



# Logo Detection and Classification

## Duke MIDS & Proofpoint Inc.

**Altamash Rafiq, Vishaal Venkatesh, Varun Prasad**

*Gross Hall, 140 Science Drive, Durham, NC, 27708*

April 24, 2021

# 1. Abstract

Phishing is a widespread and dangerous method of social engineering designed to manipulate users into providing personal or confidential information through a malicious web page or email. Many phishing emails and web pages utilize brand logos in order to appear more legitimate. Therefore, accurately detecting logos on these sites and identifying the impersonated brand is significant for additional threat protection. In this project, we built a logo detection and classification tool that will better aid our client, Proofpoint Inc, in combating brand impersonation in phishing attacks. We implemented state-of-the-art deep learning models to both detect logos on webpages and classify the brand to which they belong. We used Faster R-CNN, YOLO v3, and an ensemble of these models for logo detection and an EfficientNetB3 for logo classification. In addition, we set up a K-Nearest Neighbors based open set learning system on top of our classifier that distinguishes observed and unobserved logo classes, thus making our full pipeline deployable in an open set setting. To quantify detection performance given our open set learning system, we also developed an adjusted precision metric. At an IoU of 0.5, our best ensemble detector achieved an open-set adjusted validation set precision of 0.93 and test set precision of 0.87 at a fixed recall of 0.87, while our best classifier achieved a test set precision of 0.98 and a recall of 0.94.

# 2. Introduction

## 2.1 Overview

Phishing is a highly dangerous method of social engineering used to deceive individuals into revealing personal and confidential information. It is also one of the most common types of cyberattacks and caused approximately 93% of data breaches in 2017 **[1]**. A phishing email or message often contains a link to an external website or login page that appears legitimate but is used to steal people's private information. Many of these phishing emails deceive users by requiring them to respond at the risk of having their account terminated. In order to further feign legitimacy, many attackers employ logos of major brands, such as banks and universities, in their phishing emails and websites. An example is shown in Figure 1. Impersonating a trustworthy brand by using its easily identifiable logo gives significantly more credence to the phishing message, putting customers of these brands at even more risk. Thus, effectively detecting logos on websites and emails, especially those that may be slightly distorted or modified, is highly valuable in identifying potential brand impersonation. Doing this at scale is challenging and our goal is to develop a pipeline that automates this process. This model would ultimately be paired with other tools that identify the legitimacy of the website. Collectively, these tools will determine whether a page is malicious and also help identify the brand being impersonated.


**Figure 1.** A phishing email impersonating Costco **[2]**. The Costco logo is clearly visible and adds more validity to the attack. Note that the phishing Costco logo is slightly different from the true one. Our model is not intended to determine whether the logo used in the phishing attempt is

real or fake, but we simply want to detect it as a logo and determine the brand to which it belongs.

The exact notion of what constitutes a logo can be abstract. Other objects commonly used in image classification problems, such as animals, faces, and cars, are well-defined. Logos, however, can vary greatly from being text-like to something uniquely artistic and picturesque. This variation is demonstrated in Figure 2.


**Figure 2.** A logo belonging to JPMorgan Chase & Co. and the automobile company Alfa Romeo. Notice how the former is text-like while the latter has many visual and graphical elements.

The abstract definition of a logo adds an additional level of complexity to our project. For the purpose of this project we have defined a logo as an object that represents a company, university, government organisation, or any other establishment. Other graphics such as icons do not constitute logos (Figure 3) because they do not uniquely represent a particular brand. However, because they may look visually similar to a logo, they easily be misconstrued as logos by an improperly trained machine learning algorithm.

**Figure 3.** Icons representing a shopping cart and WiFi. These were not considered to be logos in our project.

In this project, we aim to use deep learning models to both detect and identify logos on emails, websites, and documents. However, there is not a readily available dataset of web pages and documents with clearly annotated logos. Thus, in addition to creating these robust models, a significant portion of our project requires scraping and annotating images of websites, emails, and documents. These models also must tackle the problem of open-set learning, meaning that they need to be able to recognize logos they have not seen in training and explicitly classify them as "unknown." The work presented in this report, which details both curating the annotated dataset and developing robust models, will be highly valuable in helping our client's goal of protecting customers against cybercriminals. To provide additional context for the methods used in this report, we detail some of the state-of-the-art techniques used in object detection, image classification, and open-set learning in the following sections.

## 2.2 Object Detection

Accurately detecting logos requires us to use state-of-the-art techniques in computer vision and deep learning. The two most commonly used methods in literature are Region-based Convolutional Neural Networks (RCNN) and You Only Look Once (YOLO). We briefly describe these models in the following sections.

### 2.2.1 Faster RCNN

The first versions of RCNNs came out in 2014 but we use the more recent and state-of-the-art Faster RCNN that came out in 2016 **[3]**. Faster RCNN offers great performance and also has a myriad of open-source, well-maintained code repositories for us to use. Faster RCNN is a two-stage process. The first stage is called a Region Proposal Network (RPN) and is responsible for providing initial guesses on the potential location of the object. The RPN is pictured in Figure 4 **[3]**. Note that at this stage we are only concerned about detecting the object and not classifying it. In other words, we are performing a binary classification.

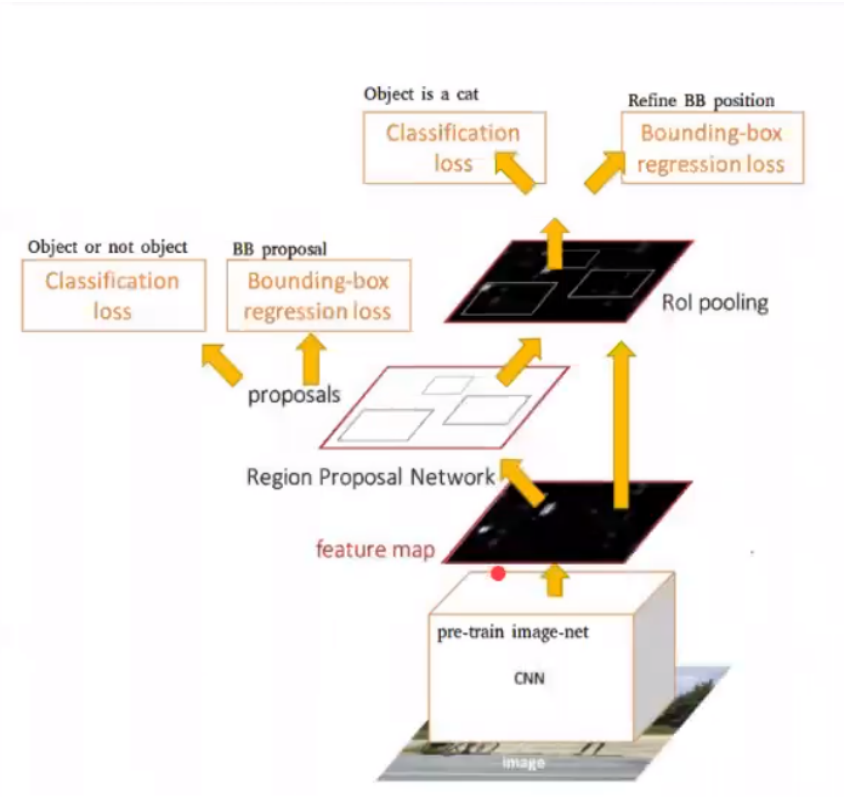


**Figure 4.** Region Proposal Network (RPN) **[3]**

The RPN works off of the feature maps produced from passing the raw images through a CNN. This CNN could have any architecture like VGG-16, ResNet 101, Inception, etc. The goal in this first step is to simply extract high-level features from the image and use these features to produce initial guesses/proposals on where the object of concern lies. In order to arrive at the initial guesses, the RPN makes use of a technique called anchor boxes **[3]**. Anchor boxes are boxes of varying sizes and aspect ratios positioned on each pixel of the feature map. The number of anchor boxes is set as a hyperparameter by the user, and the goal of the RPN is to prune these anchor boxes to keep only those that have some degree of overlap with the object of concern. In Figure 5 **[3]**, 9 anchor boxes have been drawn on a single pixel. This process of drawing anchor boxes and pruning them is done across all the pixels on the image.

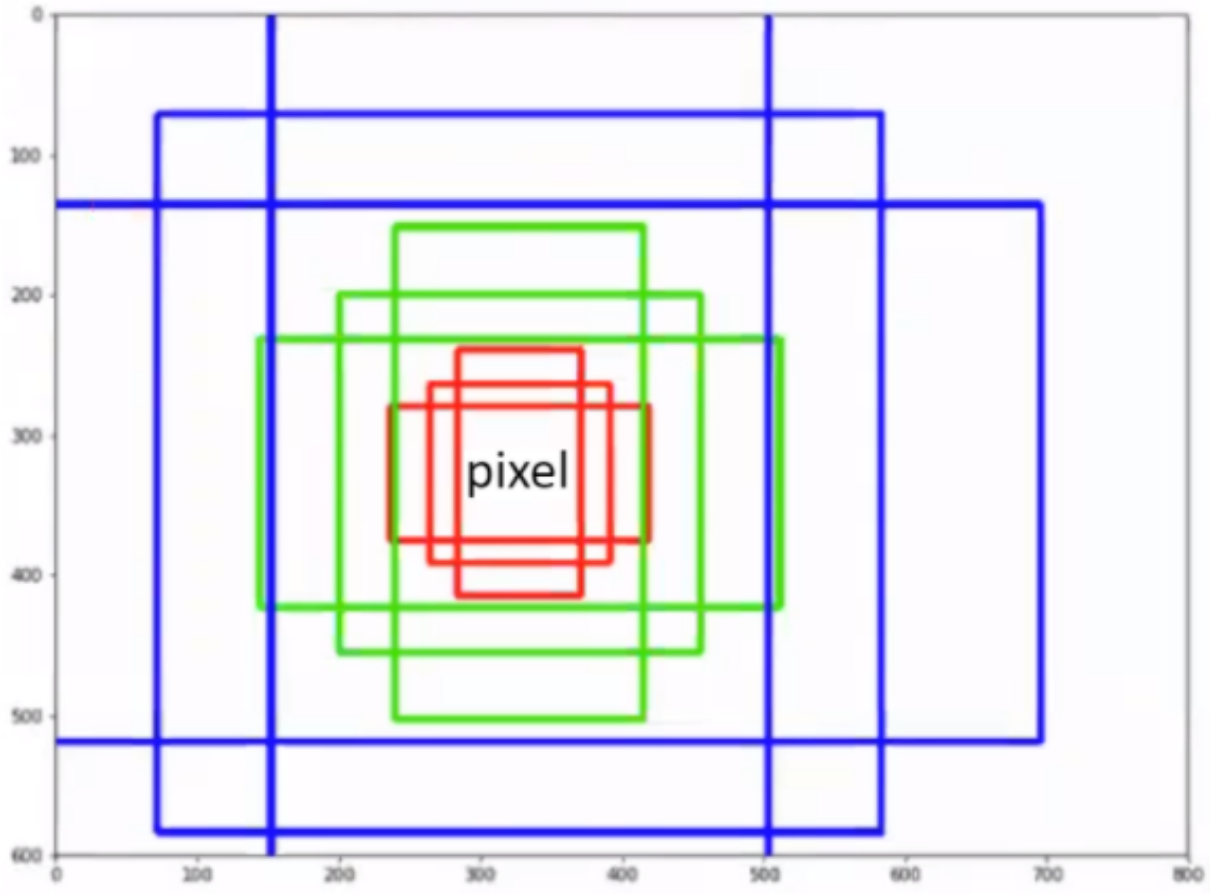


**Figure 5.** Nine anchor boxes centred around a single pixel **[3]**

While having many anchor boxes for each pixel in the image works well, it can be computationally expensive for high-resolution images. An alternative to this is having multiple anchor boxes on a **n x n** patch. This is depicted in Figure 6 **[3]** and is the method that we used.

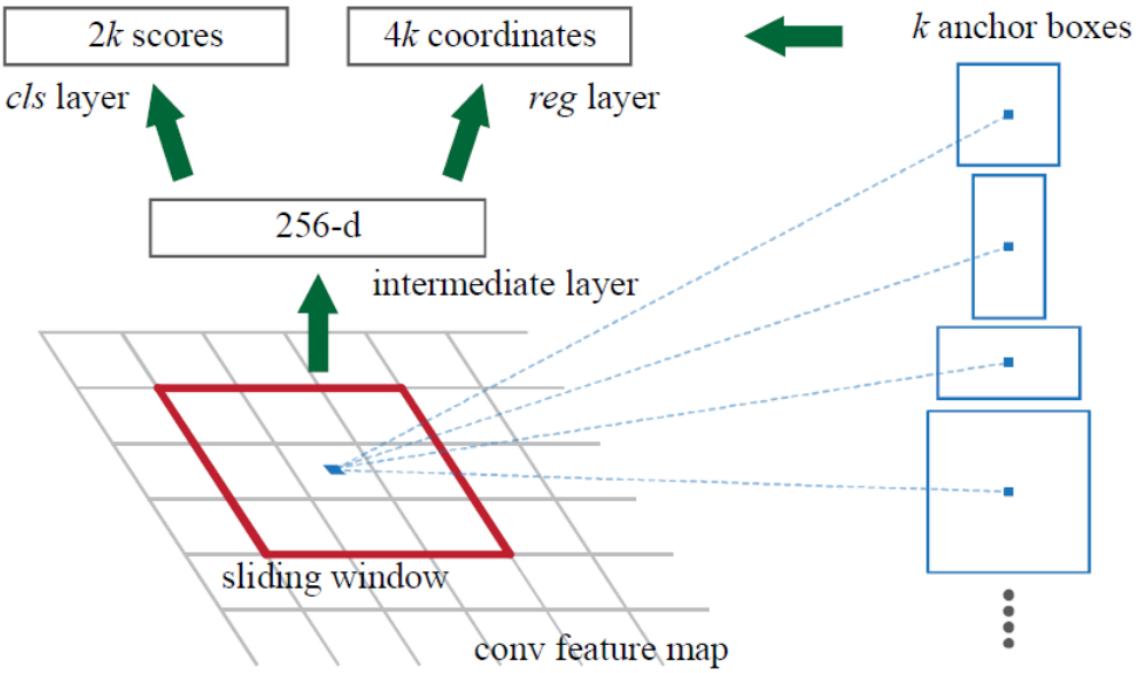


**Figure 6.** Anchor boxes deployed on a **3x3** sliding window **[3]**

The RPN is a neural network that is capable of learning to prune the anchor boxes over time. As with any neural network that learns over time, the RPN needs a loss function to supervise the learning. In fact, the RPN has two loss functions. The first is a classification loss function that is used to tell the model whether or not a certain anchor box contains the object of concern. This is a binary classification and uses a loss function such as binary-cross entropy. The second loss function is a regression loss function used to draw a prediction bounding box around the object of concern. This takes in the coordinates of the ground-truth bounding and predicted bounding boxes and applies an L1 or L2 norm to generate a loss value. The loss is minimised over time as the model improves at detecting objects and correctly drawing bounding boxes around them **[3]**.

Once the initial proposals are obtained, an operation called Region of Interest (RoI) pooling is performed. This can simply be thought of as an operation to group similar predicted bounding boxes and average them out. The problem with the RPN is that it can sometimes predict multiple bounding boxes corresponding to a single ground truth bounding box. Picking one can become a challenge and the process of RoI pooling overcomes this challenge. This technique is summarised in Figure 7 **[3]**. The specific algorithm that was used to accomplish this

is called Non-Maximal Suppression (NMS), which iteratively picks the bounding box with the maximum overlap with the ground-truth bounding box until only one such box is left.

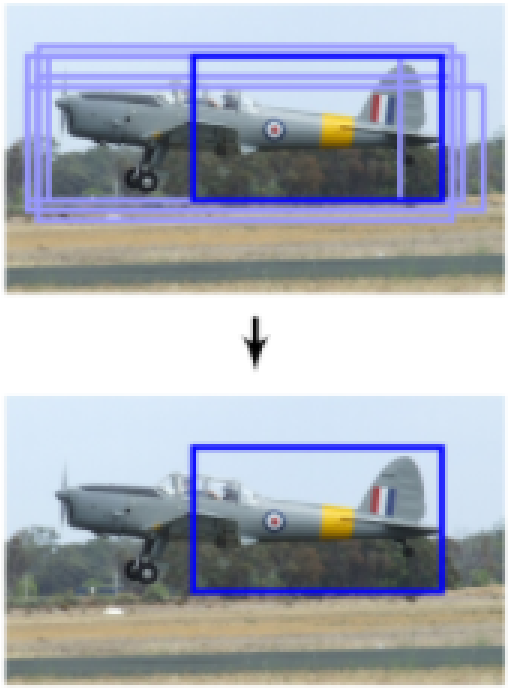

**Figure 7.** RoI Pooling using Non-Maximal Suppression (NMS) **[3]**

Finally, the pooled regions are fed into another neural network that classifies the object contained therein. This is a multi-class classification problem and requires the use of a loss function such as multiple-cross entropy. Furthermore, we also have a regression loss function that fine-tunes the final predicted bounding boxes **[3]**.

## 2.2.2 YOLO v3

Joseph Redmon and colleagues presented the first version of the You Only Look Once (YOLO) object detection model in 2016 **[7]** , providing a unified approach to object detection. Instead of selecting regions of interest like R-CNNs, YOLO tries to predict classes and bounding boxes for the whole image in one run of the algorithm. Figure 8 **[7]** illustrates this concept in action. YOLO first divides each image into an SxS grid of cells and each cell is responsible for predicting the objects that occur within it. Each cell predicts bounding boxes and class probabilities that are then pooled to get the final detections.

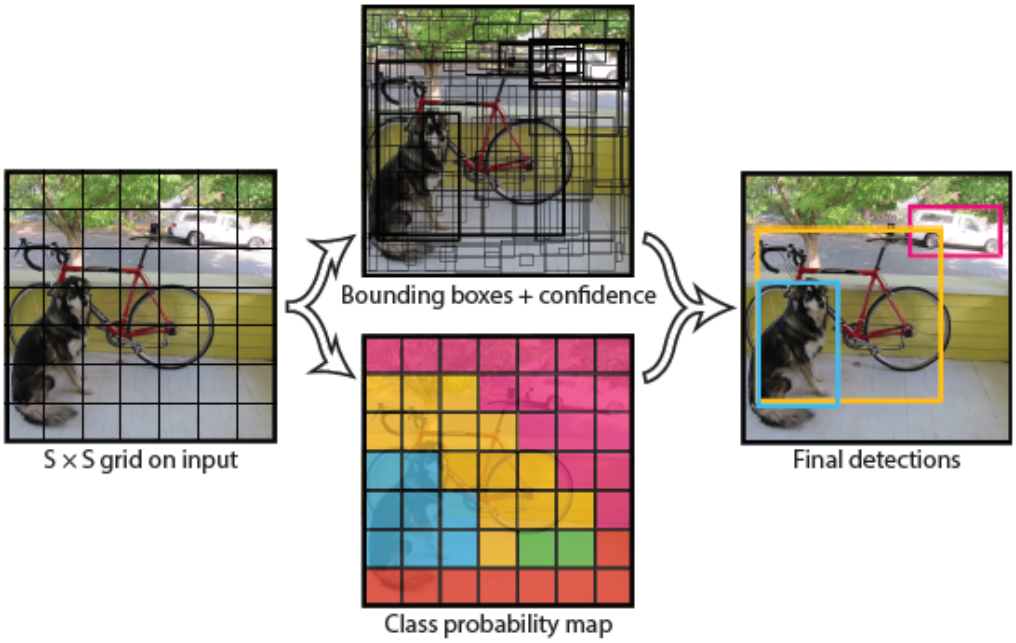


**Figure 8**. YOLO algorithm object detection approach. **[7]**

Redmon et al. published three papers on YOLO that each provided improvements to the model, with the most significant improvements coming in the second paper, "YOLO9000: Better, Faster, Stronger." **[6]** Here, Redmon et al. suggested the following changes:

1. Using a high resolution classifier that increases the resolution of the images being trained on during the process.
2. Using a variant of anchor boxes that utilizes K-Means clustering to generate dimension clusters for object sizes.
3. Allowing the model to predict object location coordinates from dimension clusters.
4. Using multiscale training i.e. training the model at varying aspect ratios.
5. Using batch normalization to stabilize model training.

These changes allowed the model to see significant gains in performance in the form of Mean Average Precision, making it competitive in terms of performance with methods like Fast and Faster R-CNN. YOLO v3 **[4]** is what Redmon et al. call an 'incremental improvement' over YOLO9000 wherein they suggest a new neural network backbone for the model called Darknet53 which is discussed in more detail in the Models and Results section of this paper.

### 2.2.3 Faster RCNN & YOLO v3 - A Comparison

We will analyze the pros and cons of both Faster RCNN and YOLO v3 in an attempt to determine which model suits our problem better. We start with the speed-accuracy tradeoff.

In the machine learning literature, Faster RCNN models are shown to generally be more accurate overall as compared to YOLO models. The tradeoff, however, is the speed at which Faster RCNN operates at both during training and test time. Faster RCNN operates in the range of 10-20 fps **[3]** while YOLO can achieve on average 50 fps **[4].** There are even versions of YOLO that are optimised for speed and are capable of achieving speeds upto 150 fps. This speed-accuracy tradeoff will play a bigger role when Proofpoint Inc. chooses to implement our models. YOLO might be a better option if the model is implemented in real-time. However, Faster RCNN would be a strong contender if implemented in near-real-time or when greater accuracy is required. For now, we have decided to implement versions of both Faster RCNN and YOLO models. These will serve as baseline models for all future work we do.

Another point of distinction between YOLO and Faster RCNN relates to versatility. Faster RCNN is more versatile because of its two-stage process. Different architectures could be combined to better tailor the model to our problem. YOLO on the other hand is a single-stage process and offers no such versatility. It has to be noted here that Faster RCNNs are prone to a greater number of false positives in the results. In other words, they incorrectly identify many more non-logos as being logos. This is because Faster RCNNs work off of region proposals and lack the global context required to contextualise objects. YOLO on the other hand does not have this issue as it does a single-shot classification on the whole image.

Finally, we would want to stress that both YOLO and Faster RCNN are very popular in literature and have optimised and publicly available implementations. Because both algorithms have their respective advantages, we decided to implement both of them and compare their results.

### 2.2.4 Small Object Detection

One of the first challenges that we recognized in this project was small object detection. Many web pages have either small social media logos in them while a sizable portion also have small versions of non social media logos. In the literature, small objects are defined in many ways with the most common descriptors being objects occupying less than 32x32 pixels in an image or objects occupying less than 1% of the area of an image. When examining the sizes of the logos we scraped for this project, we found that average sized logos tend to occupy about 1.7% of the area of our screenshots while small social media logos occupy only 0.07% of an image's area on average. However, given the high image resolutions we use in our scraped

images so far, the average size of our smallest logos is 57x47 pixels which means that they do fall below the threshold many algorithms set for small objects. Despite us not encountering any small logos during this project, we have included a brief summary of the related works in this area as highlighted during our literature review.

From our literature review, we identified five techniques in particular that are appealing as solutions for our applications. The first is data augmentation in the form of adaptive zoom **[8]**. In this technique, smaller objects in images are either upsampled to be larger or the entire image is rescaled so as to increase the sizes of the smaller objects. Another technique often used in literature is the Feature Pyramid Network **[9]**. As this technique is used in our Faster R-CNN implementation, it is discussed in detail in the Models and Results section. The third is using a loss function optimized for small object detection. Here, the most promising loss function is Focal Loss **[10]** which was initially developed for dense object detection but has proven effective in small object detection as well. This loss function is simply an addition to the cross entropy loss function wherein mistakes made by the model on harder samples (images with small or dense logos) are penalized more than mistakes on easier samples. This leads to the model paying more attention to the harder samples and learning more from them. The fourth technique involves the use of dilated convolutional layers **[11]**, i.e. layers with expanded convolutional filters and sparse parameters, to obtain larger receptive fields in the model. The final technique involves the utilization of Generative Adversarial Networks to narrow the representations between small and large objects **[12]**. As mentioned previously, despite our thorough literature review into the topic of small object detection, we did not encounter any logos that were smaller than 32x32 pixels.

## 2.3 Logo Classification

Accurately classifying a large variety of classes requires effective implementation of state-of-the-art deep learning classification models. We conducted a literature review of some of the most recent and best-performing models, which included models such as ResNet and EfficientNet. Upon reading the primary paper that described the development and performance of EfficientNet **[13,14]**, we decided to proceed using this architecture. As shown in Figure 9 **[13,14]**, EfficientNet demonstrates equivalent or higher performance than many popular architectures while also having a fraction of the number of parameters as those models. In other words, EfficientNets are often at least 5-10 times more efficient than competing model architectures.

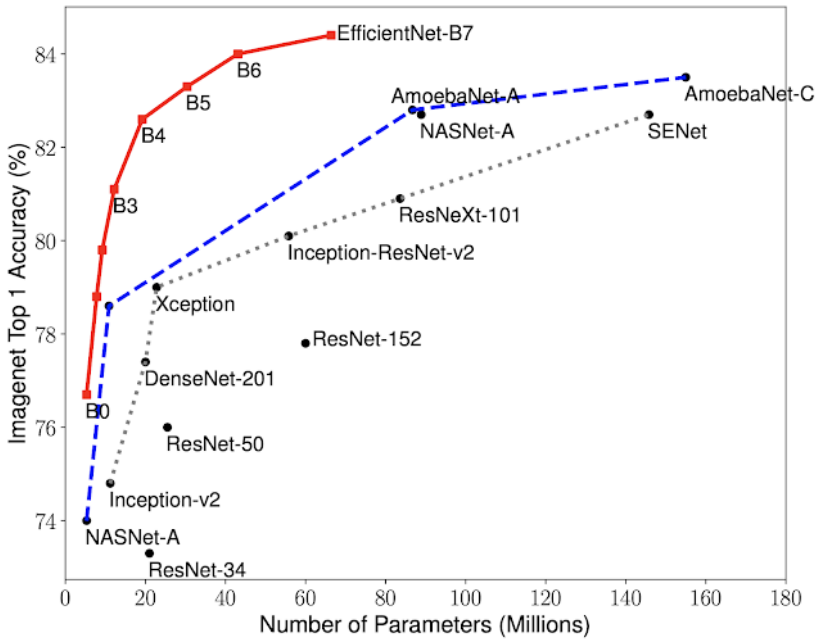


**Figure 9.** A comparison of the Top 1% accuracy of the various state of the art classifiers on the ImageNet dataset. **[13, 14]**

EfficientNets utilize what is known as compound scaling in order to achieve their high performance efficiency. In convolution neural networks, three methods can be used to scale a model. These are increasing the width of the filters to obtain finer feature maps, increasing the depth of the network by adding more layers, or increasing the image resolution. These methods are shown in Figure 10 **[13,14]**. However, many models tend to arbitrarily set these parameters and the computation required for noticeable performance increases is often too high. In addition, the performance gains will often saturate i.e. simply continuing to add layers or increase width will result in negligible performance improvements after a certain point. In an EfficientNet, the model uses compound scaling method, meaning that all three model scaling techniques are adjusted in accordance with a fixed computational resource constraint such as twice as many FLOPs as the baseline network. A grid search is performed on coefficients corresponding to width, depth, and resolution while maintaining a constant compound coefficient. The final coefficients for each of the three parameters are then applied to the baseline network in order to scale it accordingly **[13,14]**.

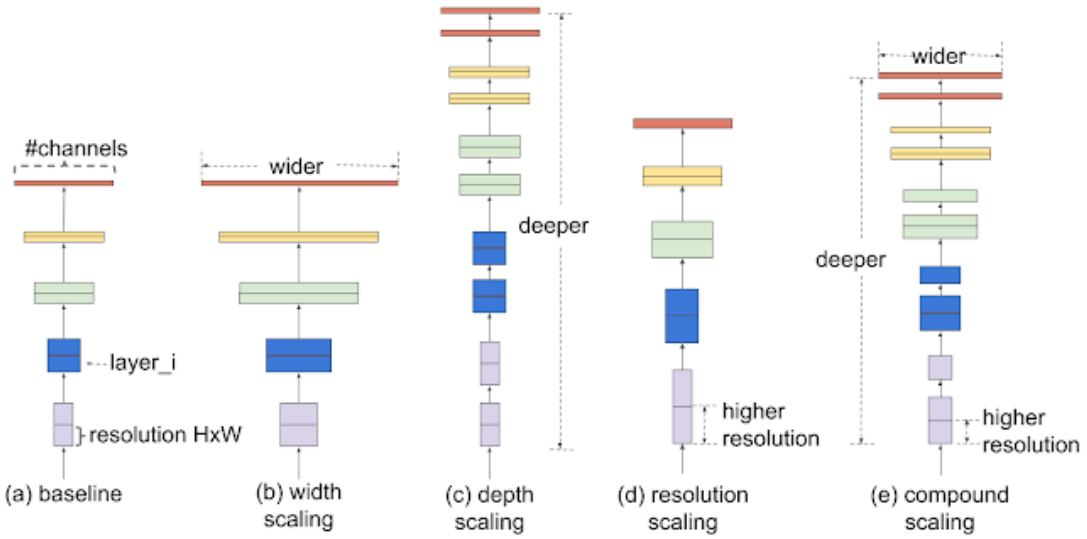


**Figure 10**. Comparison of methods used to scale a model. While many models individually scale one of (b)-(d), EfficientNets combine all three methods using a fixed ratio, greatly improving their efficiency **[13,14]**.

Due to EfficientNet's equivalent or better performance on ImageNet data with a fraction of the parameters, we chose this architecture as our logo classifier. We originally started our experiments by using an EfficientNetB0 architecture but then focused on using the EfficientNetB3 since it improved our classifier performance by over three percentage points. Future discussions of the classifier in this paper will only concern models that utilize the EfficientNetB3 architecture.

## 2.4 Open-Set Learning

Another major challenge we encountered while training our model was the problem of open-set learning. In simple terms, open set learning refers to the ability of the model to respond well to logos it has not seen during training time. We used two approaches to counter this issue: 1) Open Long-Tailed Recognition (OLTR) and 2) K-Nearest Neighbours. In the following subsections, we will describe these techniques.

### 2.4.1 Open Long-Tailed Recognition (OLTR)

This approach is based on a paper from 2019 titled 'Large-Scale Long-Tailed Recognition in an Open World'**[5]**. The literature surrounding open-set learning is sparse, but this paper proposes one of the best approaches available. We will explain the workings of this method in this section.

An example of the open-set problem is depicted in the following Figure 11**[5]**. The model sees many instances of cat photos, few instances of fox photos, and only one instance of a panda photo. The classes could represent something analogous in our problem. The cat photo could be analogous to a popular company logo such as Google, the fox photo could be analogous to an institution of repute such as Duke University (albeit not a very common logo), and the panda photo could represent a rare generic logo that does not occur more than once. The goal of the model should be to remember the training it underwent from the underrepresented classes and try to maximise accuracy during test time. However, this is only one half of the problem. What if the model is exposed to a new, previously unseen logo during test time? In Figure 11 **[5]**, this would be the picture of a rabbit. The model should be sensitive enough to realise that the rabbit photo is different from any of the other training data it has trained on and be able to classify it accordingly.

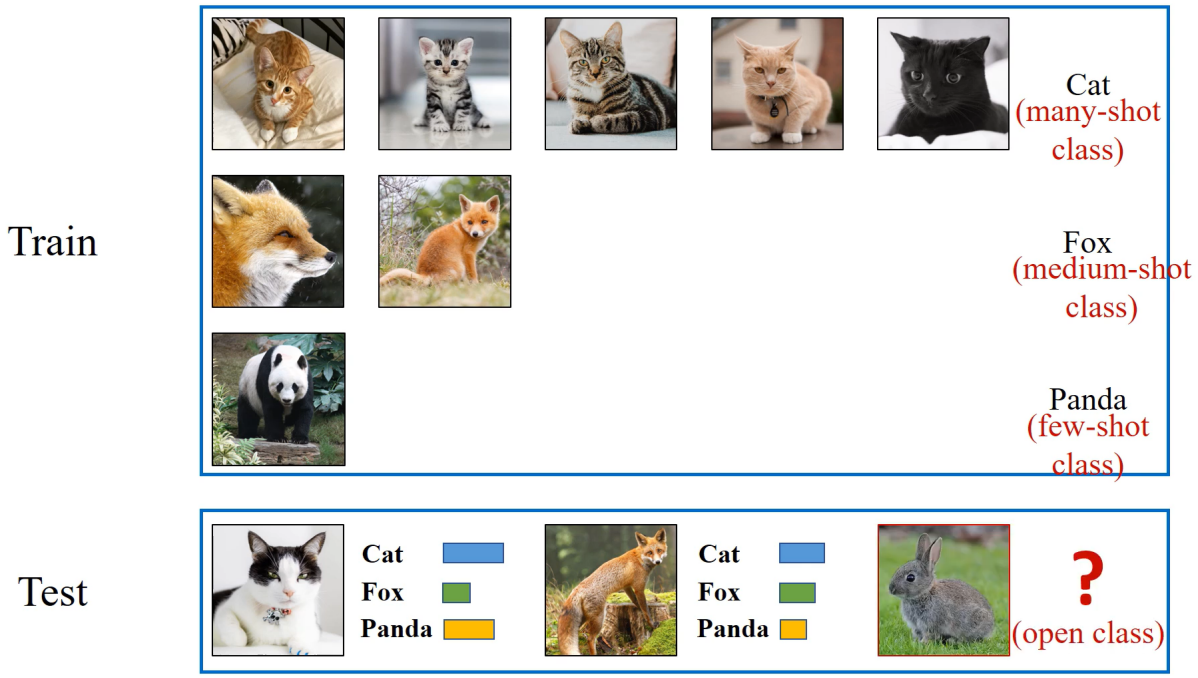


**Figure 11.** A simple example using animal photos to explain the concept of open-set learning. This example can be extended to the case of logos as well. **[5]**

The graph in Figure 12 **[5]** summarizes the above example in Figure 11 **[5]**. Only a few logos that we will encounter will be well-represented in the training data. Most of them will be under-represented and a few of them will be never seen by the model until test time. This is the problem of open-set learning and long-tailed distributions, and we will now detail a solution for solving this problem, shown in Figure 11 **[5]**.

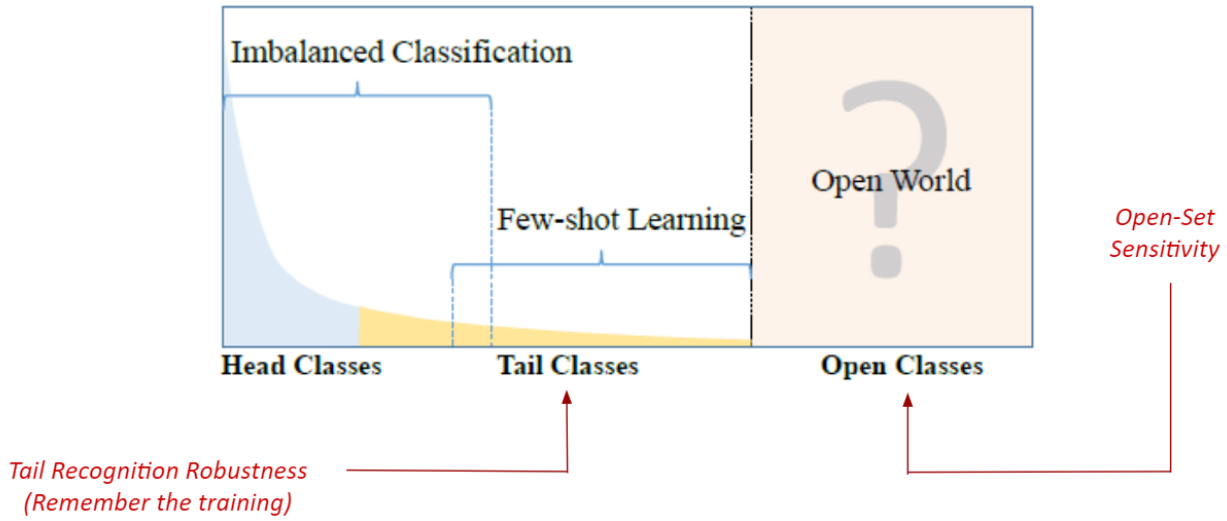


**Figure 12.** Graphical representation of open set learning **[5]**

This solution to solve the open-set learning problem uses a two-pronged approach. It uses techniques called dynamic meta embeddings and modulated attention. Dynamic meta embeddings can be understood with the help of an example. Say our model is exposed to a previously unseen image during test time. Using the direct features $v_{direct}$ from this image would be detrimental to the model as it has not done any previous training on similar features. Instead the model chooses to tap into a virtual memory that it maintains. This visual memory is very similar to a k-nearest neighbours clustering algorithm when visualised in 2-D. The visual memory plots the features of all images it has been trained on in 2-D and as expected images belonging to the same class will be clustered together. When the model is exposed to this previously unseen image it compares the direct features of this image with the clusters from the memory. Depending on how close or far away the new features are from any of the clusters, the model then infuses some features from the memory ($v_{memory}$) **[5]** into the direct features ($v_{direct}$) **[5]**. The amount of features to be infused is something the model learns over time. In fact, the model has a simple neural network called the 'Concept Selector' (Figure 13) **[5]** which can learn the amount of visual features that requires infusion.

The model also has another apparatus called the 'Hallucinator' (Figure 13) **[5]**. The hallucinator is similar to a Generator network in Generative Adversarial Network (GAN) and its goal is to create convincing variations of the object of concern in different surroundings. This is a form of data augmentation that modifies the data fed into the model in order to make the model more robust and less prone to overfitting. Together, the Concept Selector, the Hallucinator, and the Visual Memory combine the $v_{direct}$ and the $v_{memory}$ to produce $v_{meta}$ - the dynamic meta embeddings (Figure 13) **[5]**. This will then be used by the model to perform the classification.

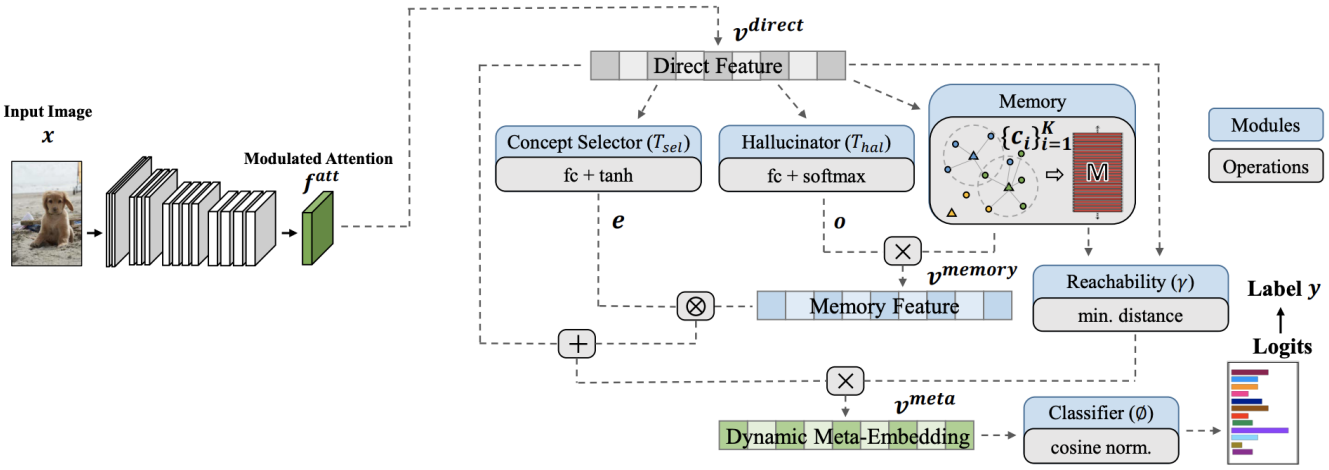


**Figure 13.** Proposed Architecture to solve the open set problem. **[5]**

Next, we will take a look at an alternative approach to counter the problem of open-set learning - the k-Nearest Neighbours approach.

## 2.4.2 K-Nearest Neighbours for Open Set Learning

For our open set classifier, it is important for us not only to distinguish known logos from unknown ones during the training stage but also to ensure that our solution is adaptive enough to learn new classes of known logos without any need to retrain. This means that if a logo **X** was unknown and unobserved during the model training stage and logo **X** becomes important post-training, our solution should be able to start reliably identifying logo **X** as known and also classify it accurately as logo **X** and not as any of the logos known during training. For this, we implemented a K-Nearest Neighbors (KNN) based open set learning approach that utilizes within class distances to identify unknown logos and, based on its architecture, can learn to classify any number of initially unobserved classes.

The training of the KNN method is a two stage process, both of which rely on the 'activation vectors' of a trained neural network classifier. Conventionally in the literature, the activation vector of an observation is the final layer, pre-Softmax, output of a neural network. For a neural network trained to differentiate between ***n*** classes, the activation vector is an n-dimensional vector where each value at index ***i*** corresponds to the degree of certainty the neural network has that the observation belongs to class ***i***. Some researchers, such as the developers of OLTR, also characterize the output of the second last layer of a neural network as an activation vector. Generating the second last layer activation vectors can be seen as applying a transformation to the input image that condenses the information stored in it and magnifies the

differentiating qualities of the image. Yet other researchers choose to concatenate both the last and second-last layer activation vectors and call the concatenated vector an activation vector as well.

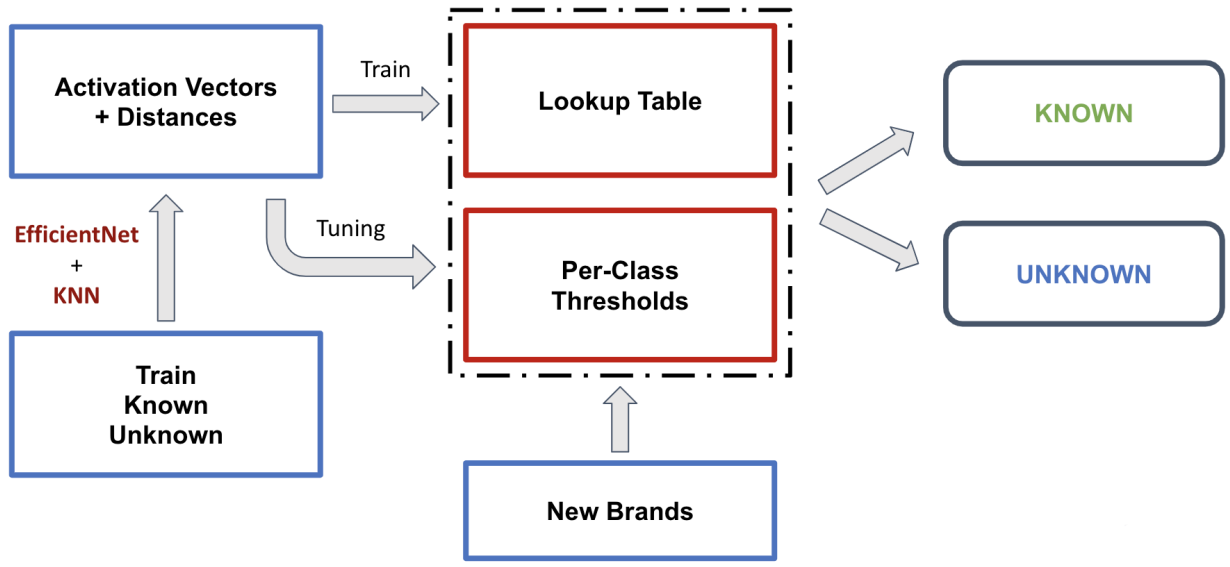


**Figure 14a**. Illustration of KNN open set method architecture.

Figure 14a illustrates the architecture of our open set method. The first stage of KNN training involves generating a Lookup Table of train data activation vectors. If there are ***m*** training data observations and each activation vector is ***n*** dimensional, then this is a ***m*** x ***n*** matrix. The label associated with each activation vector is also stored. At the testing stage, activation vectors are generated for each test data observation and then their nearest neighbor in the training data is found using the Lookup Table. If the distance of the test data point to its nearest neighbor is within the class specific threshold of the neighbor's class, then the test data point is assigned the same label as the nearest neighbor. Otherwise, it is classified as an unknown observation. This method works because the distance of true known class observations from their counterparts on the Lookup table are likely to be lesser than that of unknown observations, as illustrated in Figure 14b.

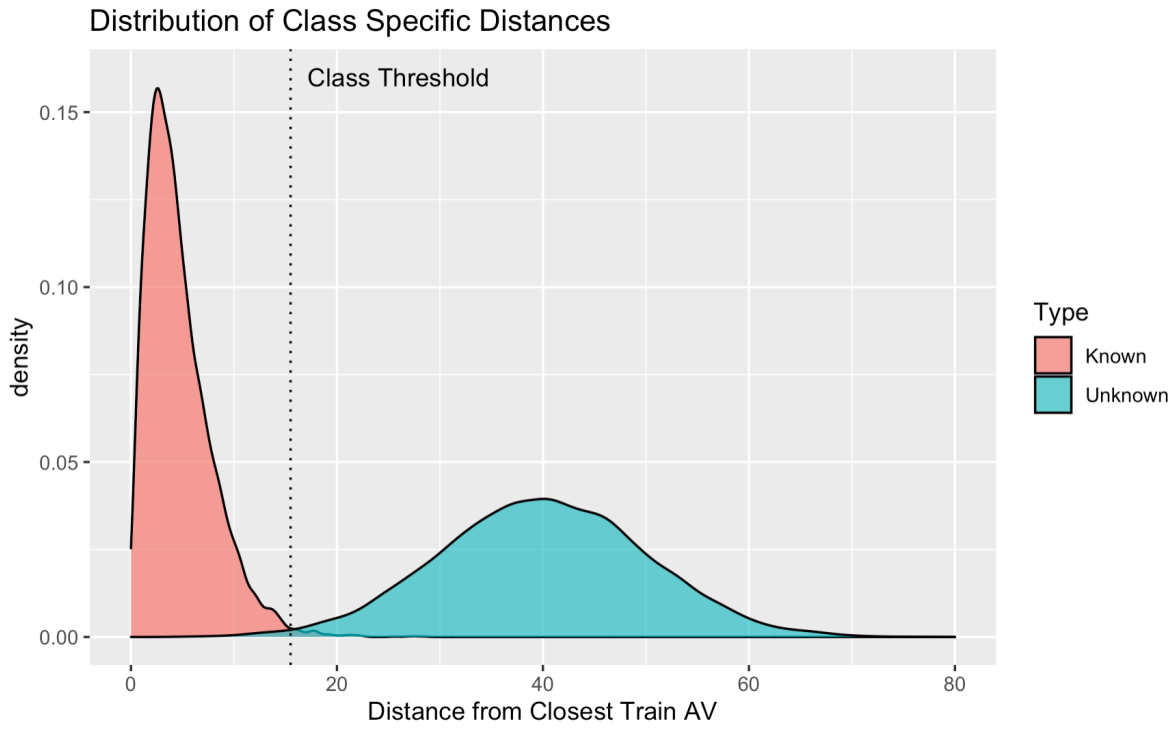


**Figure 14b**. Illustration of how a threshold is chosen in the KNN open set method. The distribution of distances of unknown logos from their nearest neighbors in the train data set is expected to be further from 0 (to the right) than the distribution of distances of known logos belonging to the class of the nearest neighbor in the train data set.

This open set classification strategy requires a threshold to be tuned for each class in the Lookup Table. This tuning is done in the second stage of KNN training. First, distances are calculated for all training images and all images in a held out dataset of known logos to their nearest and second nearest neighbors in the Lookup Table. Additionally, distances are calculated to each observation in a large dataset of unknown / unobserved logos as well as icons and large pieces of text that can easily be mistaken for logos. Using these distances, a per-class threshold is tuned for each class so that the within class accuracy is maximized and the largest number of unknown images are classified as unknown with the minimum number of known logos being classified as unknown. The KNN method relies on the existence of a large dataset of unknown logos so that a robust distribution of unknown logo distances can be generated for each known logo class for accurate threshold tuning. Generally, only the distances to the nearest neighbors are used for threshold tuning while the distances to the second nearest neighbors are utilized if too few or no nearest neighbor distances are available for a given class. Distances greater than those from the second nearest neighbor are never used as this leads to poor threshold tuning. In the case where a very large number of classes is used for training, it is possible that not even one unknown logo is available for threshold tuning for some classes. In such situations, we choose a threshold as the sum of the maximum distance observed for that class and two standard deviations of the observed distances in that class.

The biggest advantage of the KNN method is that any number of new classes can be added to the Lookup Table and used for open set classification without the need to do any expensive neural network retraining. This is because adding a new class is as simple as calculating train activation vectors for that class and appending them to the Lookup Table of activation vectors. Thereafter, the per-class distance distributions and thresholds can be recalculated, or recycled, this time including the new classes, allowing for prediction on the new classes.

# 3. Data

## 3.1 Logo Detection

In the following subsections we will discuss our approach towards collecting data for training the detectors - Faster RCNN and YOLO v3. Note that at the detection stage, we are only concerned about whether a certain object is a logo or not - a binary classification task.

### 3.1.1 Data Collection: Website Scraping

Over the course of this project, we scraped and took screenshots of the company webpages and wikipedia articles of all the companies in the S&P 500 list. We were able to successfully capture a large amount of variety among the logos in the screenshots in terms of logo size, number of occurrences, location, density, and clarity / visibility. Additionally, we obtained 375 images that contain no logos to assess the performance of our models when faced with such samples. Figure 15 shows a typical example of the kinds of screenshots we obtained, showcasing the variety we have captured. More examples showcasing the variety among our screenshots can be found on our GitLab.

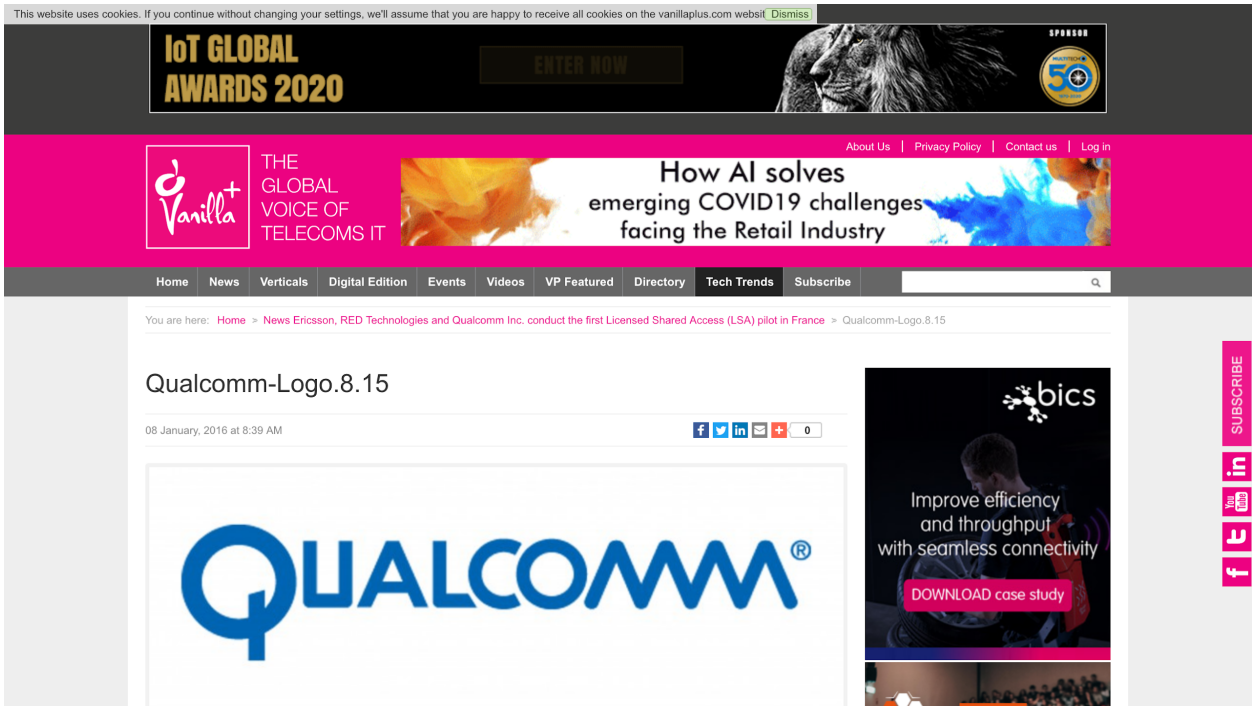


**Figure 15.** A typical screenshot in our data set. This image has a large amount of variety in the sizes, locations, and orientations of the logos.

Our screenshots were annotated in the PASCAL VOC format, which was chosen due to the ease with which it can be converted to COCO and YOLO annotation formats. To ensure minimal discrepancies between annotations, we established a number of rules that we adhered to in our annotation process. These were the following:

1. All logos, no matter how small, on an image must be annotated.
2. All unfamiliar logos need to be labeled ‘generic_logo’.
3. If a logo is darkened or has limited visibility, it should still be labeled.
4. Partially occluded logos need only be labeled if they are more than 75% visible and still identifiable.
5. Images with no logos are not to be annotated.
6. All S&P 500 logos must be labeled with their predetermined labels.

Regarding generic logos, it is possible that a logo that is labeled a recognized logo by one annotator is instead labeled a generic logo by a different annotator. Discrepancies in the labeling of the generic logos were resolved through multiple individual audits. More general discrepancies between logo annotations were resolved through an annotation cleaning script that used string matching to identify discrepancies and amend them. Figure 16 shows a typical example from an image annotation by our team.

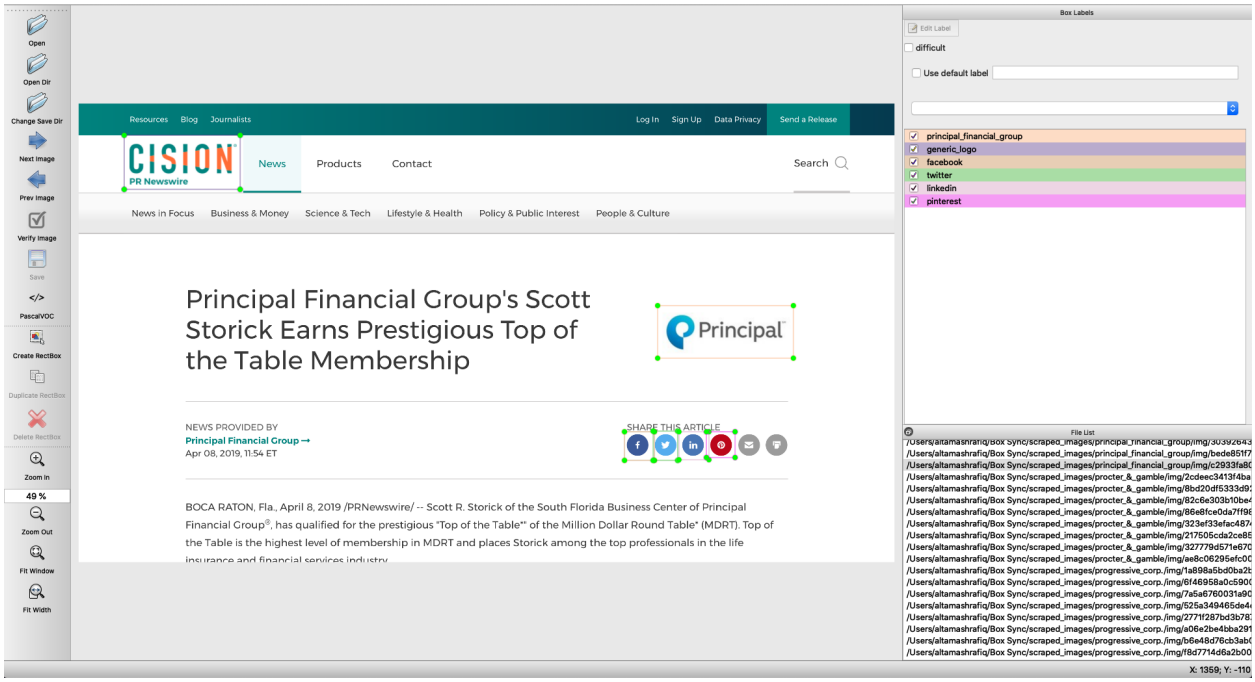


**Figure 16**: Sample Annotation. Note that the main logos have been annotated, but icons such as the mail icon next to the Pinterest logo have correctly not been annotated.

## 3.1.2 Exploratory Data Analysis & Statistics

Overall we obtained 5,225 screenshots that were annotated using the labelling tool labelImg. All of our images have a resolution of 1440x2560. Across our 5,225 screenshots, we annotated 17,154 logos with an average of 3.3 annotations per screenshot. That is, between 3 and 4 logos appear per screenshot on average in our data set.

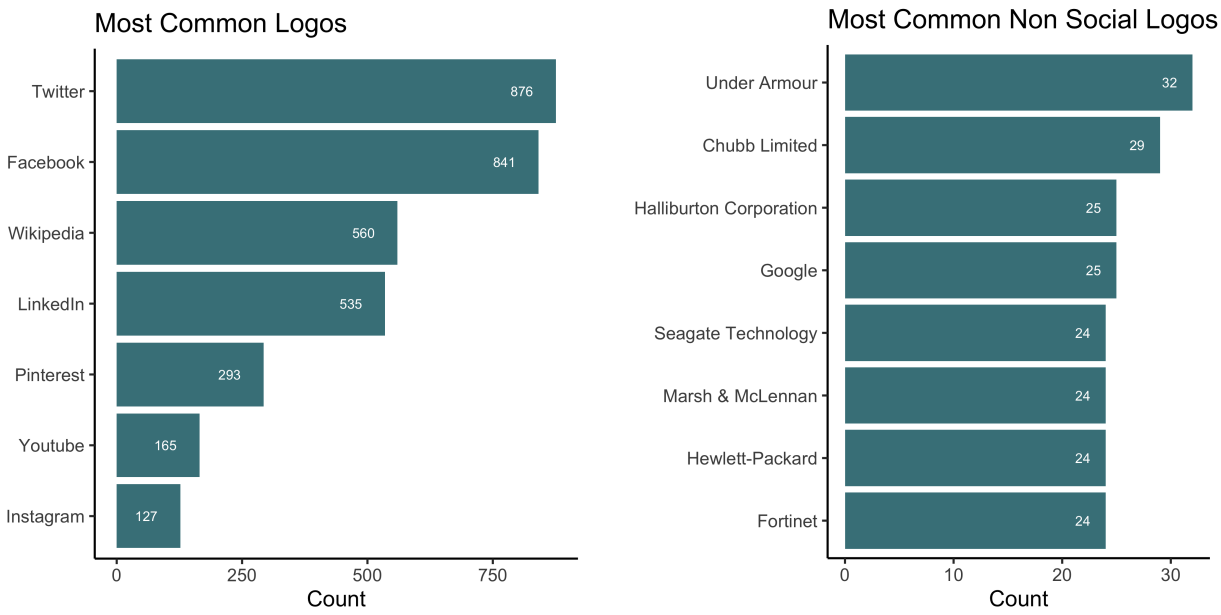


**Figure 17**: Distribution of logos from our screenshots. Note the prominence of social media logos.

Figure 17 shows two bar charts showcasing the number of occurrences in total, across all images, of the most common social media logos and the most common non social media S&P 500 companies in our data set.

The data was divided into train, validation and test datasets. The train set consists of 3,935 screenshots, the validation consists of 696 screenshots and the test set consists of 644 screenshots. Many of the images in the test dataset were specifically chosen for their difficulty, meaning that the test dataset can be viewed as a challenge dataset in our application while the validation dataset can be viewed as a more accurate representation of the kind of environment our models are designed to be deployed in.

## 3.2 Logo Classification

We will now detail our data collection process to train the logo classifier.

### 3.2.1 Data Collection: Bing Images and Extracted Annotations

In order to train our logo classifier, we needed to collect representative logos for each brand of interest. Our classes consisted of the S&P 500 companies, since these brands were used for website extraction for the logo detectors, as well as a collection of brands prominent in the technology, industrial, financial, and internet sectors. The latter set of brands were obtained from

1000logos.net, a site that contains the logo history of over 4000 brands and organizes each brand into specific categories such as internet and financial. After combining the S&P 500 and 1000logos.net brands, removing any duplicates, and adding a few others that were not in either location but still common to our extracted webpages, such as BusinessWire, we had obtained 1098 unique classes.

To obtain sample images for each class, we utilized two main sources. The first was the bing-image-downloader Python package, which allowed us to rapidly download images from Bing for each class. We ultimately limited the number of images per class to 15 since the logos of a particular brand tend not to have extensive variation. In addition, we knew we were bound to have images that did not correctly represent the brand or contained a mixture of different brands. Examples are shown in Figure 18. Having a higher download count enabled us to still have samples even if some needed to be removed. Furthermore, the images downloaded from Bing were all high resolution images, some greater than 1920x1080, and thus they represented a "pure" logo dataset.

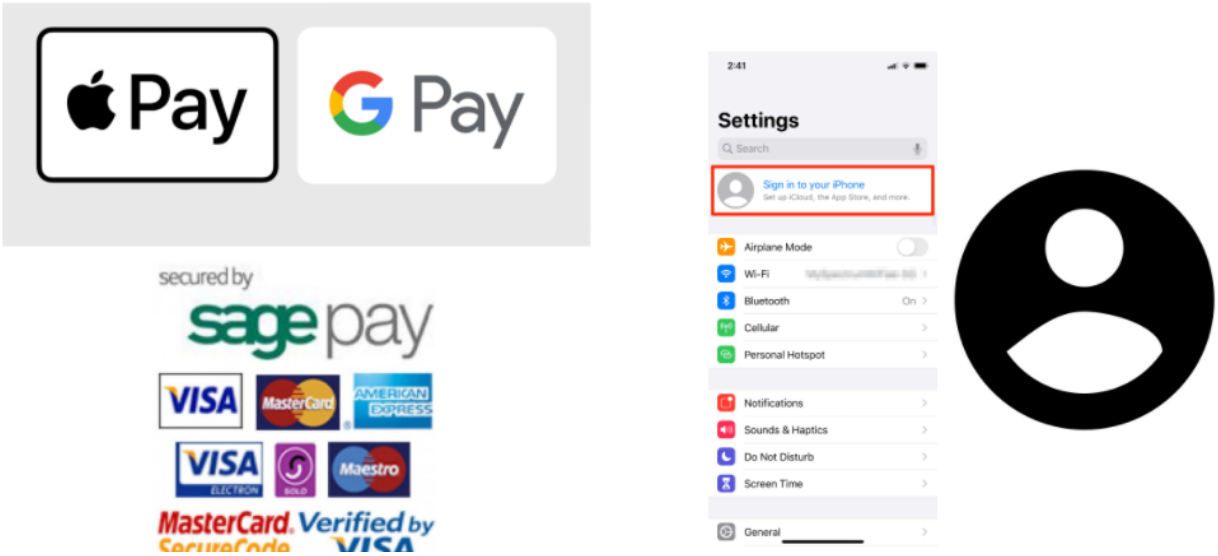


**Figure 18:** Examples of logos obtained from Bing images that were removed from the dataset. The images on the left contain multiple logos within a single image while those on the right do not represent brand logos at all.

The second source of images were obtained by extracting our annotated logos from our logo detector training set. For each of those screenshots, we had annotated logos on the webpage and labelled them with the respective company name or "generic_logo" for logos we did not recognize. By extracting the bounding box of the annotated images and placing them in their respective labelled folders, we now had additional samples for each class. Furthermore, these extracted annotations were less "pure" compared to the Bing images in that they were often of

lower resolution, may have had different background colors, or may have been extracted from a real-world image as opposed to a standard logo placed on a webpage. Figure 19 shows sample images from both the Bing image dataset and the extracted annotation dataset.


**Figure 19**: A comparison of logos collected from the Bing Downloader package and from extracted annotations. The images downloaded from Bing, while rescaled, tend to be much higher resolution and consistent. Those extracted from the annotations may have different background shades, additional text, and are generally of a lower image quality.

For the extracted annotations, images of the same brand but in different folders, such as "amazon" and "amazon_inc," were merged. Typos in folder names were corrected, and common suffixes of brands, such as "inc", "co", "group", and "company" were removed, with images of the same brand but varying suffixes merged together. The names of the extracted image folders were matched with those of the original 1098 classes i.e. the Bing images for each class. Only brands that comprised the 1098 classes, all of which had Bing images, had their respective extracted annotations merged. All annotations of the respective class were merged except for common social media brands such as Facebook, Twitter, LinkedIn, Pinterest, Instagram, and YouTube. These brands had hundreds of annotations, so we randomly sampled 30 annotations from each and added those to the dataset. This value was chosen because outside these popular social media brands, no other brand had more than 30 extracted annotation samples. In addition, many of these social media brands' annotations were similar, so adding hundreds more of these would not have improved the diversity of our samples nor contributed to a significant increase in performance.

### 3.2.2 Image Processing

Prior to training the models, several image processing and filtering steps were conducted to maximize performance. Initial runs of the classifier resulted in warnings about some images being in an RGBA format instead of an RGB format. An RGBA image contains an alpha channel

in addition to the standard red, green, and blue channels, and this channel controls transparency of an image. However, deep learning models for image classification require images to be in standard RGB format, so we converted any RGBA image to an RGB format. Only some of the images downloaded from Bing had an RGBA format and these were all PNG images. None of the extracted annotations had an RGBA format. Images that could not be converted were deleted, but fewer than 10 total in the entire dataset had this issue. In addition, the classifier also raised warnings about corrupt PIEXIF data, which is metadata often associated with jpeg images. A script was written to remove all of the metadata, and no more warnings resulted.

### 3.2.3 Dataset Creation

After collecting and processing these images, the logos downloaded from Bing and the extracted logos from the training annotation dataset were merged by respective class. Then, the images were split so that 75% were in the training data and 25% were in the validation data. Images from the test annotation set were used for the test set, but only 320 of the 1098 classes were present in the test set and only 32 of these classes had more than 5 images in the test set. The remaining classes did not have respective images from the test annotation set. The final dataset had 12878, 4668, and 1165 images in the train, validation, and test sets respectively. Distributions of each dataset are shown in Appendix A.1.1.

# 4. Results and Discussion

## 4.1 Logo Detection

We shall first discuss the specifics of the models used and the results for the logo detection task.

### 4.1.1 Performance Metrics

In the machine learning literature, one of the most common ways of quantifying object detection performance is through the use of precision and recall metrics at a fixed Intersection over Union, or IoU (see Figure 20a), generally of 0.5. In our application, the quantification of our detectors' prediction precision is complicated by our use of open set learning. During the detection stage, our detectors are prone to drawing bounding boxes around a sizable number of icons and headings, which adds to the tally of false positives, thereby lowering precision significantly as recall rises.

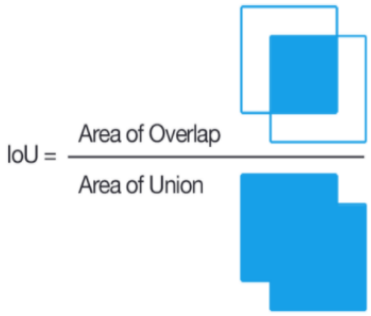


**Figure 20a.** Pictorial Depiction of the Intersection over Union (IoU) metric.

However, our detection process is not independent of our open set learning system which in our experiments proved to be highly effective at labeling boxes around icons and headings as 'Unknown' logos (or icons) that hold no importance in our application. Due to our open set learning, the standard, unadjusted precision metric is inadequate in terms of quantifying the performance of our detectors. As such, in this paper we utilize an adjusted precision metric wherein all detector false positives that are classified as 'Unknown' by our open set learning system are ***not*** included in the calculation of the precision metric. This leads to the generation of precision-recall curves that are significantly more reflective of the true performance of our models in the face of the open world context they are designed to be deployed in. Figure 20b illustrates how adjusting the precision metric improves the precision recall curve and in turn makes it more representative of open set detector performance. All precision metrics quoted in this paper are based on the adjusted precision metric.

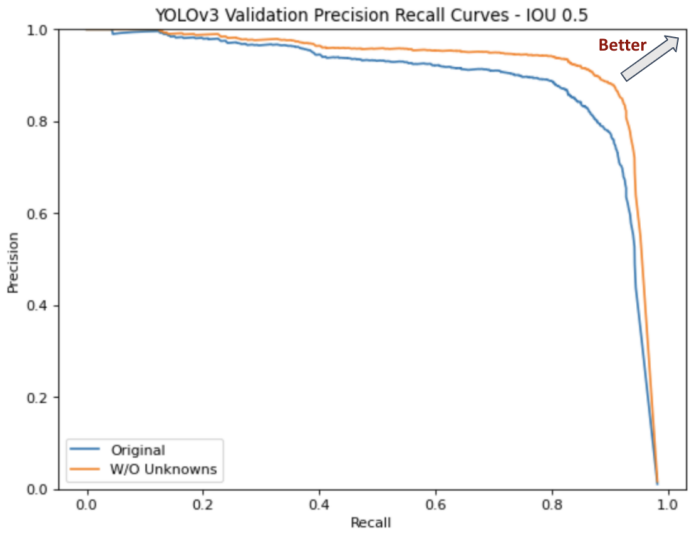


**Figure 20b.** Adjusted Precision-Recall curves (for YOLO v3 on our validation set) for the quantification of open world detector performance. Applying the adjustment makes it easier to

discern the true performance of our models.

### 4.1.2 Faster R-CNN

Our Faster RCNN model was trained using Facebook AI Research's (FAIR's) Detectron2 package (https://github.com/facebookresearch/detectron2). Detectron2 is a highly optimised, one-stop-shop for most object detection algorithms. We chose a ResNeXt 101 backbone that was equipped with a Feature Pyramidal Network (FPN). We will now briefly discuss the details of ResNeXt and FPN to familiase the reader with some architectures and terminologies.

The ResNeXt architecture takes its inspiration from the more common ResNet architecture, which was made popular because of its ability to deal with the vanishing gradient problem. Vanishing gradients are problematic for deep neural network training because they inhibit the backpropagation algorithm by resulting in non-updated or zeroed weights. ResNets solved this problem by using skip connections i.e. skipping a few layers before reconnecting. This is depicted in Figure 21. By solving the vanishing gradients problem, ResNets achieved state-of-the-art performance for the time.

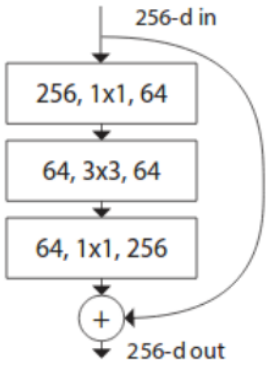


**Figure 21.** Use of skip connections in ResNet architectures

The ResNeXt architecture, on the other hand, has a parallel stack of layers all with identical topology. The number of parallel stacks is defined as the 'cardinality' - a hyperparameter that the user specifies. The paper also goes on to show that changing the cardinality has an effect on the accuracy of the model depending on the problem being solved. The following figure (Figure 22) depicts a ResNeXt block with a cardinality of 32.

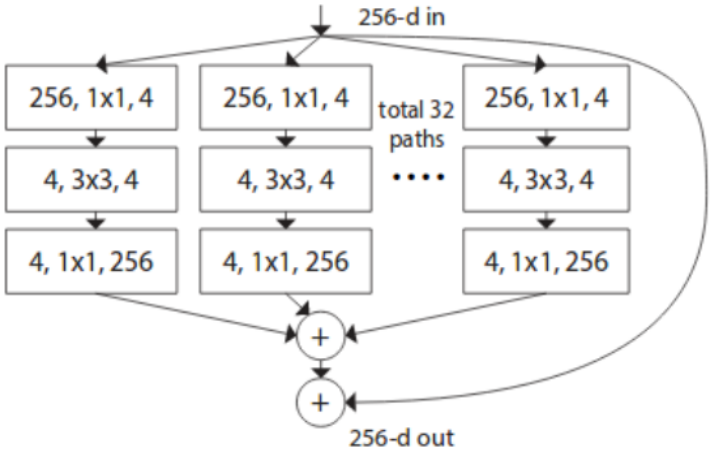


**Figure 22.** A ResNeXt block with cardinality 32

While the ResNeXt architecture is highly useful, we were more intent on incorporating the FPN into our model. The FPN is very useful when detecting objects of varying sizes. This is a recurring theme in our dataset which comprises logos of varying sizes. More importantly, FPNs are effective in detecting small objects which also makes them very useful for the problem that we are trying to solve. The architecture of a FPN is depicted in Figure 23.

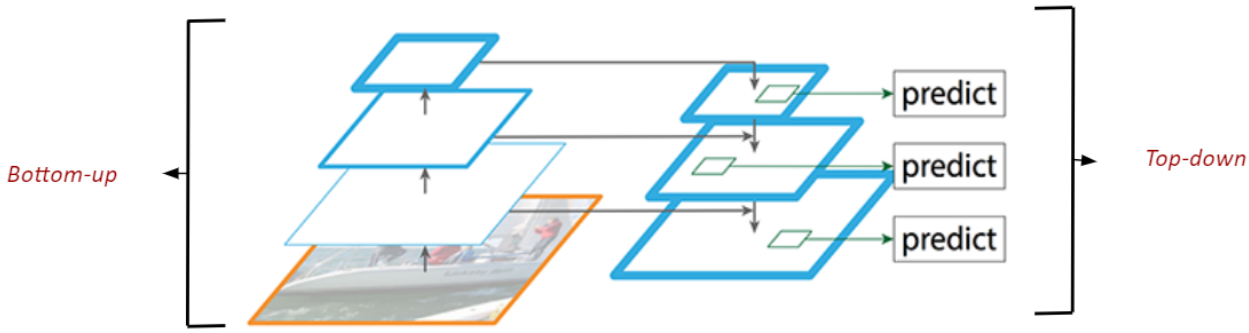


**Figure 23.** A Feature Pyramidal Network (FPN)

Briefly put, many of the features corresponding to the smaller objects on the image are lost during the bottom-up approach. In a typical feed forwards convolutional neural network - this would be the end of the process. An FPN attempts to recapture the features corresponding to the small object by passing the features through the network via a top-down approach. This process is quite successful in recapturing the features corresponding to the small objects and improving accuracy. Also notice the skip connections in the FPN. Because of the multiple passes through the neural network the FPN sometimes has issues localising objects and drawing bounding boxes around them. The skip connections help overcome this challenge.

Using the ResNeXt + FPN backbone we trained our model using an NVIDIA Tesla K80 for about 5-6 hours. The following are the results of the Faster RCNN on the held out test set. At an IoU threshold of 0.5 and a fixed recall of 0.87, the model achieves an adjusted precision of 0.88 on the validation set and 0.81 on the test set. Furthermore, the average time for detection on the test set for Faster RCNN was 150 millisecond per image.

We will now look at some of the predicted bounding boxes in our results to better understand how the model did in our test data. As we can see in Figure 24, the model is highly effective at detecting all the logos in the screenshot. Note how the model also does an excellent job detecting the smaller social media logos. A similar result can also be seen on Figure 25, where the model does a great job of identifying both the Wikipedia logo and the company logo.

Unfortunately, the model was susceptible to false positives. This was because the model works off of region proposals and as a result does not have a very good intuition of the global context of the image. An example of a false positive is shown in Figure 26. Such occurrences were uncommon but were likely to occur when a certain large text on the webpage and a distinctly different font as compared to the rest of the webpage.

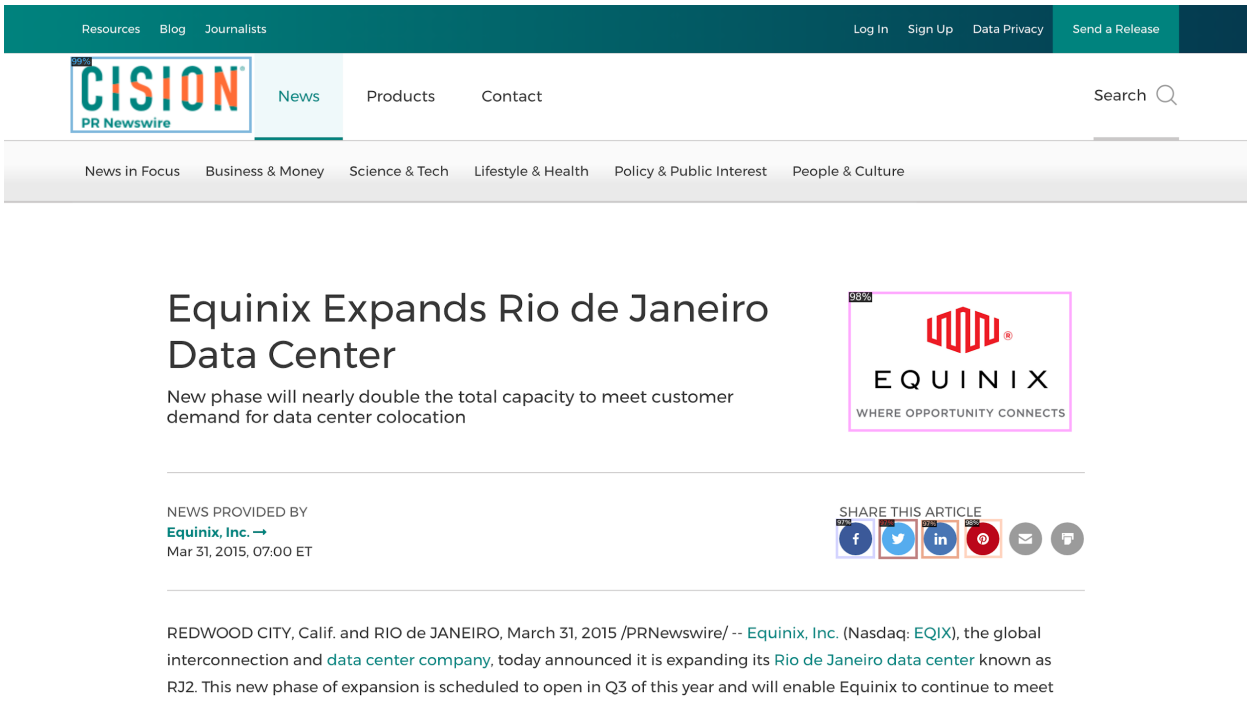


**Figure 24**. Logo detection by the Faster RCNN on a validation image

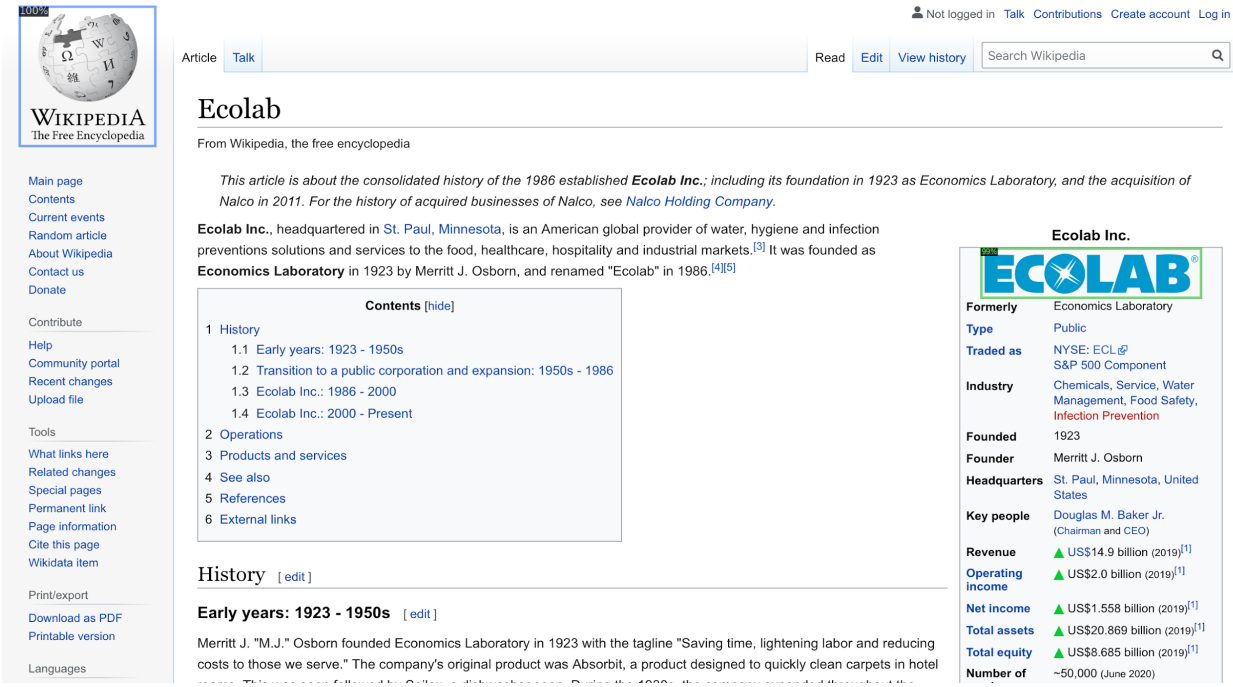


**Figure 25**. Logo detection by the Faster RCNN on a validation image

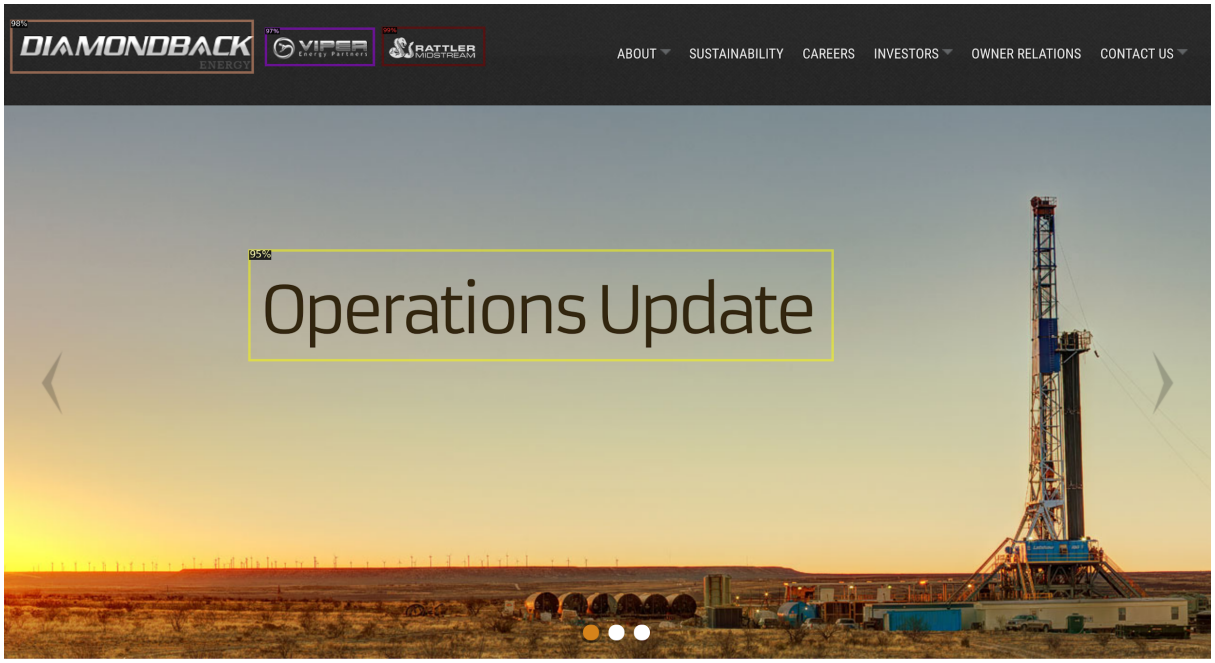


**Figure 26**. Logo detection by the Faster RCNN on a validation image. Note the false positive

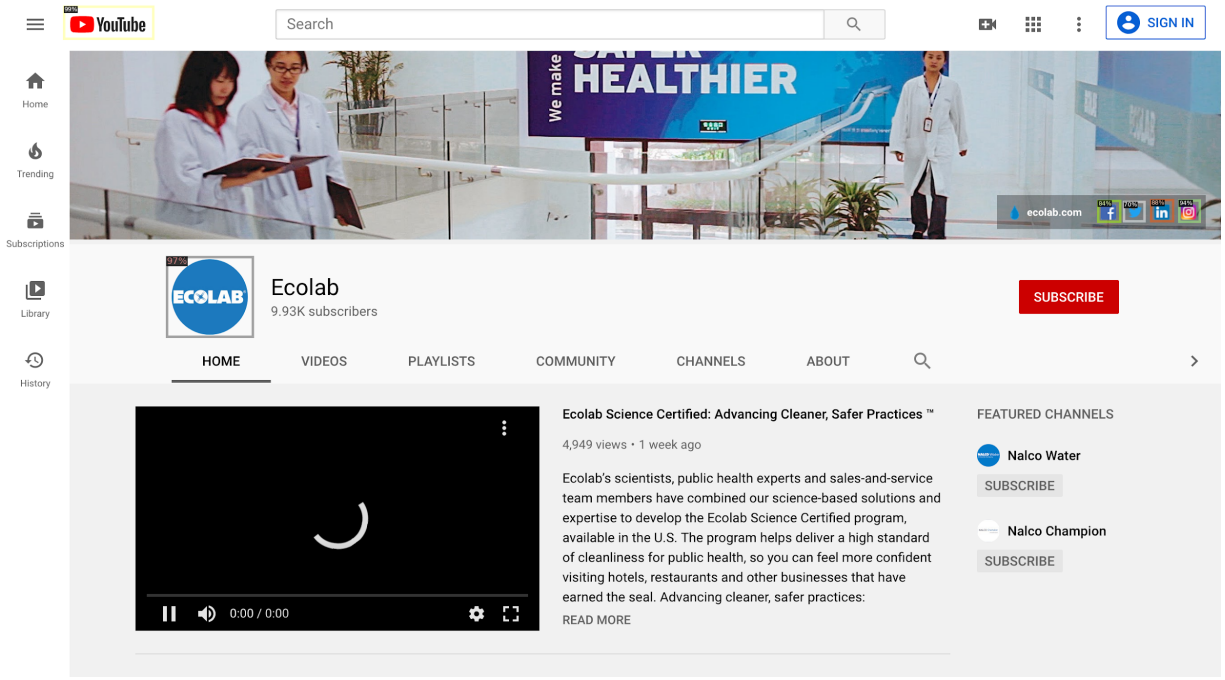


**Figure 27**. Logo detection by the Faster RCNN on a validation image. Note the false negatives

False negatives were less common but were usually small, less frequent logos that were missed. Note the tiny company logos in Figure 27 that have not been detected by our model. We shall now look at the results for the YOLO v3 model.

### 4.1.3 YOLO v3

We trained our YOLO v3 object detection model with a Darknet 53 **[4]** backbone with a spatial pyramidal network. Darknet53 is a neural network developed by Redmon et al. specifically for deployment with YOLOv3 to balance predictive power and test time speed. It contains 53 convolutional layers and a total of 23 resblocks similar to those described in the Faster R-CNN section. See Figure 28 for details about the network architecture.

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| | Convolutional | 32 | 1 × 1 | |
| 1× | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| | Convolutional | 64 | 1 × 1 | |
| 2× | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| | Convolutional | 128 | 1 × 1 | |
| 8× | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| | Convolutional | 256 | 1 × 1 | |
| 8× | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| | Convolutional | 512 | 1 × 1 | |
| 4× | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

**Figure 28**. Darknet53 architecture.

For our implementation, we utilized the open source YOLO v3 codebase developed by Ultralytics LLC available here: https://github.com/ultralytics/yolov3. Our model was trained for 50 epochs with a batch size of 3. Training lasted for approximately 9 hours on an NVIDIA Tesla V100 GPU. At a fixed recall of 0.87, YOLO v3 achieved an adjusted precision of 0.91 on the validation set and 0.85 on the test set at 0.5 IOU. At detection time, the model predicts on our large 1440x2560 images in an average of 45 milliseconds per image, providing a 3x speed gain over Faster R-CNN.

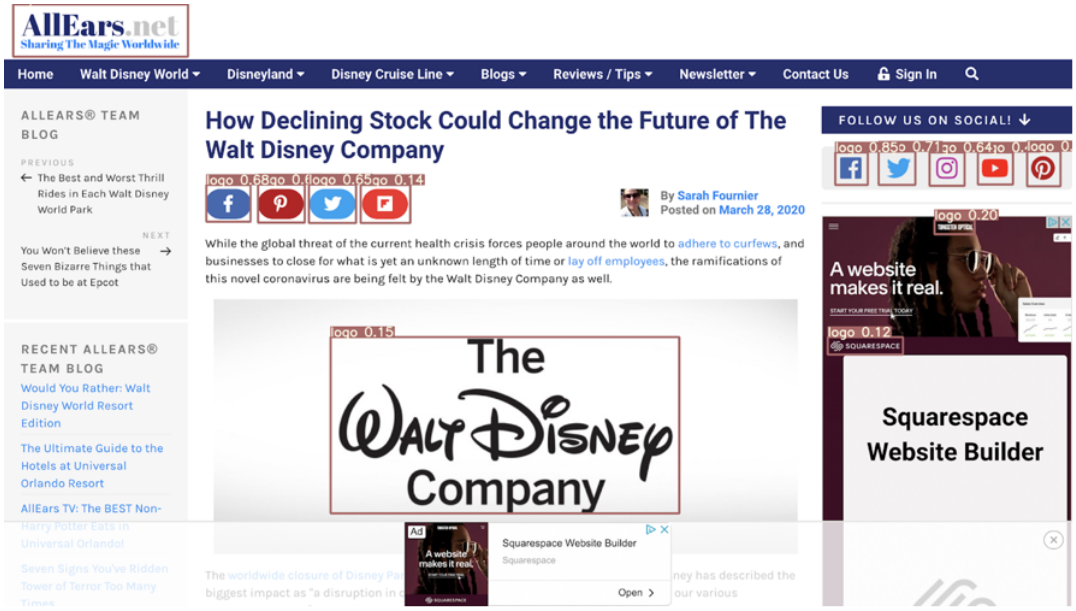


**Figure 29**: YOLO v3 no misclassifications.

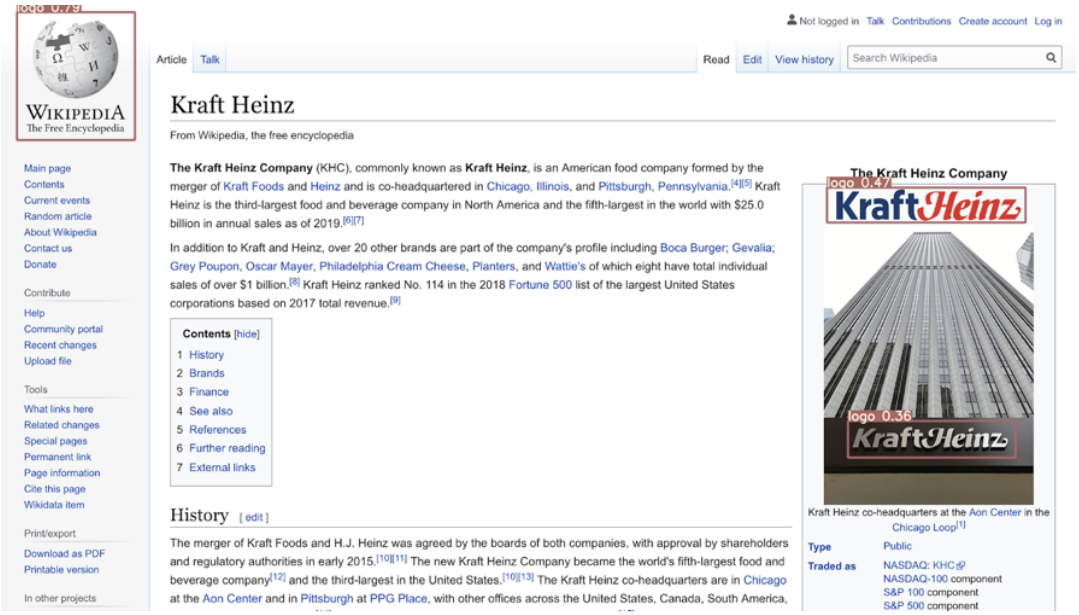


**Figure 30**: YOLO v3 detection of wild logo.

Similar to Faster R-CNN, YOLO v3 performs strongly on screenshots where there are prominent logos in expected locations such as main company web pages or Wikipedia articles. Figure 29 and 30 show examples of images where YOLO v3 correctly identifies all the logos in the screenshots, including the relatively small social media logos, without obtaining any false positives. In Figure 30, YOLO v3 even captures a prominent logo in the wild. In our experiments, YOLO v3 struggled on screenshots with significantly large logos, particularly those situated at the center of the image, as well as logos that were darkened or distorted in some way. Figure 31 shows a characteristic example of an image where YOLO v3 struggled. While the model was able to pick up the main logo at the top left as well as the small social media logos on the top right, it did not identify the large T-Mobile logo at the center of the webpage, which is the most prominent logo on the webpage.

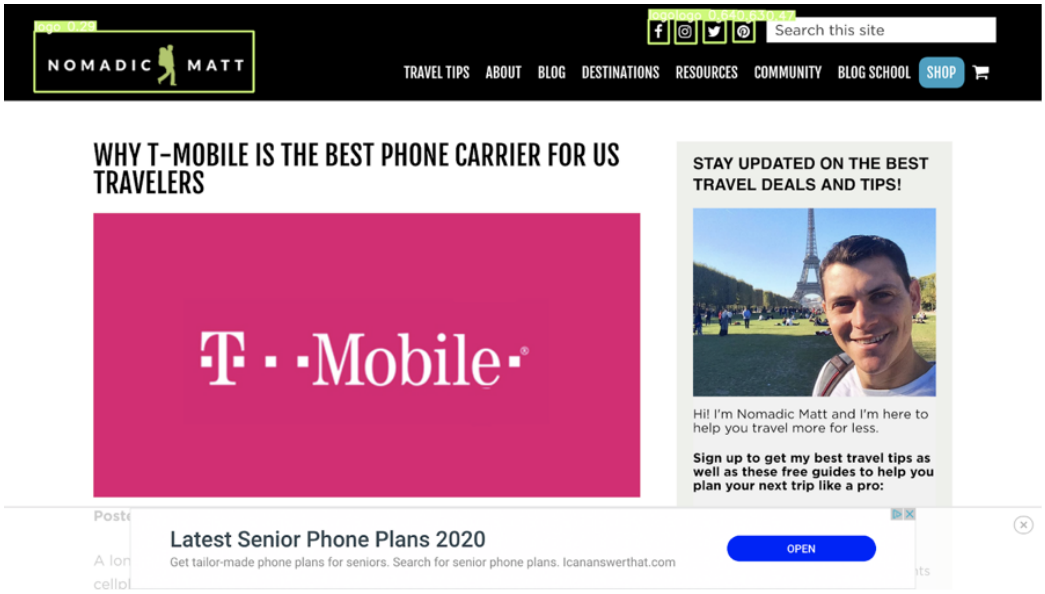


**Figure 31**: YOLO v3 false negative on large logo.

### 4.1.4 Model Comparison and Error Analysis

In contrast to performance comparisons in the machine learning literature, where Faster R-CNN tends to outperform YOLO v3, in our experiments YOLO v3 and Faster R-CNN delivered very comparable performances, with YOLO v3 having slightly better performance in our final models. At an IoU of 0.5, our final YOLO v3 had higher adjusted precision than Faster R-CNN at most recall values. The precision for Faster-RCNN was only higher than that of YOLO v3 at very high values of recall. See Figure 32 for adjusted PR-curves of YOLOv3 and Faster R-CNN.

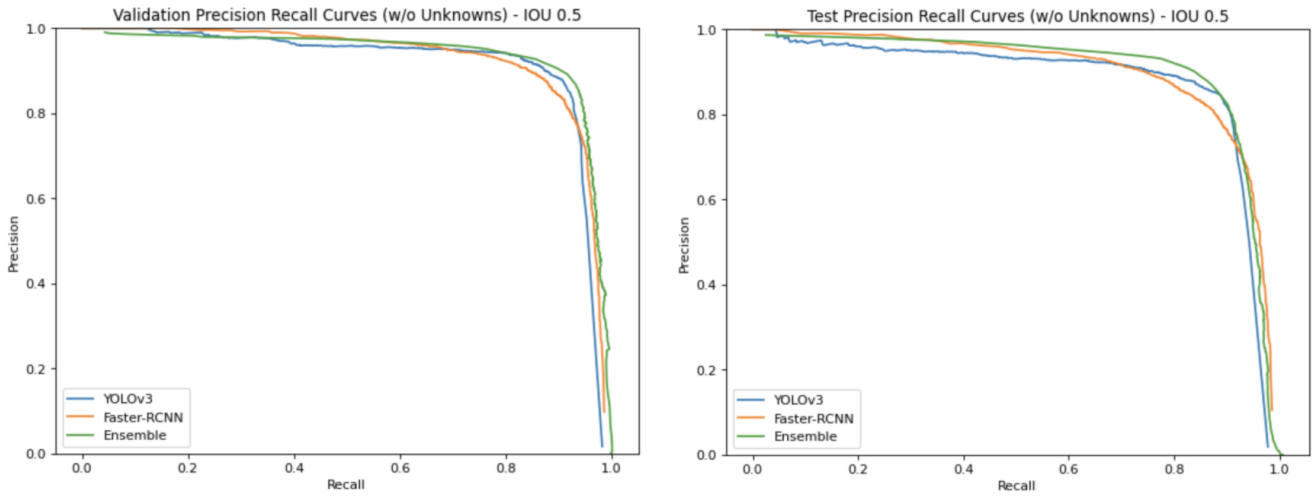


**Figure 32**: Open-Set adjusted Precision-Recall curves for YOLO v3, Faster R-CNN and their ensemble.

Compared to YOLO v3, Faster R-CNN suffers from an overeagerness to identify logos. On a data set of webpage screenshots with no logos, adjusting for false positives removed due to open set learning, Faster R-CNN erroneously found logos on 11.8% of images, finding 1.25 logos on average on the web pages where it detected logos. In comparison, YOLO erroneously found logos on 5.9% of images, finding 1 logo on average on the web pages where it detected logos.

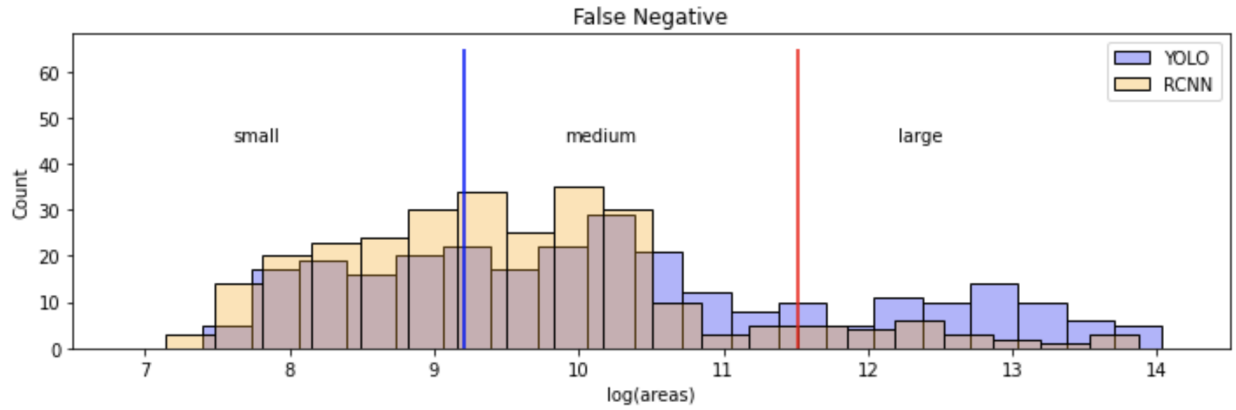


**Figure 33**: Histograms of False Negatives for YOLO v3 and Faster R-CNN at fixed recall of 0.87. Small logos at 10,000 pixels or smaller while large logos are 100,000 pixels or larger.

In our experiments, Faster R-CNN particularly struggled with images containing large stylized text, often incorrectly identifying such text as a logo. At the same time, it missed some true logos that were composed entirely of stylized text. YOLO was more robust to errors on stylized text but struggled consistently with detecting large logos, generally larger than 100,000 pixels, mistaking them to be generic images and not logos. On the other hand, Faster-RCNN

tended to struggle more than YOLO on smaller sized logos. This is illustrated in the False Negatives histogram in Figure 33 which shows how Faster-RCNN gets noticeably fewer false negatives in the large logo category than YOLO v3, while the converse is true on the range of small to medium sized logos. YOLO was also less robust than Faster R-CNN on images with darkened or unclear logos, regularly missing darkened logos on many web pages.

It was clear empirically and through our experimental results that the false negatives Faster R-CNN and YOLO v3 got were often ***not*** the same suggesting that each individual model would perform more strongly if ensembled with the other. Figure 34 shows a heatmap of the false negatives of each detector. YOLO v3's heatmap shows a concentration of false negatives near the center of the image while Faster R-CNN's heatmap shows that its false negatives are more evenly spread out through the area of the training images. These heatmaps were generated using an IoU of 0.5 and a fixed recall of 0.87, meaning that the raw count of false negatives in each heatmap is equivalent for both models. The reason YOLO v3's heatmap is darker than Faster R-CNN's is because the logos YOLO misses are much larger than those Faster R-CNN misses so the heatmap shows a lot more overlap of missed logos, making it darker.

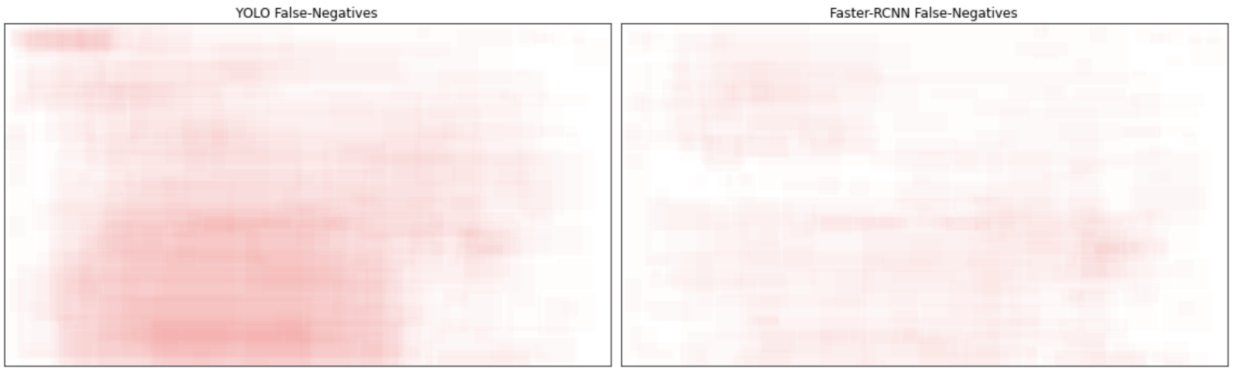


**Figure 34**: Heatmap of false negatives of YOLO v3 and Faster R-CNN at fixed recall of 0.87 on a 1440x2560 grid. YOLO's false negatives are more concentrated than those of Faster R-CNN and generally consist of larger logos than those of Faster R-CNN.

As shown in Figure 32 and Table 1, ensembling the YOLO and Faster R-CNN models led to noticeable gains in performance measured via precision and recall. At a fixed recall of 0.87 and IoU of 0.5, our ensemble achieves an adjusted precision of 0.93 on the validation sent 0.87 on the challenging test set. The ensemble method we chose for our experiments involved accepting all predictions from YOLO v3, and any predictions from Faster R-CNN that YOLO v3 had not made. As a result of the ensembling, we identified the main logo categories where both Faster R-CNN and YOLO v3 struggle, making them categories that our pipeline is particularly weak against.

| | Recall 0.80 | Recall 0.85 | Recall 0.90 |
|---|---|---|---|
| **Faster-RCNN** | 0.92 | 0.90 | 0.84 |
| **YOLO v3** | 0.94 | 0.92 | 0.88 |
| **Ensemble** | **0.95** | **0.94** | **0.90** |

**Table 1**: Comparison of precision of Faster R-CNN, YOLO v3, and their ensemble for multiple fixed recall values on the validation set.

The four categories of false negatives are shown in Figure 35. Category 1 has logos that are extremely small and/or distorted. Category 2 consists of logos that closely resemble icons. Category 3 consists of logos that are either highly stylized by design or have been stylized for special or one-off presentations. Finally Category 4 consists of small social media logos. Of these four categories, the only really detrimental category is Category 2 since we would want a well-trained detector to be able to correctly identify such logos. It is important to note regarding Category 4 that these are not 'true' false negatives; in most situations, the models have correctly identified these social media logos but as these logos are so small, their true bounding box often does not overlap with their predicted bounding box enough to meet the IoU threshold for a true positive, thereby creating a misleading category of false negatives. The three categories of false positives are shown in Figure 36. Category 1 consists of big stylized text, Category 2 consists of words that are displayed prominently (much like logos), and Category 3 consists of icons.


**Figure 35**: Ensemble False Negatives. Category 1: Small, distorted logos. Category 2: Logos closely resembling icons. Category 3: Highly stylized, one-of logos. Category 4: Small social media logos.


**Figure 36**: Ensemble False Positives. Category 1: Big stylized text. Category 2: Prominent words. Category 3: Icons.

# 4.2 Logo Classification

## 4.2.1 Classifier Overview

During our experiments, we tested several different models of both EfficientNetB0 and B3 architectures along with different data augmentation settings and a multitude of different classes. The following models present our final B3 models for a classifier trained on over 1098 classes and a classifier trained on just 32 classes plus an "unknown" class. This latter model was used to better illustrate the efficacy of the models on a "closed-set" and as a baseline comparison for open-set learning. Pretrained models were loaded from the timm library in Python, and these models had been pretrained on the ImageNet dataset. Images were resized to 300x300 through transformations because this is the input size required by EfficientNetB3. See Appendix A.1.2.2 for sample images after transformation. For reference, the main data augmentation steps are described as follows:

1. *Resize_and_pad(size = 300)*: A custom transformation that maintains the aspect ratio of the image based on the final desired size. Effectively, the image is rescaled so that its largest dimension is 300 pixels, and this rescaled image is pasted onto a white square of size 300x300. This ensures that images are not distorted when used for training or testing. See Figure A.1.2.1 in the Appendix to see the difference in images without maintaining aspect ratio during resizing.
2. *ResizeCrop(size = 300, scale = (0.5,1))*: Randomly crop a portion of the image and resize it to 300x300

3. *RandomRotation(degrees = 90)*: randomly rotates the image at a degree in a range between 90 degrees counterclockwise and 90 degrees clockwise
4. *ColorJitter(brightness = 0.25, contrast = 0.25, saturation = 0.25, hue = 0.5)*: adjusts the brightness, contrast, saturation, and hue of an image for additional color schemes
5. *Normalize(mean = (0.4914, 0.4822, 0.4465), standard dev = (0.2023, 0.1995, 0.2010))*: normalizes the images based on the mean and standard deviation of the ImageNet dataset since the model has been pretrained on ImageNet.

The primary model hyperparameters are described as follows:

| Batch Size | Learning Rate | Learning Rate Decay | Epochs |
|---|---|---|---|
| 32 | 1e-4 | 0.1 | 30 (decay at 15) |

**Table 2.** Hyperparameters for training the EfficientNet B3 model. These settings were used for all final B3 models described in the next few sections.

## 4.2.2 EfficientNetB3: 1098 Classes

Our primary EfficientNet model was trained on 1098 classes spanning companies from the S&P 500 and from popular internet, financial, technological, and industrial brands. In total, there were 12878, 4668, and 1165 images in the train, validation, and test sets respectively.

Learning curves for the models are shown in the figures below:

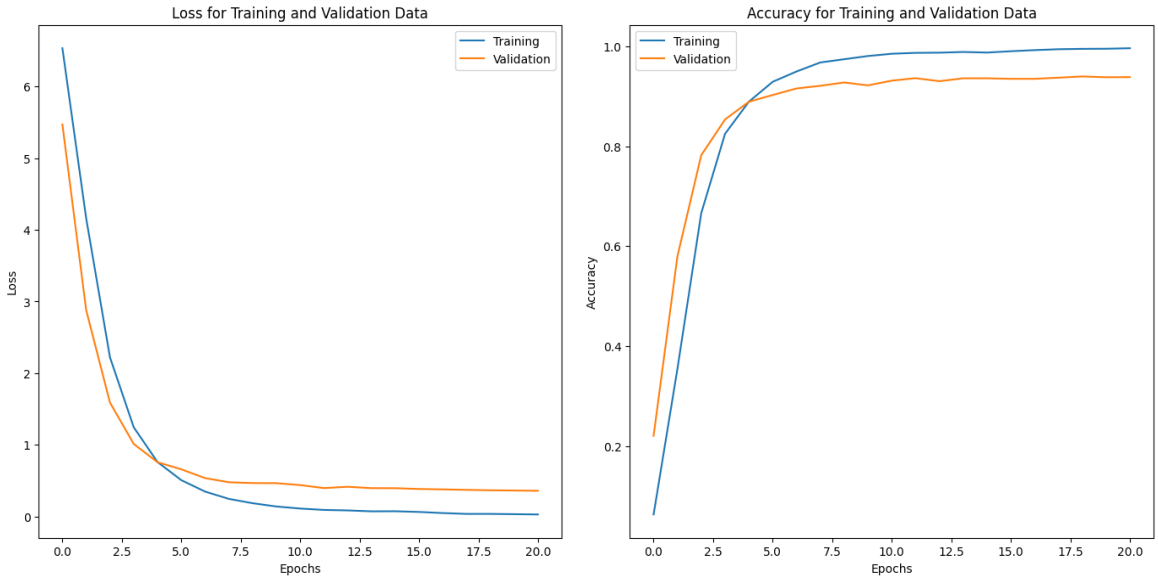


**Figure 37.** Performance curves for the EfficientNet B3 classifier. These curves show stable training, the model reaches maximum performance very quickly.

Table 3 shows the performance of this classifier on the test dataset. Note that not all classes had a prediction and many classes had fewer than 3 samples. Despite this, we can see very strong performance by the model overall on the test set. A more extensive summary of the model performance for test set classes with more than 10 samples is shown in APPENDIX A.1.3.1.

| Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|
| 0.9451 | 0.9825 | 0.9451 | 0.9583 |

**Table 3.** Test set performance of the EfficientNet B3 classifier trained on 1098 classes of logos. Note that not all classes had a prediction and over 90% had fewer than 5 samples.

Examples of correct and incorrect predictions are shown in Figure 38. In general, the model performs very well when it has seen particular logo variations of a specific class. For example, the model was trained on a variety of Facebook logos that included the main Facebook text, the small f logo, and those with some additional text such as “like” and “share.” There were also Facebook logos of different colors and shapes to add sample variety in addition to our data augmentation techniques. By combining images downloaded from Bing with those extracted from our annotations, we vastly increased both the number and variety of samples for many of our classes, improving model robustness.

However, the model is not without its faults. Logos of similar but different classes, such as MasterCard and MasterPass, can often be misclassified. In addition, logos extracted from a real-world image, such as on a sign or building, or with a text or style that differs from the standard logo can also be misclassified. There are also classes for which the model has seen very few if any samples of a particular logo variation making it harder to discern new samples. Still, as shown by our model performance metrics, these cases are rare, and the classifier is robust overall. Additional improvements can be made by obtaining more test samples for more effective quantification of performance as well as fine-tuning the number and type of classes. There are several classes that may not be of interest, such as foreign payment systems or obscure internet brands, and better selecting these classes may help slightly boost performance.

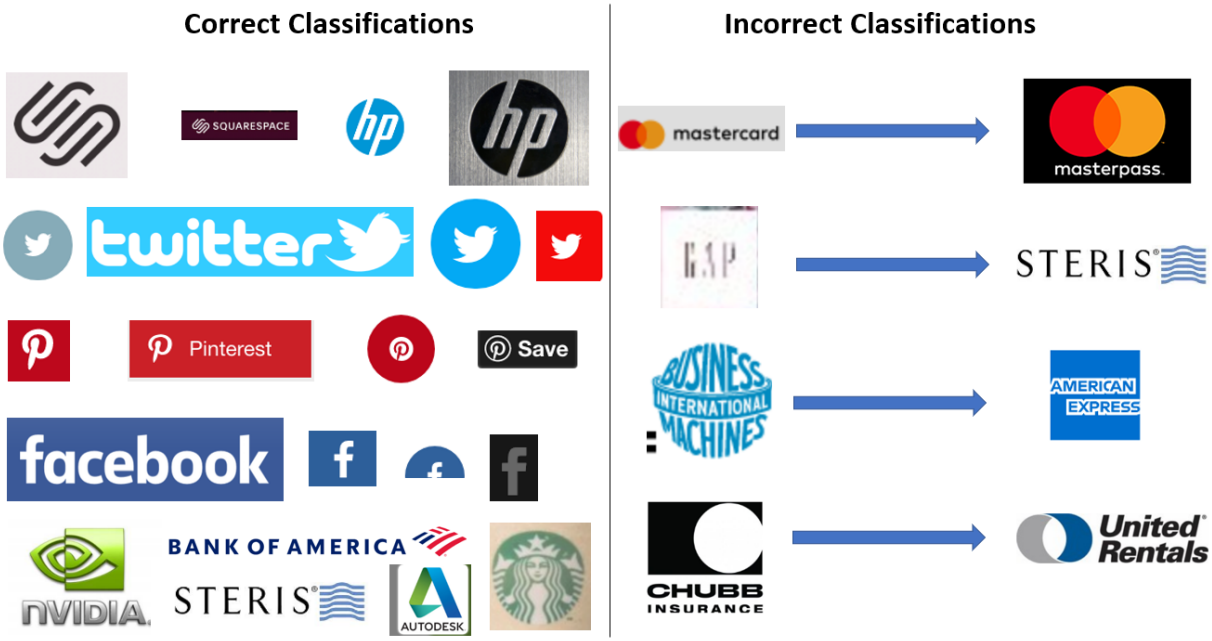


**Figure 38.** A comparison of test set logos correctly and incorrectly classified. The left column shows the variety of logos classified both between samples and within samples. Logos of the same brand but different color, such as Twitter, were classified correctly, as were real-world image logos such as that of Starbucks. The right column shows incorrectly classified samples. The left portion in this column shows the test image, and the right portion shows the brand assigned to the image. For example, in the top row, the MasterCard image was classified as MasterPass instead. Likewise IBM was misclassified as American Express.

### 4.2.3 EfficientNetB3: 32 Classes + Unknown Class

The 32 known classes selected for this model were chosen if they had more than 5 images in the test dataset. Using these classes would present a more refined outlook on the model's performance. In addition, since it is impossible to train on every logo, we added an "unknown" class to this model as a catch-all for samples that were not part of the main 32 classes. Effectively, we want the model to recognize a logo it has not seen before, and adding an unknown class enables us to see how well the model performs on such samples. Figure 39 depicts examples of images added to this unknown class. These images were randomly sampled from our "generic_logo" annotated class as well as from false positives generated by our YOLO detector model on the training annotation set. These images were then split into train and validation datasets. The test dataset consisted of randomly sampled "generic_logos" from our annotated test data.


**Figure 39**. Sample images for the "unknown" class. The left portion shows images not represented by the 32 known classes, such as Cision and Brita, that were sampled from the "generic_logo" class. The right portion shows generated false positives from running the YOLO model on the annotation datasets.

After adding the unknown class, the model was retrained on the dataset with the same model parameters as described in section 4.2.1. We also trained a model with the exact same augmentation and hyperparameter settings but without the unknown class to compare the two models. The results are shown in Table 4.

| | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| **Without Unknown** | 0.9518 | 0.9645 | 0.9518 | 0.9537 |
| **With Unknown** | 0.9210 | 0.9291 | 0.9210 | 0.9224 |

**Table 4.** Test set performance of models trained with and without an unknown class. As expected, adding the unknown class, which has high variation, led to overall worse performance. However, in both cases, performance is still quite strong on the test dataset.

As expected, having an unknown class with a large variety of samples worsened performance slightly compared to not having the unknown class. However, this model still performed very well and had both a precision and recall of 84.4% on the unknown class. Many completely new samples were correctly categorized as unknowns. A normalized confusion matrix of the model's performance is shown in Appendix A.1.3.2. Figure 40 depicts examples of test "unknown" images that were both correctly and incorrectly classified.


**Figure 40**: Correct and incorrect predictions of unknown samples. The left portion shows test set images correctly classified as "unknown." The right portion shows images in the unknown class and indicates what they were classified as by the model.

While adding an unknown class to the model was successful in identifying samples that the model had not been trained on, having this class is still not the optimal approach for this problem because we not only want to recognize unknown samples, but we also want to be able to add new classes to the dataset with minimal or no retraining of the main model. To better solve these two issues simultaneously, we turned our focus toward open-set learning.

## 4.3 Open-Set Learning

As mentioned earlier, Open-Set learning refers to the ability of the model to respond accurately to previously unseen logos. It is practically impossible to train our classifier on all the logos in the world, while it is almost imperative that at some point our deployed model will encounter a logo it has never seen before. How will the model react to this logo? Will it correctly classify it as an unknown logo, or will it misclassify it into one of the existing classes? In this section, we present the results of two approaches we took to tackle the open-set learning problem - the K-Nearest Neighbours approach and the Open Long Tail Recognition (OLTR) method.

Before we present the results, we would like to provide a brief description of the metrics used to compare the different methods.

I. **Correct Unknowns** - Percentage of *unknown* logos correctly classified as unknown.
II. **Incorrect Unknowns** - Percentage of *known* logos incorrectly classified as unknown.
III. **Open-Set Accuracy** - Accuracy of the model on known logos when trained under the assumption that it would be deployed in an open world.

### 4.3.1 Open-Long-Tailed-Recognition (OLTR)

For the OLTR method, we used code from a repository that is maintained by the original authors of the paper **[5]**. In order to execute their code, our logos were split into four different sets - a training set, a validation set, a test set and an open set. The training, validation and test sets only consisted of known logos belonging to 32 companies while the open set consisted of a random collection of generic, unknown logos that the model wasn't trained on.

Training the model had two stages (Figure 41). Stage 1 involved using a ResNet 152 architecture with a categorical cross-entropy loss function to build a simple closed-set classifier on the training set. The model was not exposed to any unknown logos at this stage and Stage 1 is in many ways identical to the logo classification stage in our pipeline using EfficientNet B3. The second stage of our pipeline involved generating meta-embeddings to train a classifier. Obtaining the meta-embeddings requires us to create the visual memory (Figure 11) first, which is in many ways similar to a K-Means clustering algorithm. The loss function used here is known as a Discriminative Centroid Loss which consists of two opposing terms - an attracting loss and a repelling loss. Simply put, the attracting loss minimises the intra-cluster distance which allows similar logos to be clustered together, while the repelling loss maximises the distance between centroids to keep clusters of dissimilar logos far apart from each other. Meta embeddings simply refer to linear combinations of the direct features($v_{direct}$) from an image with some combination of features from the visual memory ($v_{memory}$) (Figure 11). Creating such linear combinations of the features allows this method to create a more granular and nuanced view of logos allowing us to distinguish known logos from unknown logos during test time.

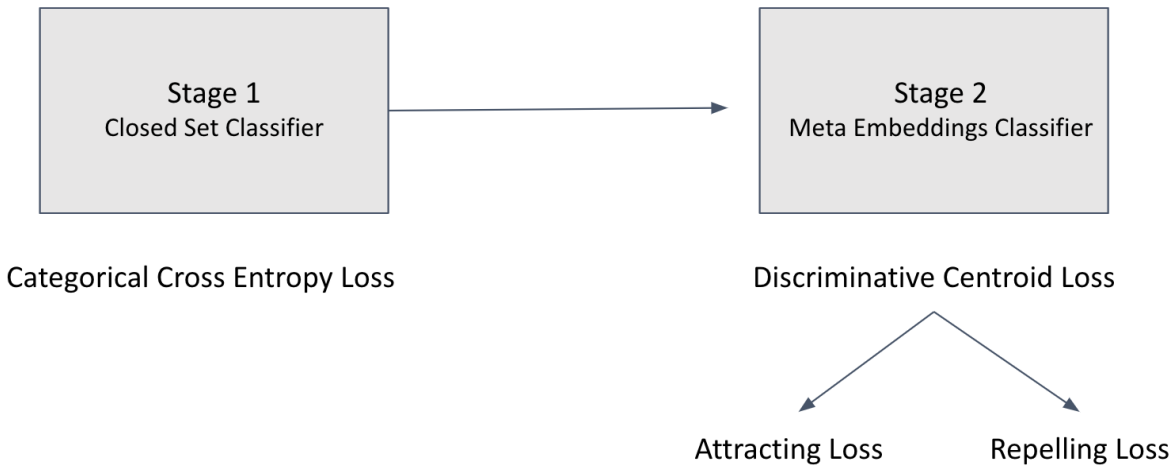


**Figure 41.** Procedure for training the OLTR method. Note the two stages involved and the unique loss functions used.

### 4.3.2 K-Nearest Neighbors

For the KNN method, we developed our own codebase from scratch to implement the method at training time as well as to prepare the Lookup Table and tuned thresholds from user provided datasets. In addition, we developed functionality for users to add any number of additional classes to the Lookup Table while utilizing previously calculated results where possible to minimize new computations.

In our experiments, we worked with all three types of activation vectors described in section 2.4.2 and found that just using the second-last layer activation vectors delivers the best open set performance. On our 32 class data sets, the KNN method correctly identified 92.8% of unknowns while only calling 9.1% of known logos unknown, resulting in an open set accuracy of 90.6%. Similarly, on our 1098 classes data sets, the KNN method correctly identified 78.1% of unknowns while only calling 5.9% of known logos unknown, resulting in an open set accuracy of 91.9%.

In our experiments, the KNN method also proved to be impressively robust to the addition of new classes, showing few misclassifications even though the underlying neural network had not been trained on these new classes. Table 5 shows how the addition of new classes impacts open set performance for the KNN method. As the number of additional classes increases, the number of incorrect unknowns on the classes used during neural network classifier training increases slowly, with only 1 additional incorrect unknown after the addition of 15 new classes. For the newly added classes, there are 11 incorrect unknowns after the addition of 15 new classes. Similarly, there are no additional new incorrect knowns on the classes used during neural network classifier training after the addition of 15 new classes while there are 7 new incorrect knowns associated with the 15 new classes. These results suggest that the KNN method is well suited for use in open set classification scenarios where new classes need to be added at regular intervals without compromising on open set performance.

| | Additional Incorrect Unknowns (False Negatives) | | Additional Incorrect Knowns (False Positives) | |
|---|---|---|---|---|
| **Additional Classes** | **Trained Classes** | **New Classes** | **Trained Classes** | **New Classes** |
| None | 64 / 706 | - | 13 / 180 | - |
| 0 → 5 | 65 / 706 | 2 / 13 | 15 / 180 | 2 / 180 |
| 5 → 10 | 65 / 706 | 5 / 26 | 14 / 180 | 7 / 180 |
| 10 → 15 | 65 / 706 | 11 / 42 | 13 / 180 | 7 / 180 |

**Table 5**: Incorrect Unknowns and Knowns as number of additional classes added to the KNN open set method. The Trained Classes columns correspond to incorrect unknowns and knowns among the logo classes used in the neural network training which the New Classes columns correspond to incorrect unknowns and knowns on the newly added classes, on which no neural network training is done. The fractional format represents: <number of incorrect unknowns> / <total number of test data logos in this category>

### 4.3.3 Comparison of OLTR and K-Nearest Neighbours

In this section we will attempt to compare and contrast the pros and cons of the OLTR method and the KNN method for open set learning. OLTR is one of the most sophisticated, state-of-the-art techniques for open set learning. It was vastly quicker than KNN both during training and test time as OLTR does not require the generation, storage and retrieval of a Lookup Table of activation vectors or the tuning of a set of thresholds. KNN on the other hand requires significant computation at the training stage to generate the Lookup Table and thresholds and must check every new observation against all Lookup Table entries during the testing stage. Despite highly parallelized GPU operations, this leads to a much slower processing time than OLTR.

One major downside of OLTR was that the code accompanying the paper was not very intuitive to use. Moreover, switching out the default ResNet 152 architecture for our trained EfficientNet B3 was much harder than we anticipated and was something we could not achieve during the timeframe of this project. On top of this, the addition of new classes for open set detection is less straightforward for OLTR than KNN, making it unsuitable for many open set applications.

| | Correct Unknowns | Incorrect Unknowns | Accuracy Open Set |
|---|---|---|---|
| **Unknown Class** | 84.4% | 4.0% | 94.5% |
| **OLTR** | 88.9% | 8.9% | 90.7% |
| **KNN** | 92.8% | 9.1% | 90.6% |

**Table 6.** Results of the open-set learning approaches.

As shown in Table 6, in our experiments, KNN outperformed both OLTR as well as prediction using a model trained with an explicit unknown class. KNN achieved the highest percentage of correct unknowns out of the three methods without exceeding the acceptable range of percentage of incorrect unknowns. To achieve a similar level of performance to KNN in terms of correct unknowns, OLTR gets 16.2% of known logos as incorrect unknowns i.e. 6.8% worse than KNN. Due to KNN's strong open set performance and the ease with which new classes can be added to it, we chose it as our open set learning method of choice.

# 5. Conclusions and Future Directions

The results of this work showcase the impressive performance of state-of-the-art deep learning and computer vision models in solving problems pertaining to cybersecurity. Our logo detection and classification system provides an effective tool to support efforts to combat brand impersonation. Our detectors and classifier achieved impressive performance and our open set learning system proved effective in solving the challenges that come with deploying a pipeline like ours in the open world.

Future work could entail testing the performance of our pipeline on real-world phished data. In addition, open set learning could be better optimized for logo classification by using the OLTR method discussed in this paper, specifically through incorporating architectures, such as EfficientNet, that were not explored during our open set learning experimentation.

# 6. References


1. 2017 Data Breach Investigations Report, 10th Edition. *Verizon*. https://www.phishingbox.com/downloads/Verizon-Data-Breach-Investigations-Report-DBIR-2017.pdf
2. Ellis, D. (2021, February 19). How to recognize and avoid phishing scams. Retrieved March 05, 2021, from https://www.consumer.ftc.gov/articles/how-recognize-and-avoid-phishing-scams
3. S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, Adv. Neural Inf. Proces. Syst. (2015), pp. 91-99
4. Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
5. Liu, Ziwei, et al. "Large-Scale Long-Tailed Recognition in an Open World." 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, doi:10.1109/cvpr.2019.00264.
6. Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).
7. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788)
8. Wang, J., He, Z., & Li, H. (2018). Small object detection survey paper.
9. Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*.
10. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; and Dollr, P. 2018. Focal loss for dense object detection. arXiv preprint arXiv:1708.02002.
11. F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," arXiv preprint arXiv:1511.07122, 2015.
12. J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan, "Perceptual generative adversarial networks for small object detection," in IEEE CVPR, 2017.
13. Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International Conference on Machine Learning*. PMLR, 2019.
14. Tan, Mingxing, and Quoc Le. "EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling." *Google AI Blog*. 29 May 2019. https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

# A. Appendix

## A.1 Logo Classifier

### A.1.1 Distribution of Samples (1098 Classes)

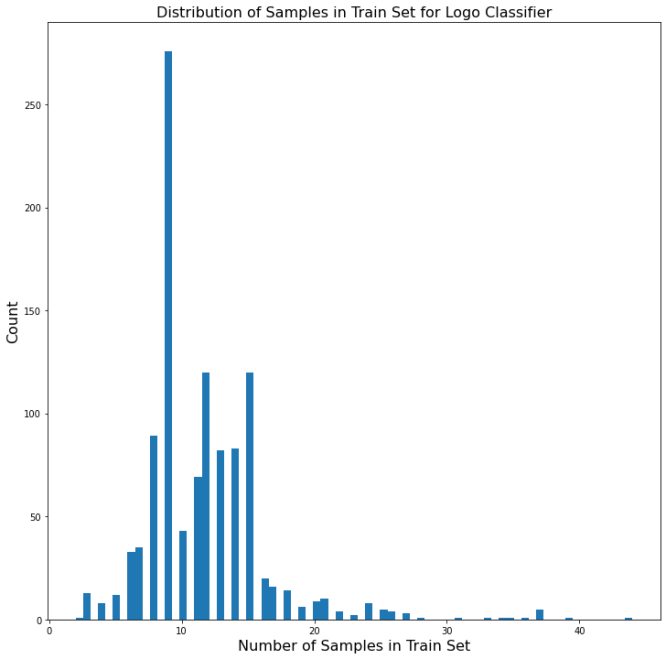


**Figure A.1.1.** Distribution of sample quantity for classes in the training set. Most images have between 8 and 15 samples while some more popular social media brands have over 35.

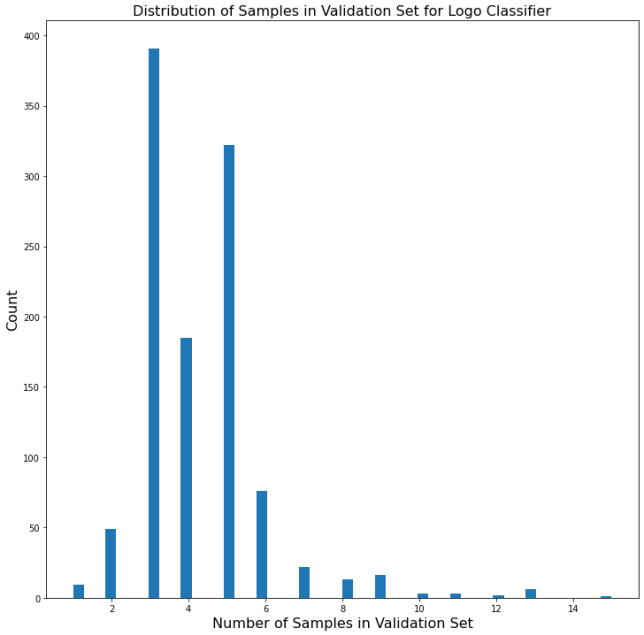


**Figure A.1.1.2:** Distribution of sample quantity for classes in the validation set.

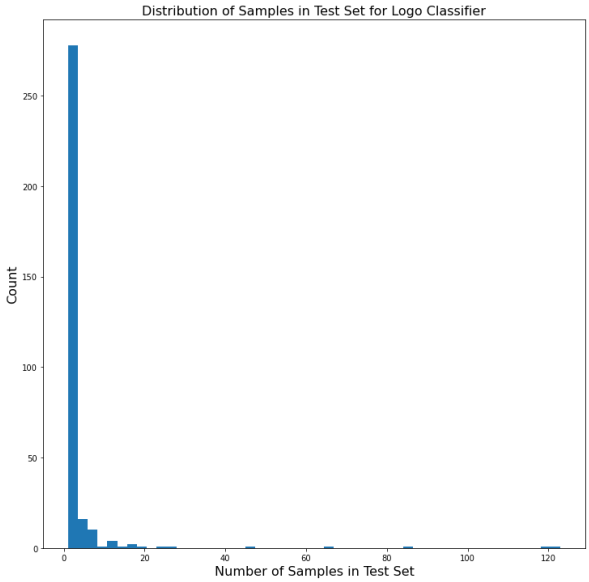


**Figure A.1.1.3:** Distribution of sample quantity for classes in the test set. Most brands had fewer than 5 images

## A.1.2. Data Augmentation


**Figure A.1.2.1:** Comparison of resized images in augmentation before (left) and after (right) maintaining image aspect ratio


**Figure A.1.2.2:** Example images augmented using techniques such as ColorJitter, RandomRotation, RandomResizedCrop, and custom resize class

## A.1.3. Classifier Results

| brand | precision | recall | f1-score | support |
|---|---|---|---|---|
| twitter | 1 | 0.9837 | 0.9918 | 123 |
| facebook | 1 | 0.9833 | 0.9916 | 120 |
| wikipedia | 1 | 0.9167 | 0.9565 | 84 |
| linkedin | 1 | 1 | 1 | 65 |
| pinterest | 1 | 1 | 1 | 45 |
| youtube | 0.9630 | 1 | 0.9811 | 26 |
| chubb_limited | 1 | 0.6250 | 0.7692 | 24 |
| steris | 0.9 | 0.9474 | 0.9231 | 19 |
| instagram | 1 | 1 | 1 | 18 |
| hp | 1 | 0.8125 | 0.8966 | 16 |
| sealed_air | 1 | 0.8667 | 0.9286 | 15 |
| fastenal_co | 1 | 0.9231 | 0.96 | 13 |
| nvidia | 1 | 0.9167 | 0.9565 | 12 |
| wikimedia_commons | 1 | 1 | 1 | 11 |
| google_plus | 1 | 1 | 1 | 11 |
| intel | 1 | 0.9000 | 0.9474 | 10 |

**Table A.1.3:** Per-class results on test set classes with more than 10 sample images. These results are from the classifier trained on 1098 classes.

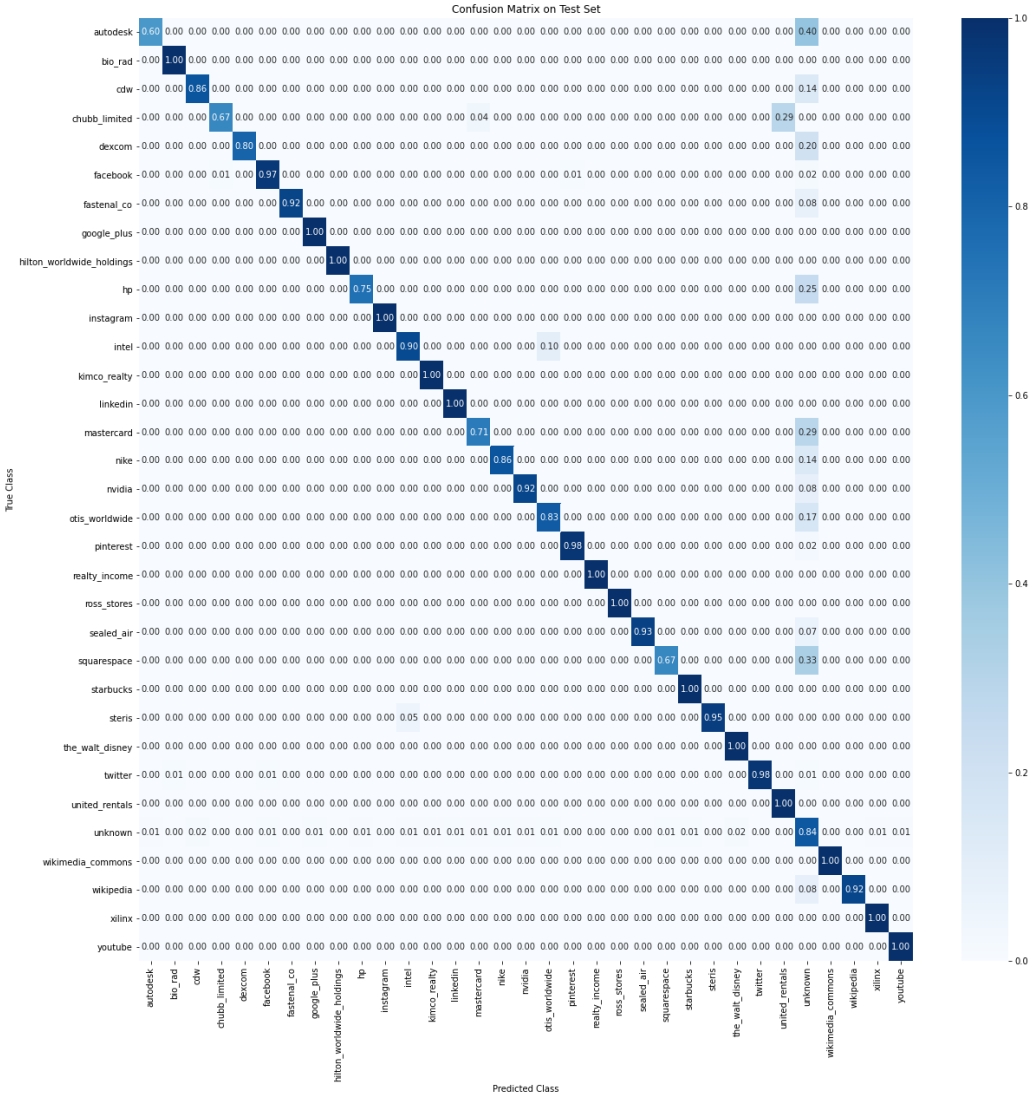


**Figure A.1.3:** Normalized confusion matrix of classifier on test set. This presents the results of the classifier trained on 32 classes plus the unknown class.