only able to memorize the objects on their dataset. The k-means clustering approach to data diversity would say that the active learning dataset provided more pixel variance than the control set. It appears that the variance in colors and the variance in the location of features gave the model the generalizability it needed to begin making worthwhile predictions on the validation set. However, given such a small training set, the generalizability of the model is limited. To illustrate this, we freeze the best active learning model at its best validation score, and visualize its prediction on the Wikipedia validation image.

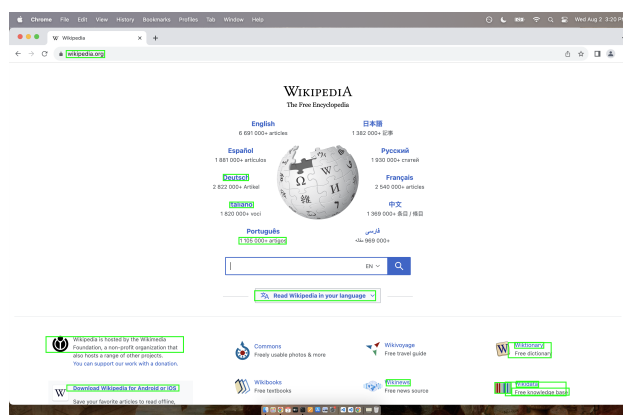


Figure 3.6: Best model predictions on the Wikipedia validation image. This model was trained for 11 epochs on the active learning dataset, after 30 epochs of training on the initial dataset. Sourced from www.wikipedia.org

The best models predictions on the fandom homepage image are not included, because the model made no predictions. It did not detect any objects on that page. Perhaps something about the dark background was too unfamiliar for the model. This suggests that the best model’s generalizability is limited to certain types of images.

Recall that we began this experiment with a pre-trained network on the COCO dataset, and performed transfer learning during the entire experiment. Below is a table summarizing how the model’s weights have changed over the course of training. Earlier, we had found through testing that the best model only had the RoI heads and RPN available for training, as the feature extraction layers were adequately trained for task. Therefore we froze the backbone feature extraction layers, so those vector norms are 0. The box head classification

Table 3.3: Model Component Analysis

Model Component	Aggregate Vector Norm of Weight Difference	Number of Layers	Average Vector
backbone body	0.000	122	0.000
RPN	0.366	4	0.092
RoI heads	6.030	11	0.548

and bounding box regression layers were initialized with new sizes and weights for this experiment, since the object classes are different for this experiment. For that reason, the difference in those weights is understandably the largest, as the weights were randomly initialized at the start of the experiment, when we replaced the box heads. Those layers are within the RoI heads component.

The best network was trained using the active learning dataset, and performed 12.5% better on the validation data than the control. Surprisingly, it was better to train on the initial data for 30 epochs than it was to train for a more conservative number of epochs, such as 10. The model was able to generalize to the Wikipedia image that it had not trained on, but not the Fandom image. It is clear that transfer learning for object detection in website images is effective, but a dataset would have to be much larger than 10 images. Unfortunately the meta trainer was not capable of supplying that data.

CHAPTER 4

Conclusion

In a deterministic environment like a webpage, using a meta trainer to create labelled data is beneficial and entirely possible. However, the human interpretation to changes in state is not always aligned with how the computer interprets the state space, so it is necessary to develop tools to record state changes in a reliable manner. This project would have benefited from using an image classification model to interpret state change between one image to the next, as opposed to using Selenium. Paired with the click locations, one could generate labelled data. Once the object detection model had improved to the point of making reasonable predictions, it could be used to define what click locations are to be explored by the meta trainer, as was the hope for this project.

Even without a large dataset, the images of webpages were well understood by the object detection model, despite the fact it had been pre-trained on real photographs. Given that website designers are employed to make navigating a website intuitive, this is not a big surprise. Additionally, even though the images in the COCO dataset are quite different, it is a reasonable argument to say that a website provides comparably hard edges with high contrast. Therefore the object detection model was quickly able to learn the 3 class problem posed in this experiment.

Active learning methodologies proved to be able to differentiate between high quality training data and low quality training data. The model trained on the images recommended by active learning proved to be able to generalize in some cases, whereas the model trained on the leftover image data was not. However, it remains the case that the deep learning paradigm for object detection, which Faster R-CNN is a member of [5], requires numerous training examples to generalize and avoid overfitting. This was evident as our best model

made no object predictions for the Fandom.com homepage, which has a similar UI to an image the model had already trained on, that of the Twitch.tv homepage. These two images were part of the same k-means cluster, but the Faster R-CNN model did not find enough support to generate high objectness scores, proving it has yet to learn the problem space.

CHAPTER 5

References

1. Yang, B., Sun, J.-T., Wang, T., & Chen, Z. *Effective Multi-Label Active Learning for Text Classification*. *Microsoft Research Asia*, 49, 2009.
2. Yang, C., Huang, L., & Crowley, E. J. *Plug and Play Active Learning for Object Detection*. *ArXiv*, abs/2211.11612, 2022.
3. Vo, H. V., Siméoni, O., Gidaris, S., Bursuc, A., Pérez, P., & Ponce, J. *Active Learning Strategies for Weakly-Supervised Object Detection*. *arXiv*, 2207.12112, 2022.
4. Desai, S. V., Chandra, A. L., Guo, W., Ninomiya, S., & Balasubramanian, V. N. *An Adaptive Supervision Framework for Active Learning in Object Detection*. *arXiv*, 1908.02454, 2019.
5. Ren, S., He, K., Girshick, R., & Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. *arXiv*, 1506.01497, 2016.
6. Girshick, R., Donahue, J., Darrell, T., & Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation*. *arXiv*, 1311.2524, 2014.
7. Girshick, R. *Fast R-CNN*. *arXiv*, 1504.08083, 2015.
8. Shi, T., Karpathy, A., Fan, L., Hernandez, J., & Liang, P. *World of Bits: An Open-Domain Platform for Web-Based Agents*. *Proceedings of Machine Learning Research*, 70, 2017.
9. Balasubramaniam, A., & Pasricha, S. *Object Detection in Autonomous Vehicles: Status and Open Challenges*. *arXiv*, 2201.07706, 2022.
10. Brust, C.-A., Käding, C., & Denzler, J. *Active Learning for Deep Object Detection*. *arXiv*, 1809.09875, 2018.

11. Zhdanov, F. *Diverse mini-batch Active Learning*. *JarXiv*, 1901.05954, 2019.
12. Li, Y., Xie, S., Chen, X., Dollár, P., He, K., & Girshick, R. *Benchmarking Detection Transfer Learning with Vision Transformers*. *arXiv*, 2111.11429, 2021.
13. Gur, I., Rueckert, U., Faust, A., & Hakkani-Tur, D. *Learning to Navigate the Web*. *ICLR*, 2019,.
14. “ChromeDriver - WebDriver for Chrome.” ChromeDriver - WebDriver for Chrome, <https://chromedriver.chromium.org>
15. “Selenium”, <https://www.selenium.dev/>.
16. asweigart. “GitHub - Asweigart/Pyautogui: A Cross-Platform GUI Automation Python Module for Human Beings. Used to Programmatically Control the Mouse & Keyboard.” GitHub, <https://github.com/asweigart/pyautogui>.
17. “COCO - Common Objects in Context”, <https://cocodataset.org/>.
18. Howarth, Josh. “Most Visited Websites In The World (August 2023).” Exploding Topics, Exploding Topics, 7 Apr. 2023, <https://explodingtopics.com/blog/most-visited-websites>.