Contextualised Image Classifiers: Using User Tags to Classify Social Media Images

Kieran Litschel

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2020

Abstract

Mahajan et al. [25] and Yalniz et al. [41] demonstrate the benefit of pre-training on large unsupervised image datasets using user tags as labels for weak-supervision. We take this approach further, considering whether the use of user tags in large supervised image datasets is beneficial.

We propose modifying the teacher in the semi-weakly supervised approach proposed by Yalniz et al. to include user tags as features. To enable this we constructed a new dataset with user tags and human-verified labels, by combining the Open Images dataset [21] and Yahoo Flickr Creative Commons 100 Million dataset [37].

We were unable to combine user tags with image classifiers due to engineering challenges that we were unable to solve due to the time constraints. Instead we experimented with classifying images using only their user tags. Despite this, our model outperforms our naive baseline, and achieves a 0.1 higher average AP than our image classifier baseline for some individual classes, most notably ”animal”. This suggests that the use of user tags as features for image classifiers may help to improve performance for at least some individual classes, warranting further research.

Table of Contents

Chapter 1

Introduction

The most commonly used datasets for training image classifiers and object detectors use images originating from Flickr [23, 24, 29, 37], which is a social media website used to share images.

Users provide user tags with each image [37], to help others find their photos. They are typically succinct descriptions of the contents of the image, for example ”dog” is a frequent user tag. This information can help to understand the contents of each image, as it provides further context. But the majority of datasets discard them, and classifiers that do use them utilise them only as labels for pre-training [25, 41].

In this work we investigate whether utilizing user tags as features for image classifiers improves performance. We refer to such architectures as contextualised image classifiers.

1.1 Motivation

Our motivation for this work is to allow more accurate predictions for social media images. This has several applications.

It has been shown that training image classifiers on large image datasets enables state-of-the-art performance [25, 41].

But labelling these datasets is expensive. Large datasets like Yahoo Flickr Creative Commons 100 Million (YFCC100M) [37], which consists of 97 million images, are too large to extensively label by hand. Even smaller datasets like Open Images Datset (OID), [23] which consists of 10 million images, are still too large to label exhaustively by hand.

It is much cheaper to label large datasets using an image classifier trained on a smaller human-labelled dataset. In this method the image classifier labelling is referred to as the teacher, and the labels it generates are used to train a student network. This is referred to as a teacher-student paradigm. Hence if we can improve the performance of the teacher, allowing it to produce higher quality labels, we could improve the performance of the student.

Yalniz et al. [41] achieved state-of-the-art results in a teacher-student paradigm by pre-training the teacher using user tags as labels (this work will be discussed in detail in section 4.1.2). Our proposed architecture could replace the teacher in this work, which we hope would allow better performance.

Being able to classify social media images with greater confidence would also have applications for social media platforms such as Flickr, where the number of images users upload is too large for humans to moderate.

1.2 Challenges

There are some challenges with using user tags as features.

User tags do not usually list every object in the image, and objects listed may not be present in the image [19, 25]. In addition a lot of user tags do not describe the image, for example people will often include the year of the photo, such as ”2019”. This means there is a lot of noise in user tags

Flickr is used by people across the world, and as a consequence the user tags cover over a hundred languages [20]. Over 75% of the language is in English, which means there are lots of examples for English user tags, but there are many fewer examples for other languages. This makes understanding user tags challenging as although user tags may translate to the same meaning, it is harder to learn the meaning of user tags with fewer examples.

1.3 Hypothesis and Aims

We hypothesised that an image classifier with the joint knowledge of the image and user tags would outperform an image classifier that could only make predictions using images.

To test this hypothesis we aimed to:

1. Construct a dataset which contains user tags and human-verified labels for images.
2. Develop a feature extractor for user tags.
3. Develop a network that combined the features extracted from user tags with those extracted from images to classify the image’s contents.
4. Reduce the impact of user tag noise on classification.
5. Improve classification for under-represented languages.

1.4 Summary of the Work

In chapter 2 we briefly describe what image classification and object detection entails.

In chapter 3 we describe the datasets we used in this work, OID and YFCC100M. We conclude by joining the two datasets, creating a single dataset containing images with human-verified labels and user tags, which is the dataset we use for this work. We also discuss an open source library we created to allow others to join, process, and download the combined dataset.

In chapter 4 we discuss related work. This includes existing architectures that use user tags as labels, and others' analysis of the language in YFCC100M. This chapter is referenced frequently throughout the following chapters.

In chapter 5 we propose an image classification architecture and an object detection architecture which both use user tags and images as input. We discuss the engineering challenges that prevent us from implementing either. We then propose an image classification architecture that only uses user tags as input, which becomes the focus of the remainder of the work.

In chapter 6 we discuss existing metrics for object detection, including a metric specifically devised for OID. We propose a modification to the OID metric which we use to evaluate our image classifier.

In chapter 7 we propose two baselines to compare the performance of our model against. The first uses machine-generated labels for the test set to understand how an image classifier would perform. The second naively predicts a class is present in every image if it is most frequently labelled as present in the training set, and vice versa. This allows us to understand if our model performs better than a naive approach.

In chapter 8 we choose the hyperparameters for our network and experiment with preprocessing.

In chapter 9 we experiment with pre-training the best model found in chapter 8 using the remaining samples in YFCC100M that are unused in our dataset.

In chapter 10 we analyse the best model found in chapter 8 and compare it against our baselines models. We perform an analysis of the individual classes.

In chapter 11 we discuss experiments we started working on but did not have time to complete. These are further proposed solutions to the challenges introduced by language.

Chapter 2

Background on Tasks

This chapter introduces the tasks of image classification and object detection.

2.1 Image Classification

The goal of this task is to identify whether or not each class is present in the image [24]. In ground truth examples, binary labels identify which classes are present or not. An example ground truth image is shown in figure 2.1. Classifiers trained for the task take as input the image, and output for each label the classifier's certainty (a probability) that the label is present in the image.

Figure 2.1: Example of ground truth image with positive image classes [24]

2.2 Object Detection

The goal of this task to identify which objects are present in the image, and where each object is located in the image [24]. Locating the objects is referred to as localization. The location of objects in the image is identified using bounding boxes, which are described using the relative positions in the image of the corners of each box. Ground truth images are annotated with bounding boxes around each object, along with which class is present in each box. An example ground truth image is shown in figure 2.2. Classifiers trained for the task take as input an image and predict the relative coordi-

nates of bounding boxes in the image, which object is in each box, and a confidence (probability) for each box containing the predicted object.

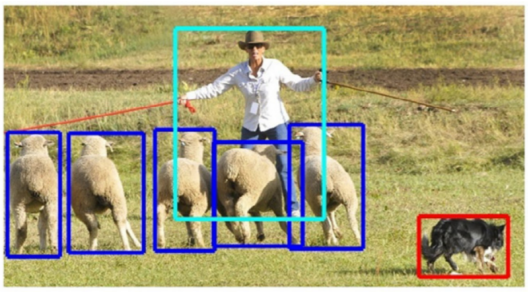

Figure 2.2: Example of ground truth image with bounding boxes. Cyan boxes contain humans, blue boxes contain sheep, and red boxes contain dogs [24]

Chapter 3

Datasets

In this chapter we discuss the dataset we used in this work. We start by introducing the two datasets we joined. We conclude by discussing why and how we joined them.

3.1 Open Images V5 Dataset

The Open Images V5 dataset [21, 23] (abbreviated to OID) consists of all Flickr images that had a CC-BY license as of November 2015. It contains 10 million images, with 6 million images annotated with ground truth image-level labels indicating which classes are present in the image, and which are not. 2 million of these images are also annotated with ground truth bounding boxes for objects and segmented objects. An example of an image annotated with image-level labels and bounding boxes can be seen in figure 3.1. The purpose of having a single dataset covering image-level detection, object detection, and object segmentation is to enable cross-task training.

3.1.1 Labels

There are 20,000 image-level classes in the OID dataset, with 9,000 trainable classes. These cover a broad range of concepts from events to objects, some of the most frequent image-level labels are summarised in figure 3.2. These classes were selected as a subset of the internal JFT dataset [33], which is a much larger dataset consisting of 300 million images with 375 million noisy labels.

It would be impractical for humans to exhaustively label OID given the number of images. Instead a few classes are identified as present (positive) or not present (negative) in each image, the statistics on these are summarised table 3.1. These labels form the human-verified labels. The process of choosing which image's classes should be verified by humans was automated using an ensemble of networks trained on JFT. The ensemble produced a probability for each class being present in each image. Google Annotators were asked to annotate the presence of classes in images sampled across the full range of probabilities. When sampling which labels to verify across the probabilities, preference was given to those with lower image id's, and consequently images with low id's have many more labels than those with high id's.

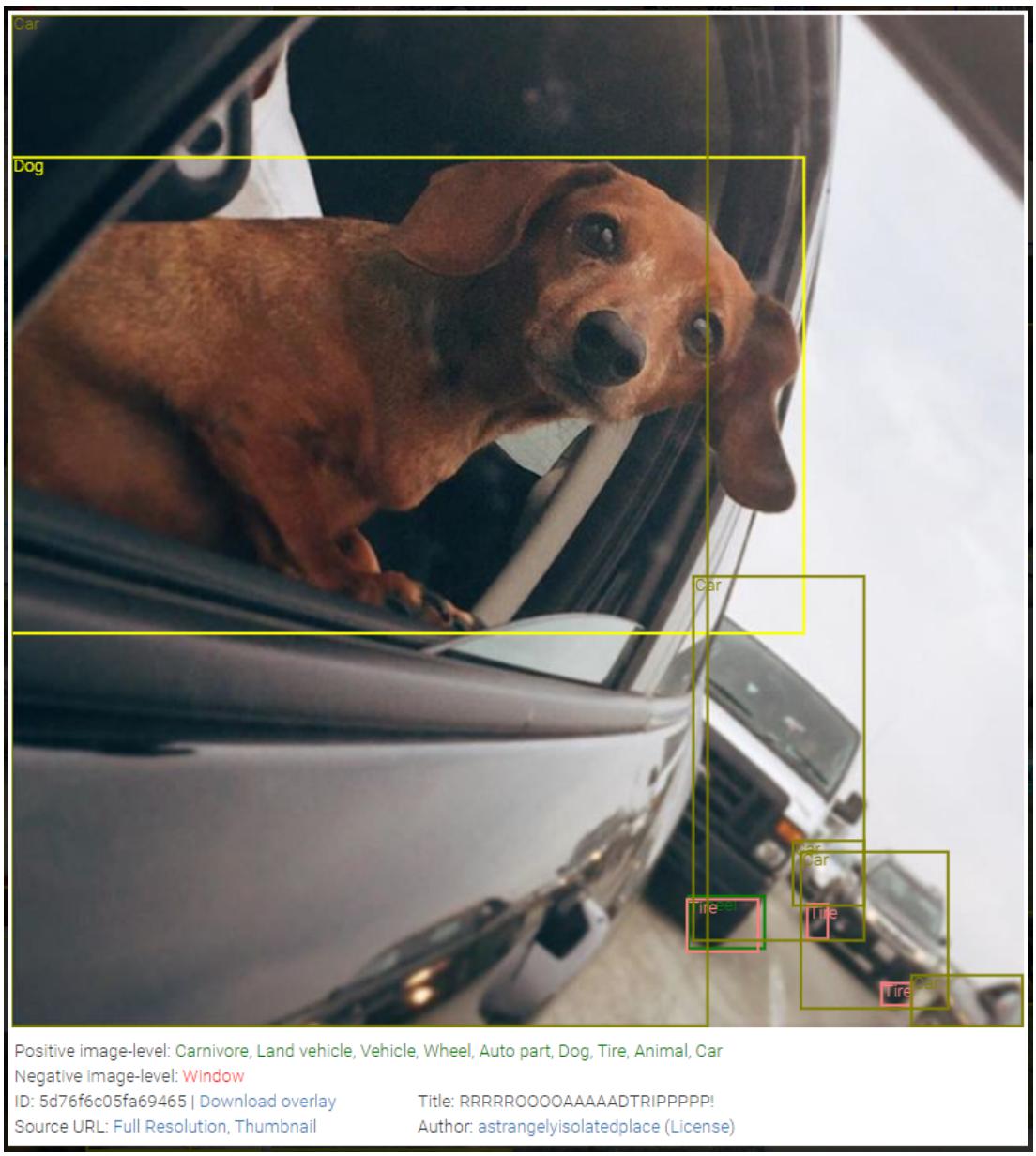

Figure 3.1: Example of image from OID with human-verified image-level labels and bounding boxes [1]

The dataset also contains machine-generated labels, which were generated by an Inception Resnet V2 [35] trained using the JFT dataset. Only machine-generated labels for each image with a confidence greater than 0.5 were included in the dataset.

There is large variation in the number of positive and negative labels in the dataset for each class, as seen in figure 3.3. For example humans dominate, with far more positive detections than negative detections than any other class. These variations occur as the 10 million images were collected without regard to their contents, hence the bias for the presence of classes on Flickr is reflected in the dataset.

600 image-level labels were chosen as object classes for the object detection task,

Figure 3.2: Most frequent image-level labels, the size of the words indicate their frequency [23]

	Train	Validation	Test
Images	9,011,219	41,620	125,436
Machine-Generated Labels	164,819,642	681,179	2,061,177
Human-Verified Labels	34,069,338 pos: 15,273,617 neg: 18,795,721	595,339 pos: 367,263 neg: 228,076	1,799,883 pos: 1,110,124 neg 689,759

Table 3.1: Number of images and labels for each subset for both machine-generated labels and human-verified labels for image-level classification [14]

chosen for their importance and being box-able. Like image-level labels, not every class of objects that are present in the image are annotated. The same process used to decide which image-level labels were annotated was used to decide which object classes to annotate in each image. The number of images and the number of boxes in each subset is described in table 3.2.

In cases where it is challenging to place a box around a single object as they occur in a group, then a box is drawn around the group, and a flag in the box's metadata indicates that the box contains a group. For example a packed bowl of apples would be annotated in this way.

The objects are arranged in a dendrogram ranging from coarse objects such as animals

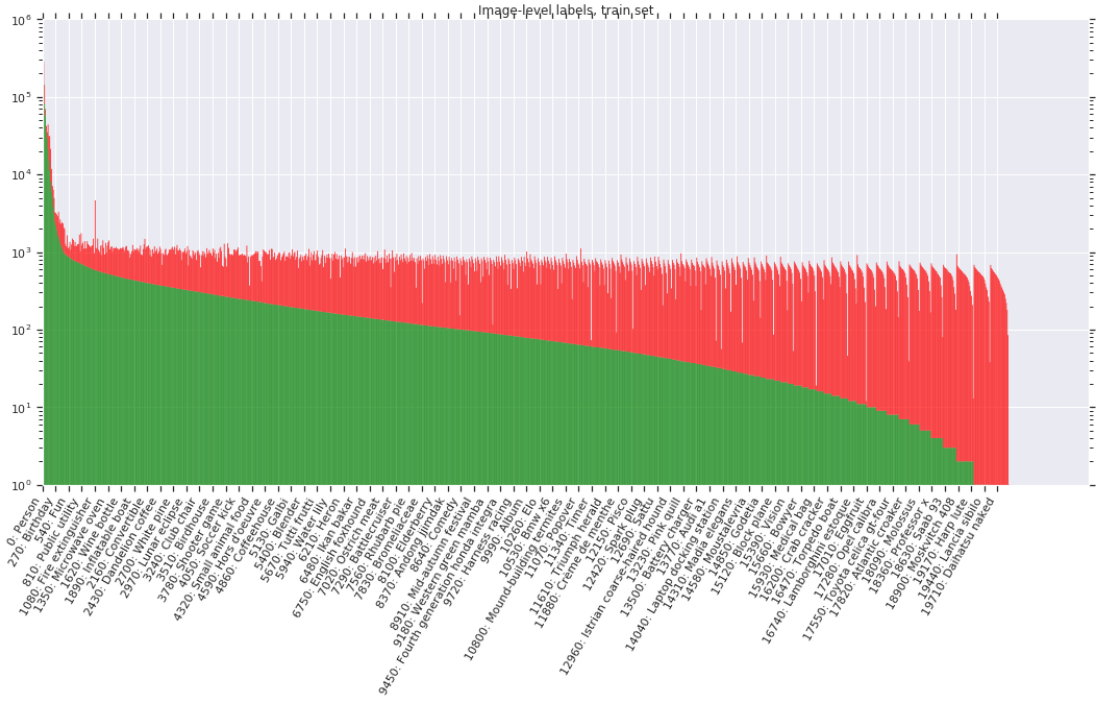

Figure 3.3: Distribution of positive (in green) and negative (in red) image-level labels for a subset of classes across training set [14]

	Train	Validation	Test
Images	1,743,042	41,620	125,436
Boxes	14,610,229	303,980	937,327

Table 3.2: Number of images and human-verified bounding boxes for each subset for object detection [14]

at the top of hierarchy, to fine-grain objects such as dolphins at the bottom. We use the subset of classes chosen for the OID challenge in 2019 [11]. In this subset 500 of the most frequently occurring classes are selected, and they are arranged in a different hierarchy to enable the use of the evaluation metric which we will discuss in section 6.2.2. The dendrogram of the 500 classes is shown in figure 3.4.

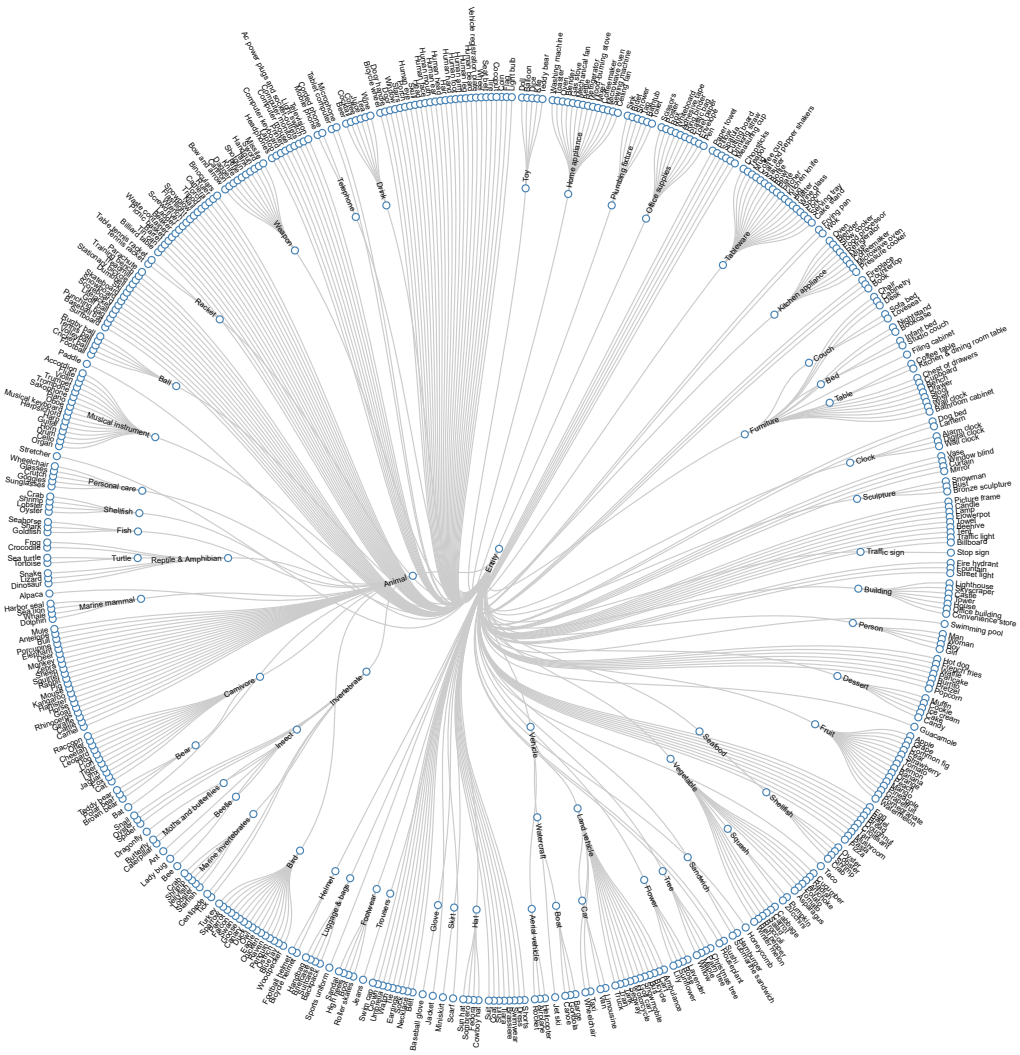

Figure 3.4: Dendrogram of 500 object classes selected for OID challenge in 2019 [11]

3.2 Yahoo Flickr Creative Commons 100 Million Dataset

The Yahoo Flickr Creative Commons 100 Million Dataset[1] [37] (abbreviated to YFCC100M) is a dataset of 99 million images and 800 thousand videos from Flickr that are licensed under the Creative Commons license. The capture and upload dates for the content span from 2000 to 2014.

3.2.1 Metadata

Each image is accompanied by 12 fields of metadata. Including fields describing the image's unique Flickr ID, user tags for the image, and the user's description and title for the image. Text fields are percent encoded, with the exception that spaces are replaced with plus symbols. The percent encoded user tags are comma separated.

[1] Yahoo Webscope - http://webscope.sandbox.yahoo.com

The dataset consists of 7,964,004 unique user tags, and 486,975,371 user tags in total. The distribution of the most frequent user tags is shown in figure 3.5. We also show the log frequencies of the user tags in figure 3.6 and observe that the user tags are Zipfian distributed.

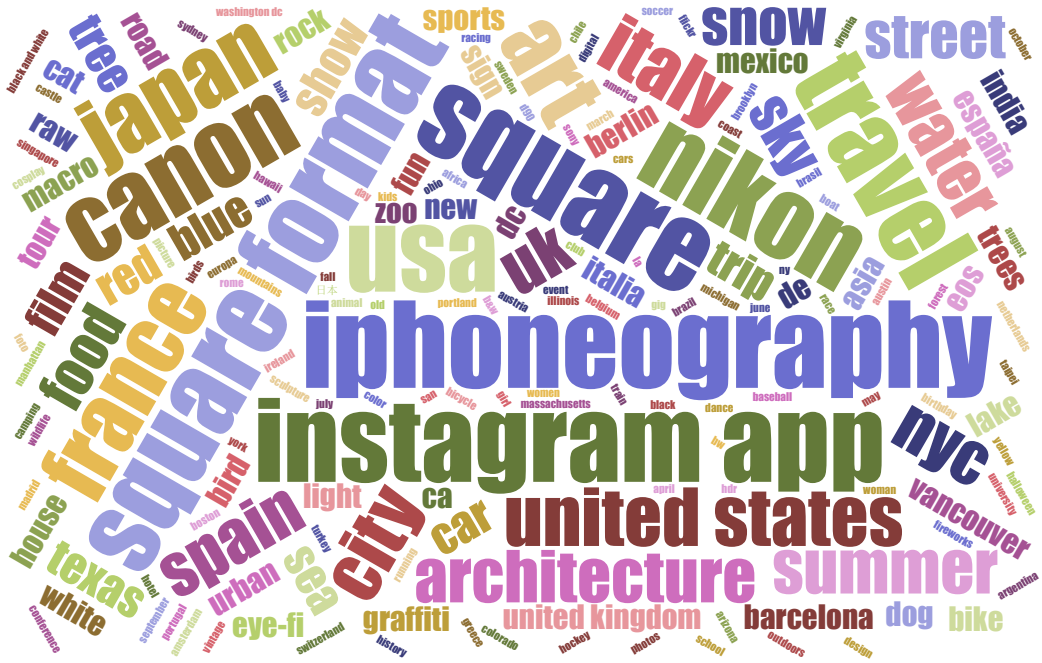

Figure 3.5: Most frequent user tags (excluding numbers), the size of the words indicate their frequency [37]

An example of a sample from YFCC100M is shown in figure 3.7, accompanied by its metadata.

3.2.2 Labels

There are no human-verified labels for the images. But machine-generated image-level labels are provided [36]. The labels cover 1,570 concepts, from objects to events. In total there are 905,407,562 labels in the dataset. The distribution of the most frequently predicted concepts is shown in figure 3.8. We also show the log frequencies of the labels in figure 3.9 and observe that the labels are Zipfian distributed. However, we note that the tail is quite short, likely a result of the dataset having a relatively small number of labels.

To predict the labels AlexNet [22] was used as a feature extractor for each image, with the output of the second to last layer used as features. A SVM was trained for each concept to predict its presence in each image from the features, giving a confidence score between 0 and 1. The SVMs were trained on a separate dataset of 15 million Flickr images with crowd-sourced labels. Only concepts for each image with a confidence greater than 0.5 were included in the dataset.

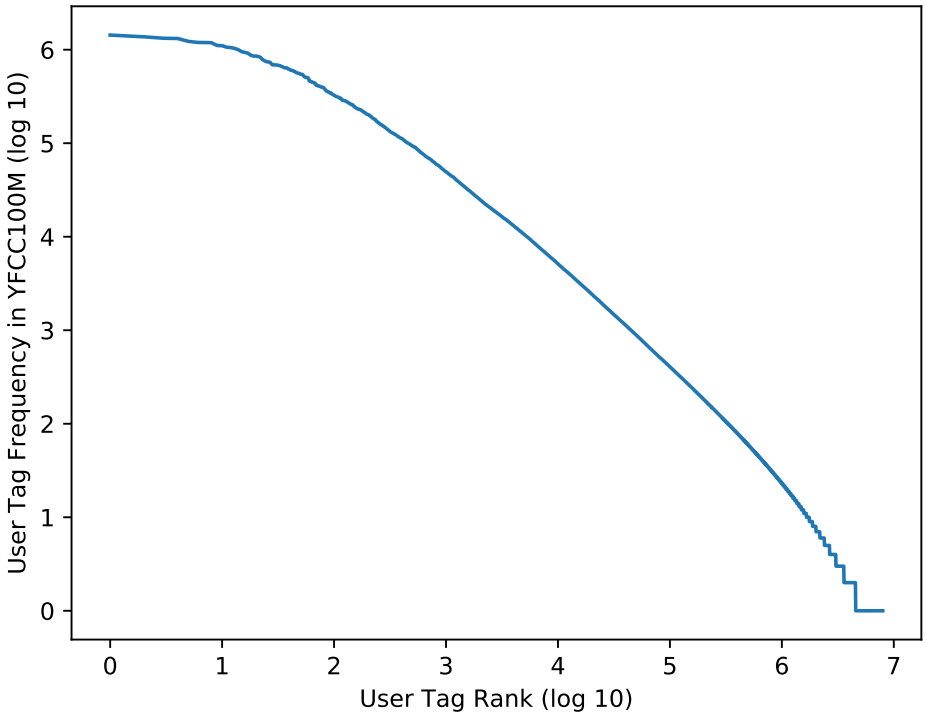

Figure 3.6: Log frequencies of all user tags in YFCC100M [37]

Figure 3.7: Example of sample from YFCC100M dataset [26]. Images metadata - ImageID: 8222923933, Title: Maya+and+Rudy, Description: Kharkiv%2C+Ukraine, User Tags: animals, dog, fun, maya, pet, siberian+husky, siberian+miracle+bird+of+paradise, ukraine. Predicted Concepts - outdoor: 0.958, nature: 0.928, animal: 0.92, screenshot: 0.783, text: 0.783, ethereal: 0.712, serene: 0.626, autumn: 0.609, foliage: 0.609, plant: 0.609, dog: 0.569, pet: 0.569, surreal: 0.568, safari: 0.562, jump: 0.544, romantic: 0.544. [37]

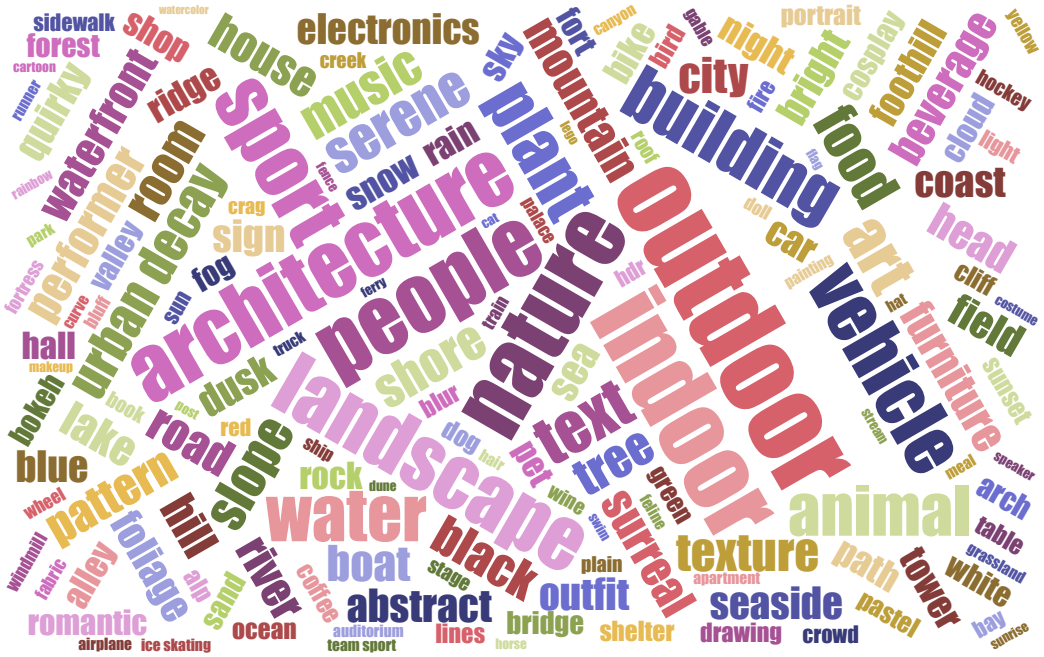

Figure 3.8: Most frequently predicted concepts, the size of the words indicate their frequency [37]

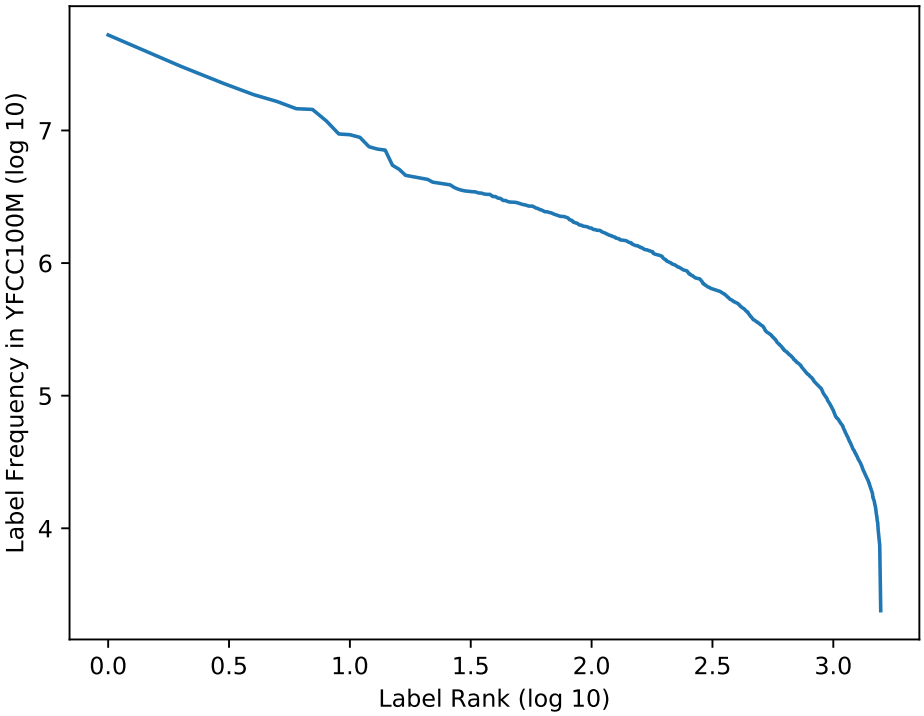

Figure 3.9: Log frequencies of labels in YFCC100M [37]

3.3 Combining OID and YFCC100M

3.3.1 Motivation

Conducting this work required a dataset that contained social media images with both human-verified labels, and their corresponding user tags. By combining OID and YFCC100M by matching the samples that corresponded to the same image on Flickr, we created a dataset that met these requirements. To our knowledge no one had attempted to combine the two datasets before, but it seemed likely the datasets overlapped given they both consist of images from Flickr with Creative Commons licenses.

3.3.2 Methodology

Each sample in OID is accompanied by an URL to the picture on Flickr's website. The URLs follow a pattern which includes the image's Flickr ID. As mentioned in section 3.2.1, included with each sample in YFCC100M is the images Flickr ID. By extracting the Flickr ID for each sample in OID, we then could iterate over YFCC100M to determine if the dataset also described the same Flickr image. We then combined the information from both datasets for each sample where there was a match, creating a joined dataset.

As described in the Flickr API [8], all Flickr photo URL's take one of the three formats below.

```
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_[mstzb].jpg
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)
```

We observed that the Flickr ID follows the last forward slash in the URL, and ends at the next preceding underscore. Using this we were able to construct the following regular expression to extract the Flickr ID from the URL.

```
(?:.*?\/\/?)+([^_]*)
```

This regular expression could be simplified assuming that there are a fixed number of forward slashes in an URL, but we found in practice there were some URL's that had more forward slashes than specified in the documentation, and consequently settled on the above regular expression.

3.3.3 Results

We discovered there is a large overlap between the two datasets. We found that 20% of the 10 million images in OID were also described in YFCC100M, and of the 2 million annotated with bounding-boxes, 25% were also described in YFCC100M.

Given the Flickr ID we were also able to request the user tags for the remaining images through the Flickr API. But this is expensive with there being a limit of 1 request per second. This meant gathering the user tags for the remaining 8 million images would

have taken 93 days. Instead we decided to prioritise collecting the user tags for the remaining images in OID's validation and test sets, which took 2 days.

The size of the combined dataset is described in table 3.3. We are unable to include 100% of the validation and test sets in OID as some images from OID have since been deleted from Flickr, meaning we are unable to retrieve their user tags through the API. For this work we have removed images from the dataset that do not have any user tags, so these aren't counted in this table. We discuss the reason for doing this in section 8.2.3.

	Train	**Validation**	**Test**
Images	1,867,841 (20.7%)	39,253 (94.3%)	118,301 (94.3%)
Machine-Generated Labels	34,830,690 (21.1%)	641,855 (94.2%)	1,940,704 (94.2%)
Human-Verified Labels	9,853,222 (28.9%) pos: 4,466,677 (29.2%) neg: 5,386,544 (28.7%)	561,243 (94.3%) pos: 346,115 (94.2%) neg: 215,128 (94.3%)	1,695,881 (94.2%) pos: 1,045,079 (94.1%) neg: 650,802 (94.4%)

Table 3.3: Number of image level-labels and images for each subset for both machine-generated labels and human-verified labels in the joined dataset. The percentage represents the proportion kept from the OID dataset (see table 3.1).

3.3.4 Open Images V5 Tools

As part of this work we built an open source library called Open Images V5 Tools. It implements the method described above to join the two data sets. In the combined dataset it includes all of the metadata from YFCC100M associated with the image, meaning it is a much more general purpose application than what we require in this work, and we hope it will enable other work to take advantage of the wide range of metadata YFCC100M has to offer. As a result we decided keep the functionality to extend the dataset through the API separate, as this only extends it for user tags, which does not fit our desire for the library to be general purpose.

We added additional functionality inspired by open_images_downloader [6] and OIDv4 ToolKit [39]. Our library offers functionality to download the photos in OID from Flickr, with the ability to download photos in parallel to take advantage of high speed internet connections. The MD5 hash for each image is included in OID, and we use this to validate that the image we downloaded is the one expected. If not we retry the download up to 3 times, and otherwise we remove it from the dataset. It allows the user to specify which classes of images they want to download, and what percentage of the training, validation, and test subsets they would like to download, to prevent them having to download the whole dataset unnecessarily.

We also added functionality to resize the images. This is important as the 2 million uncompressed images with bounding boxes are 6.2 TB alone [34]. This is too large for most users (including us) to process and store. Our solution was to resize the images using aspect ratio resizer [28]. This method resizes images maintaining their aspect ratio such that its shortest dimension is a maximum of 600 pixels long, and its longest is 1024 pixels. This is the same resizing method used by the Inception network pre-trained on OID [13] which we will discuss further in section 5.5.3. This means that the compressed images can still be used with the pre-trained model without interference. This approach is effective, with the 500,000 images in the joined dataset being compressed to 32GB, a reduction in size of 4700%.

Chapter 4

Related work

We start this chapter by discussing research that inspired this work which used user tags as labels for image classification. We conclude by discussing others' analysis of languages in YFCC100M, which identifies challenges in understanding natural language in the dataset. This chapter is referenced frequently throughout this work.

4.1 User Tags as Weak-Supervision in Large Unsupervised Image Datasets

4.1.1 Pre-Training on User Tags

Mahajan et al. [25] introduced using large unlabelled social media datasets with user tags for image classification. Their work used the ResNeXt [40] architecture, which is a modified ResNet [16] with grouped convolutional layers. They took a dataset of 3.5 billion public Instagram images and pre-trained the network to predict the user tags from the images. They then fine-tuned their network on ImageNet, which allowed them to achieve state-of-the-art performance on the ImageNet 1k image classification task with single crop top-1 accuracy of 85.4%.

They investigated the effect of user tag noise on the classifier. This is important as there is a lot of noise in user tags, with sometimes user tags present being irrelevant to the contents of the image, and the problem that user tags relevant to the contents of the image are not always present. In their experiments they randomly replaced $p\%$ of the user tags in the dataset with user tags sampled from the distribution of user tags across the dataset. They pre-trained the network on the noisy datasets, and then finetuned the feature classifier on ImageNet. They experimented with several different thresholds for p, and the results are summarised in figure 4.1.

It is noteworthy that adding noise has very little effect on the classification accuracy of the network. This suggests that training on such a large dataset helps to circumvent the noise in the user tags.

The authors also investigated object detection, using ResNeXt pre-trained on user tags as a feature extractor and adding Mask R-CNN [15] for object localization and classi-

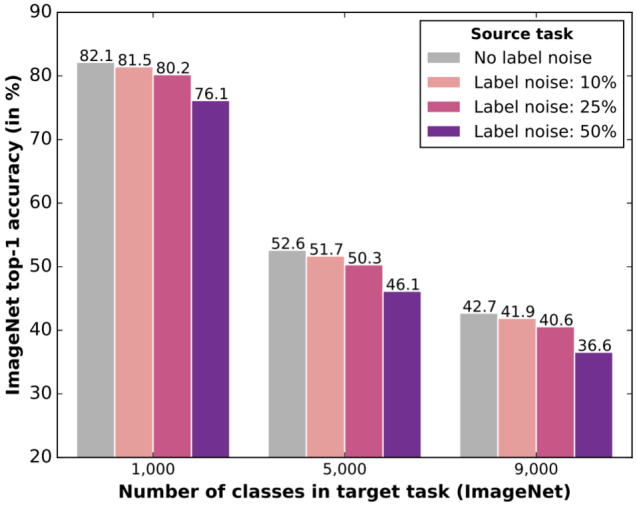

Figure 4.1: Effect of label noise on classification accuracy [25]

fication. They evaluated the network on the COCO dataset [24] using *mAP* (described in section 6.1). They used COCO's default *mAP* metric, which averages the *mAP* for the *IoU* thresholds 0.5 to 0.95, and also using Pascal VOC 2010 (only $IoU > 0.5$). COCO's metric places more emphasis on localization whereas Pascal VOC 2010 emphasises it less. They found that pre-training had the biggest improvement on Pascal VOC 2010, suggesting that pre-training is more beneficial to classification than localization.

4.1.2 Teacher-Student Approach

Yalniz et al. [41] investigated the application of semi-supervised learning to an unlabelled dataset U of a billion images.

They use a teacher-student paradigm, using a teacher model with more capacity, to train a student model with less capacity. For the teacher and student models they experiment with various architecture configurations based on ResNet and ResNeXt. The approach is based on distillation [17], with the upper bound performance of the student network being that of the teacher network, but the hope being that the knowledge of the teacher network is compressed into the smaller student network.

The teacher network was trained on ImageNet 1k, denoted D. This teacher network was then used to label U. For each class in D, they take the top-k most confident predictions for that class in U, and take them as positive training examples for the student network. The positive training examples for all classes form the dataset $\hat{D}$, which the student network is pre-trained on. One of the benefits of this approach is that the classes are balanced in $\hat{D}$, which solves the common problem in image datasets of long-tail distributions, like we saw in OID in figure 3.3. The student network is then fine-tuned on D. This pipeline is summarised in figure 4.2. They refer to this approach as semi-supervised.

Yalniz et al. also took inspiration from Mahajan et al. and explored pre-training the

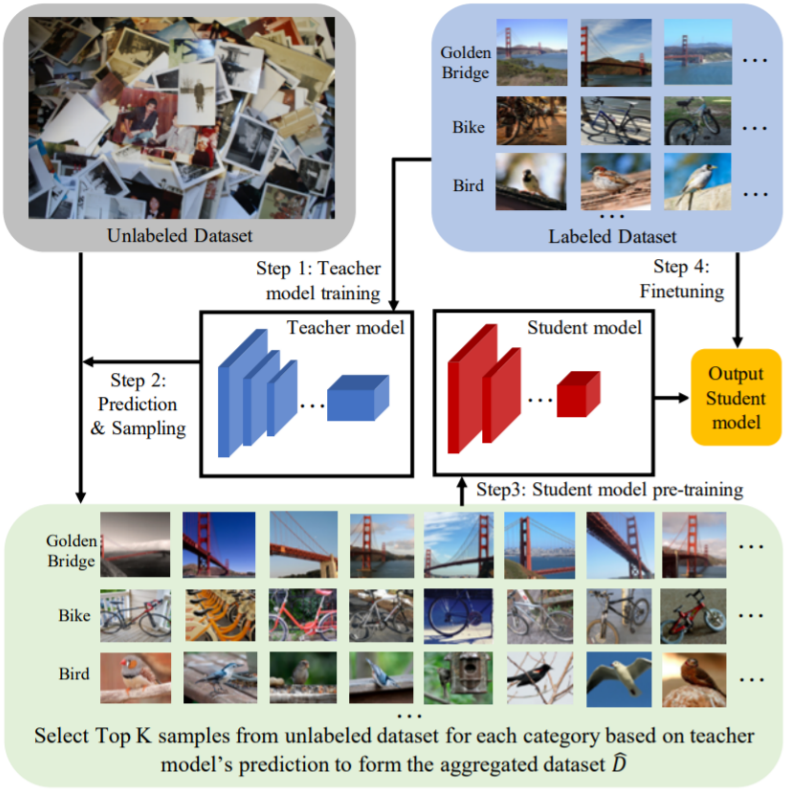

Figure 4.2: Semi-supervised image classification pipeline [41]

teacher for the semi-supervised approach on a dataset of 1 billion user tags. They refer to this approach as semi-weakly supervised.

4.1.3 Results

The results for the semi-supervised and the semi-weakly supervised approaches are summarised in table 4.1. Semi-weakly supervised at the time achieved the state-of-the-art, outperforming Mahajan et al.

	ResNet-50	ResNeXt-101-*		
		32x4	32x8	32x16
Xie et al. [40]	76.1	78.8	-	-
Mixup [42]	76.7	79.9	-	-
LabelRefinery [2]	76.5	-	-	-
Autoaugment [5]	77.6	-	-	-
Mahajan et al.	78.2	81.2	82.7	84.2
Yalniz et al. (semi-supervised)	79.1	80.8	81.2	81.2
Yalniz et al. (semi-weakly supervised)	**81.2**	**83.4**	**84.3**	**84.8**

Table 4.1: Performance of Mahajan et al. and Yalniz et al. against other state-of-the-art models [41]

4.1.4 Summary

In summary, Mahajan et al. and Yalniz et al. demonstrate the use of large unsupervised image datasets enabled state-of-the-art performance in image classification. Despite the noise in user tags, they proved promising for weak supervision due to the size of the datasets. When applied to object detection user tags appeared to only help improve image classification, not localization.

4.2 Natural Language in YFCC100M

Given the user tags, titles, and descriptions in YFCC100M, it can also be considered a corpus. Koochali et al. [20] conducted an analysis of the languages in the corpus. The corpus is multi-lingual with photos taken by people from across the world, as shown in figure 4.3.

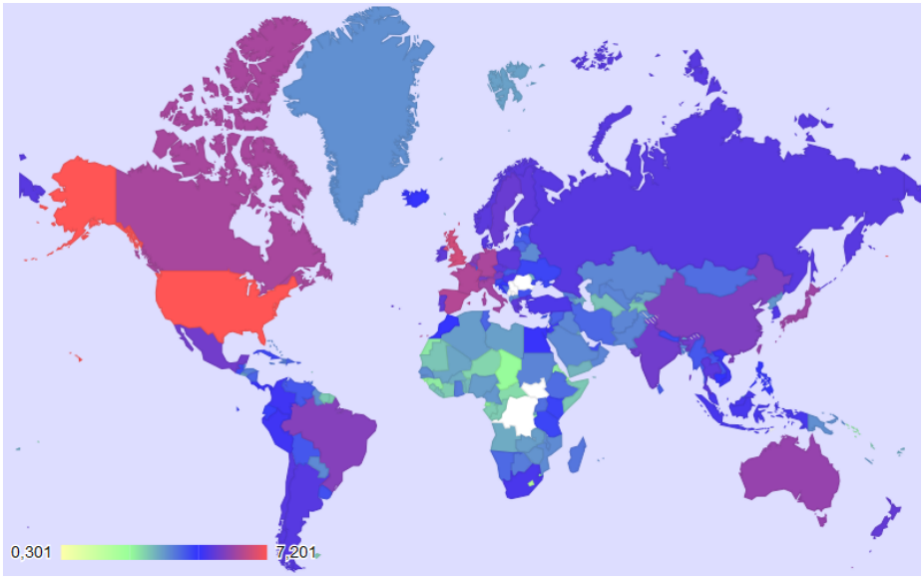

Figure 4.3: Distribution of where photos were taken in the world using a logarithmic scale. The location of where photos were taken was determined using their geolocation metadata included in the dataset. [20]

4.2.1 Languages

An analysis was performed of languages in YFCC100M using Compact Language Detector 2 (CLD2) [10]. CLD2 uses a Naïve Bayesian classifier to predict the most likely language of texts, and it supports 165 languages. For Latin characters it takes as input quad-grams of sequences of letters. Prior to converting language to quad-grams it is pre-processed, with punctuation removed, letters lower cased, and underscores added between words. For other character sets such as Han script, where a single character corresponds to a word or phrase, unigrams are used instead. The training data for the classifier was gathered from web pages, with a selection carefully selected for each language, and a further 100 million web pages automatically gathered.

A problem with this approach to language analysis is that the authors assume that CLD2 will work well for this use case. But titles, description, and user tags are often short, and it is stated on the CLD2 website that the detector does not work well for short texts. Hence this assumption seems weak. Nonetheless the analysis is still useful as it gives us a general idea of the distribution of languages in the dataset, but we should consider the results with caution.

In total only 62% of images are detected with having at least one language. It is likely that the language could not be detected for the remaining photos as too little text was provided in their metadata to make a conclusive detection. The languages detected are summarised in figures 4.4 and 4.5. We see that the corpus is dominated by English.

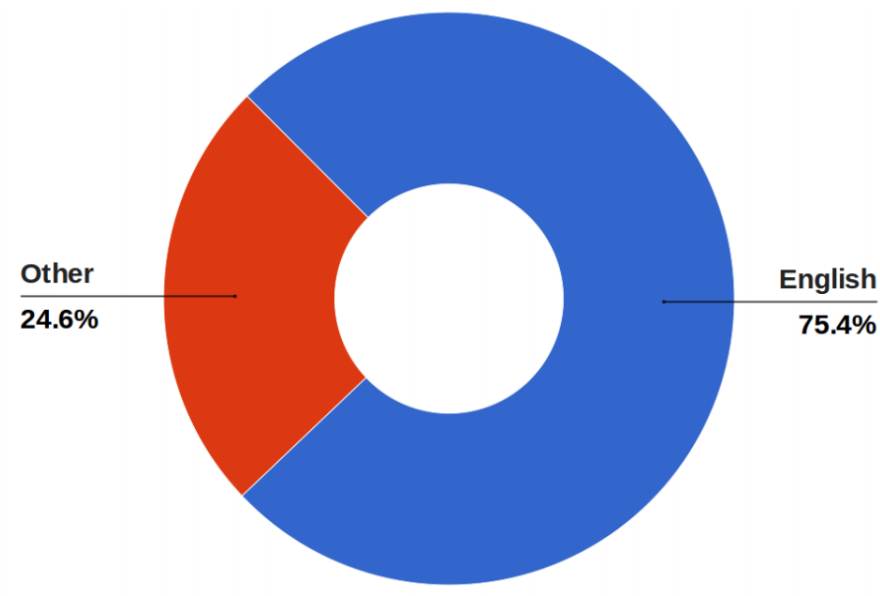

Figure 4.4: Percentage of images where English vs Other Languages was detected out of the 62% where a language was detected [20]

They also conducted an analysis of the languages for only user tags, and the results are summarised in table 4.2. We see that the majority of images have at least one user tag. Out of the images where the language of the user tags is detected, only 5% of these images are detected with more than one language.

	Count	Percentage
without tag	31,028,877	31%
at least 1 tag	68,971,123	69%
language not detected	26,145,788	26.1%
with one language	41,539,461	41.5%
with two languages	1,256,732	1.2%
with three languages	29,142	0.02%

Table 4.2: Numbers of images with tags vs without tags, and of the images with at least one tag, how many languages were detected [20]

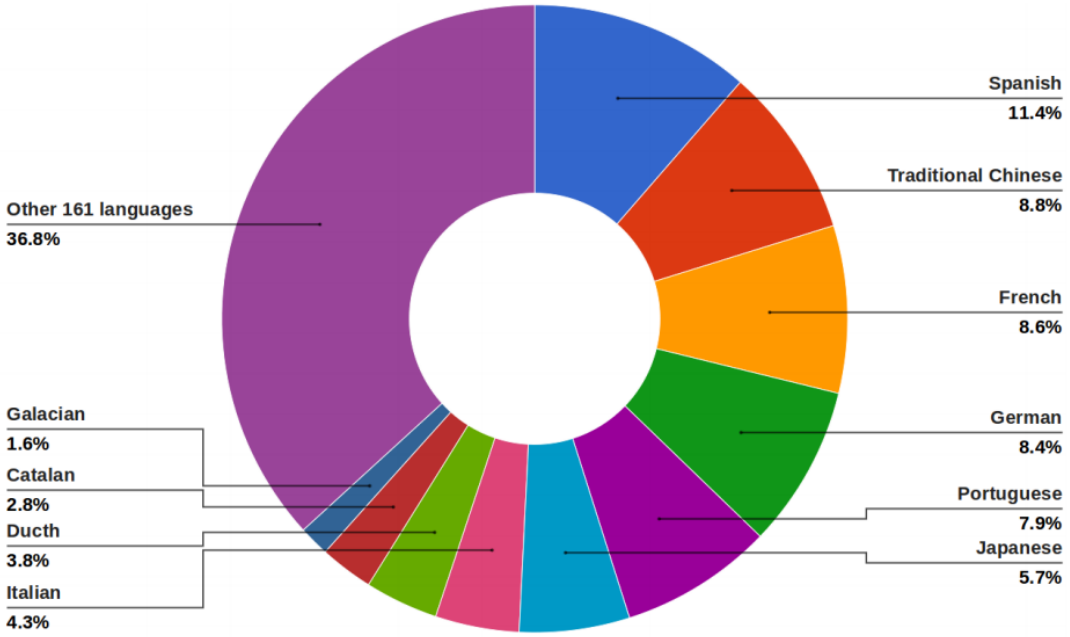

Figure 4.5: Percentage of languages detected in the 24.6% of images where languages other than English were detected [20]

4.2.2 Challenges

Izadinia et al. [19] also investigated the prediction of user tags from images, but using YFCC100M. Their research is more outdated, but as part of their work they offer some insightful analysis of the user tags in the dataset.

They note that multi-word user tags are often split up. For example ”New” and ”York” instead of ”New York”. The problem of this is the independent tags having an entirely different meaning. This makes it challenging to understand the contents of the image by looking at individual user tags.

Different words also often refer to the same coarse classes. Most commonly root words with their stems, for example ”cars” and ”car”. For such examples we could just reduce the word to its stem. But there are also examples where determining they correspond to the same concept is not so easy, such as ”cat” and ”kitten”.

The multi-lingual nature of the corpus identified by Koochali et al. exacerbates this issue. We investigated this, translating ”cat”, ”cats”, ”kitten”, and ”kittens” into each of the 106 languages supported by Google Translate, and found that 132 of the unique translations are also user tags in YFCC100M. The distribution of these user tags frequencies in the dataset is Zipfian distributed, as shown in 4.6. We would like our model to learn they all correspond to the same concept. But having vastly different frequencies makes learning this challenging. This is because there will be fewer examples to learn from at the tail of distribution.

Izadinia et al. also identified that some user tags can have multiple meanings. For example ”rock” can refer to the music genre, and also the physical object. The presence of multiple languages in the corpus also exacerbates this problem. This is because

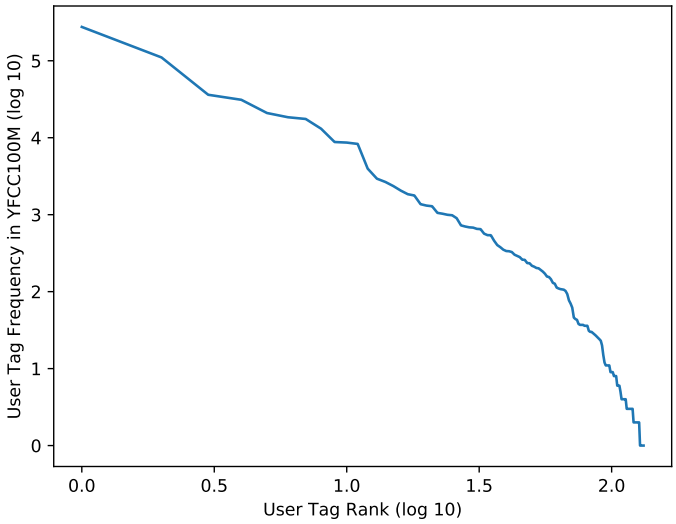

Figure 4.6: Log frequencies of the unique translations of ”cat”, ”cats”, ”kitten”, and ”kittens” that are user tags in YFCC100M

many languages share the same words, but they have entirely different meanings. For example the word ”araba” occurs in the corpus. In Turkish this word means ”car”, and in Basque it means ”Arabic”. This ambiguity is problematic as it's more complex to learn the meaning of user tags if the meaning isn't distinct. The learnt meaning will be biased towards the most common meaning, which in turn is likely to lead to bias against less common languages in the corpus.

They also perform an analysis of the co-occurrence of ground truth labels and corresponding user tags. They do this using the NUS-WIDE [4] dataset, which is a subset of YFCC100M containing 250,000 images and their usertags. High-school students annotated the images with ground-truth labels covering 81 concepts. They performed the analysis by assuming the user tags that were identical to labels corresponded to the same concept. For example the label ”cat” and the user tag ”cat” both correspond to the concept ”cat”.

They find that where the user tag is present, on average the corresponding class is identified in the image only 62% of the time, and when the class is present, on average the corresponding user tag is only present 38% of the time. This reinforces the claim by Mahajan et al. that user tags are sometimes irrelevant to the contents of the image, and relevant ones are often not included. They also find user tags corresponding to entry-level categories[1] are the least often omitted, and users often omit tags for classes that are not the focus of the scene (e.g. buildings).

They also investigated whether the position of tags in the list supplied by the user had an impact on whether the corresponding class was present in the image. The results of this investigation is summarised in figure 4.7. They found that tags supplied earlier

[1]Entry-level categories are the coarse categories that people tend to use to refer to objects. For example you would usually refer to a Labrador Retriever as a dog [27]

in the list give a stronger indication that the corresponding class is present than those later in the list.

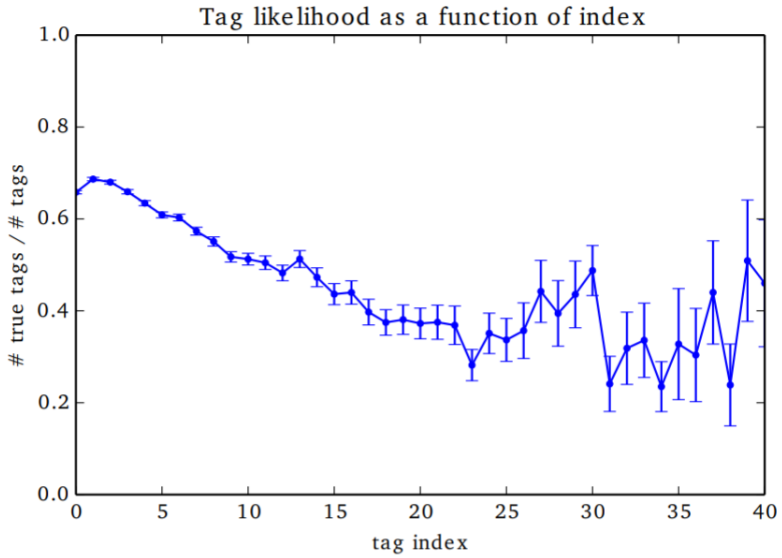

Figure 4.7: Likelihood that user tags correspond class is present given its position in the users list of tags [19]

4.2.3 Summary

In summary Koochali et al. and Izadinia et al. demonstrate the challenges of understanding language in YFCC100M. The corpus is dominated by English, but consists of over a hundred languages. In each language there are often many ways to describe a single concept, with the large number of languages in the corpus we can expect there to be hundreds of ways, and the frequencies of these ways is Zipfian distributed. The multi-lingual text also introduces the challenge that languages can share the same word, but it can have entirely different meanings in each language. People tend to use user tags corresponding to coarse categories, and often do not include user tags corresponding to objects that are not the focus of the image. User tags tend to be more reliable in describing the content of the image the earlier they occur in the user's list of tags. When user tags are present their corresponding classes are not always present, and when classes are present, more often than not their corresponding user tags are not present.

Chapter 5

Proposed Architectures

In this chapter we propose architectures for image classification and object detection that use user tags as features. We conclude by discussing the challenges we encountered that prevented us from implementing either of these architectures, and our decision to research classifying images using only their user tags.

5.1 Purpose of Contextualised Architectures

We propose modifying the teacher in the architecture proposed by Yalniz et al. (as shown in figure 4.2) to include user tags as features. This would allow the teacher to make predictions about the contents of images using both the images and their user tags. We hypothesise that this joint knowledge would allow the teacher to make more accurate predictions than with the images alone.

5.2 Contextualised Image Classifier Architecture

For the image classification task, we propose the architecture in figure 5.1, which we refer to as a contextualised image classifier.

5.2.1 Serial vs Parallel Processing of Images and User Tags

We propose extracting features from the user tags and images in parallel.

We did consider processing them in series. The most logical way to do this would be to treat the user tags as privileged information as proposed by Vapnik et al. [38] Privileged information is features that are only available at training time. The authors proposed how to learn from data along with its privileged information in series using an SVM, but this is not that relevant for image classification as CNNs are the current state-of-the-art. Bisla et al. [3] proposed using privileged information with CNNs, introducing the privileged information as a mask. This approach focuses the attention of the network on parts of the image highlighted by the mask. The question that follows from this research is whether we can transform user tags into a mask. We question

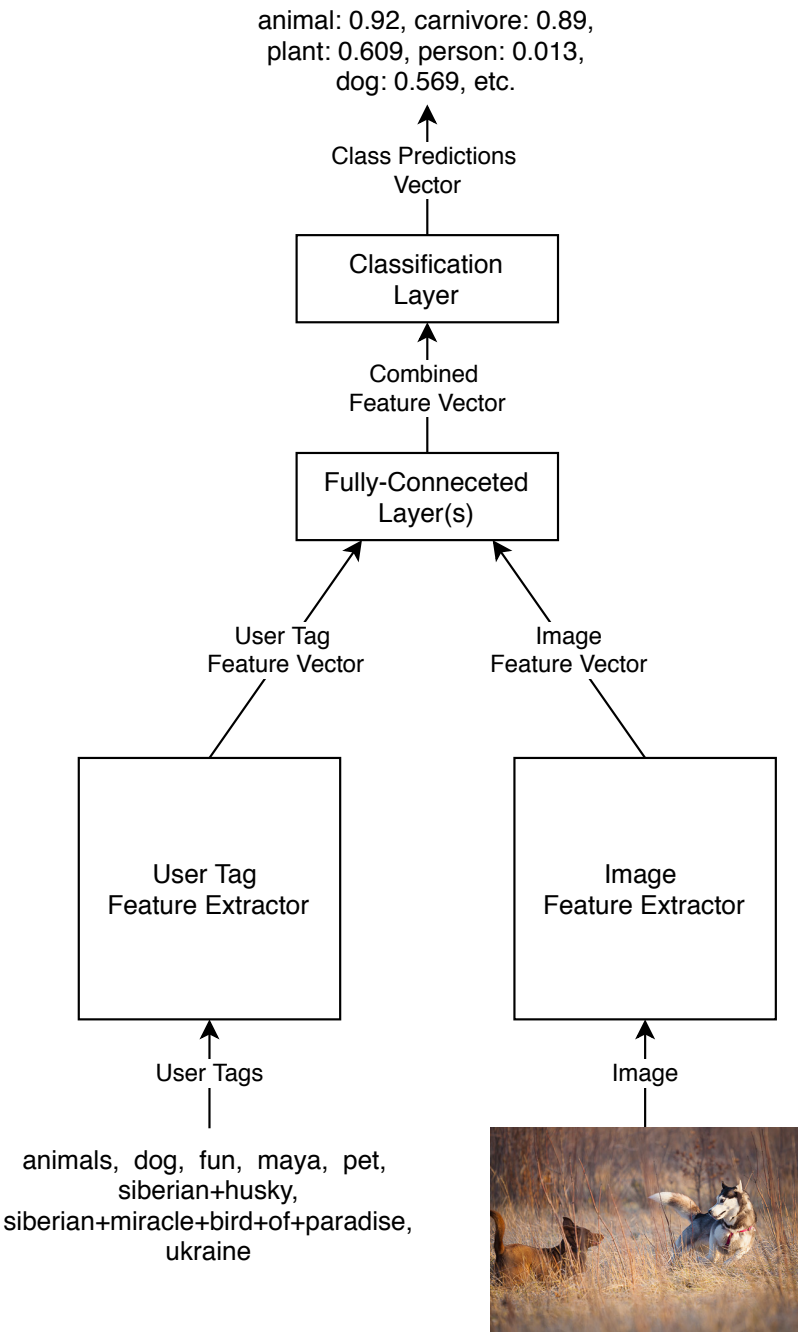

Figure 5.1: Contextualised image classifier architecture

whether this approach is appropriate in our domain given Mahajan et al. found that user tags did not improve localization (as discussed in section 4.1.1), hence user tags do not seem appropriate to use as a mask. More fundamentally transforming user tags into a meaningful mask seems challenging, if not impossible. Considering this, using user tags as a mask for images does not appear to be a good approach. We could not think of a good way to incorporate the privileged information into a CNN other than a mask, and consequently we decided that extracting features from the user tags and images separately was the best approach.

5.2.2 Model Pipeline

We will discuss both feature extractors in detail in section 5.3 and 5.4, but for now we will discuss them at a high level. The user tag feature extractor takes as input the user tags, and produces an output u, which is a vector of length l describing the extracted features from the user tags. Similarly the image tag feature extractor takes the image as input, and produces an output i, which is a vector of length m describing the extracted features from the image.

The two feature vectors u and i are concatenated as a vector f such that:

$$f = (u_0, u_1, ..., u_{l-1}, i_0, i_1, ..., i_{m-1})$$

The feature vector f is then passed through n fully connected layers, each with q units, and an activation function a between each layer. In this work we use ReLU as our activation function for simplicity. Let W_j denote the weight matrix of the j^{th} fully connected layer, with the weights learnt through back-propagation. Then the output o (the combined feature vector) of the fully connected layers is:

$$o = a(W_{n-1} \times a(W_{n-2} \times (a(... \times a(W_0 \times f)))))$$

l, m, n, and q are all hyperparameters that we will discuss more in chapter 8.

The resulting vector o is a high-level abstraction of the joint knowledge from the user tags and the image.

o is input into the classification layer, which is another fully connected layer with c units (where c is the number of classes), with a sigmoid activation function to bound the output of each unit between 0 and 1. The output of the classification layer is the confidence of the network for the presence of each of the classes in the image. We use sigmoid rather than softmax as multiple classes can be present in each image.

5.3 User Tag Feature Extractor

The user tag feature extractor takes as input the user tags, and outputs a user tag feature vector. The architecture we propose for the user tag feature extractor is shown in figure 5.2.

5.3.1 Model Pipeline

The first layer in the architecture is an embedding layer. Embedding layers assign each word in a vocabulary to a unique feature vector [32]. In our context a word is a user tag. All feature vectors are of length l, and the values of the feature vectors are learnt when training through back-propagation. Consequently the vocabulary must be specified at training time. Words outside of the vocabulary are assigned to a single feature vector. For example if we have the vocabulary:

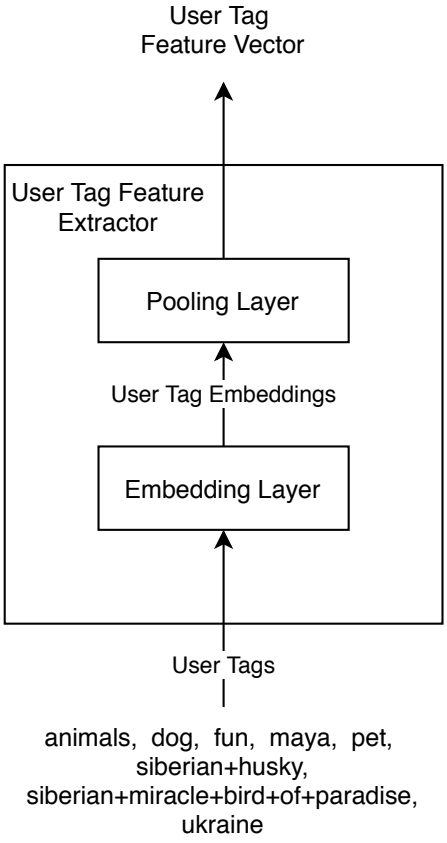

Figure 5.2: User tag feature extractor

$$V = \{\text{"}animals\text{"},\text{"}dog\text{"},\text{"}fun\text{"},\text{"}pet,\text{"}siberian+husky\text{"},\text{"}ukraine\text{"}\}$$

If e_{j+1} denotes the feature vector for the j^{th} word in the vocabulary, with e_0 being reserved as the feature vector for words outside the vocabulary, then if we apply the embedding layer E to the user tags in figure 5.1, the output is:

$$E([\text{"}animals\text{"},\text{"}dog\text{"},\text{"}fun\text{"},\text{"}maya\text{"},\text{"}pet\text{"},\text{"}siberian+husky\text{"},$$
$$\text{"}siberian+miracle+bird+of+paradise\text{"},\text{"}ukraine\text{"}]) =$$
$$[e_1,e_2,e_3,e_0,e_4,e_5,e_0,e_6]$$

The produced output is the user tag embeddings. The number of vectors output by the embedding layer varies, as there is one for each user tag, and the number of user tags provided for each photo varies.

Another option would have to been to represent each of the words using one-hot encoded vectors. But the word embeddings representation for user tags is desirable over this as the embeddings are not sparse. Sparsity makes it challenging to learn the meaning of the components of the vector, as there are many more examples where each component's value is 0 than 1. Embeddings also allows learning similar feature vectors

for words with the same meaning, whereas with one-hot-encodings the representations are vastly different, which would make learning they have similar meanings more challenging. This is important in our case given that there can be hundreds of words with the same meaning as we discussed in section 4.2.2.

The final layer in our feature extractor is a pooling layer. Pooling layers are used to combine multiple vectors into a single vector. This operation is necessary as our embedding layer produces a vector for each input user tag, but our feature extractor should only output one. One of the benefits of pooling layers is they can combine any number of vectors together, which handles the variation in the number of vectors output by the embedding layer.

The type of pooling layer we use is a hyperparameter in our experiments, and we will discuss this more in section 8.3.

The main disadvantage of pooling layers is that the order of the vectors is lost, and thus the order of the words in the text is also lost. In structured language this is problematic, but most user tags can be understood independent of the surrounding user tags. Consequently this does not seem a large disadvantage for our task.

5.3.2 Simple vs Complex Models

The user tag feature architecture we propose could be viewed as simplistic, given the popularity of deep learning for text classification. But Shen et al. [31] showed that simple models like ours can achieve comparable results to deep learning architectures for topic classification, which is a task very similar to ours. The corpora they used to conduct this research was also structured language, hence this architecture might work even better for us given the language is mostly unstructured.

We question how beneficial architectures such as CNNs and RNNs would be for our task given their main benefit is the ability to model the dependencies between words, but the majority of user tags can be understood independent of the other user tags. There are cases where it might be beneficial, for example for understanding multi-word user tags which are often split up as highlighted by Izadinia et al. (discussed in section 4.2.2). Hence we leave exploring whether more complex architectures are beneficial as future work.

5.4 Image Feature Extractor

The image feature extractor takes as input an image, and outputs an image feature vector. We require our feature vector to summarise the contents of the image, to allow us to classify the image's contents. This is exactly what is produced by image classification networks prior to the classification layer. Hence we propose using an existing image classifier with the classification layer removed as our image feature extractor. Consequently the hyperparameter m we discussed in section 5.2.2 is the number of units in the image classifier in the layer before its classification layer.

For example a ResNet-34 image feature extractor is shown in figure 5.3. The archi-

tecture is identical to the ResNet-34 image classification network proposed by He et al. [16], with the exception that we have removed the classification layer. The output from the pooling layer is a vector, which we can use as our image feature vector.

5.4.1 Fine-Tuning Existing Classifier

We would expect the learnt weights for the image feature extractor to be very similar to an equivalent trained image classification network. Hence we can use the weights from a pre-trained image classification network, and fine-tune them in our contextualised image classifier. A good candidate for this is the ResNet-101 [13] pre-trained on OID provided by Google.

This approach is advantageous as training image classification networks from scratch is expensive, especially with large datasets like ours. For example the pre-trained network provided was trained asynchronously using 50 GPUs with a batch size of 32 for 61,995,903 steps. But in our compute cluster nodes have at maximum 8 GPUs, which can be used at maximum for 3 days at a time, and the cluster is also being used by several hundred other students so entire nodes are rarely idle. Hence training this network from scratch would be too expensive given our resources. Fine-tuning a pre-trained network seems more feasible as we'd expect to train for many fewer steps, which would compensate for the bottleneck in the number of GPUs we can use.

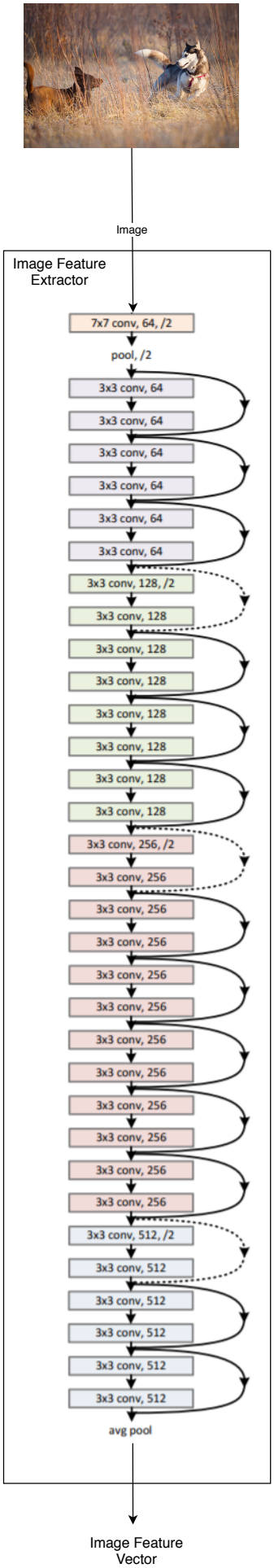

Figure 5.3: ResNet-34 [16] image feature extractor

5.5 Contextualised Object Detector Architecture

We propose that user tags could also be used as features in an object detector architecture. Our proposed architecture is shown in figure 5.4.

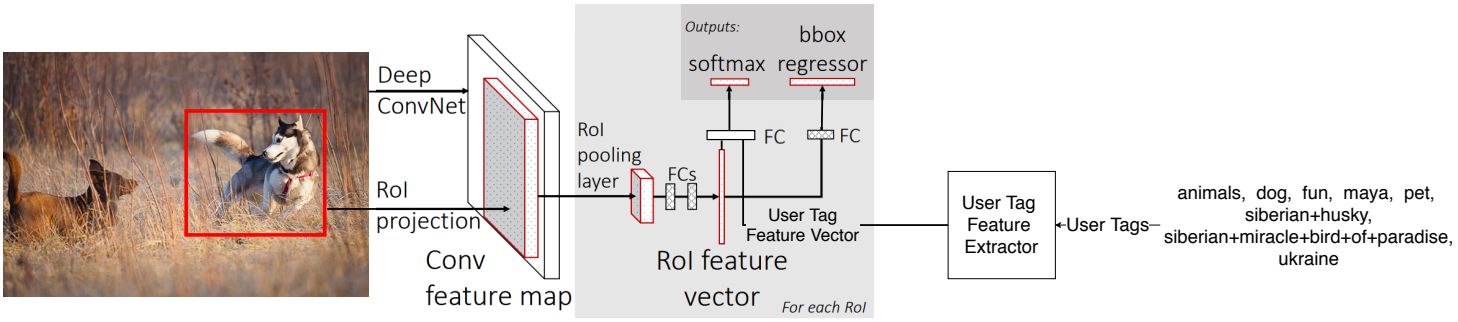

Figure 5.4: Contextualised Fast-RCNN [9] object detector architecture

5.5.1 Base Architecture

The architecture is based on a Fast-RCNN object detector [9]. In this architecture a convolutional neural network generates a feature map for the input image, and another model proposes regions of interest (denoted *RoI*s) in the image which may contain an object. Each *RoI* is projected onto the feature map, and the corresponding features are passed through a pooling layer and a sequence of fully connected layers to produce a feature vector for each. The feature vector is used to predict the object in each *RoI* or if no object is present at all, and the position of the bounding box around the object. To do this it is passed through two parallel fully connected layers, with one to predict which object is present, and another for localizing it. The output of the fully connected layer to predict which object is present is passed through a softmax layer to ensure the probabilities of each class being present sum to 1.

Fast-RCNN is an obsolete architecture, with it superseded by Faster-RCNN [28]. Faster-RCNN improved Fast-RCNN by merging the network computing the feature map and the network predicting *RoI*s into a single architecture, which sped up training and inference. Hence it would be more appropriate to experiment applying our modification to Faster-RCNN than Fast-RCNN. Given that Faster-RCNN only differs from Fast-RCNN in the way regions are proposed, our architecture could also be applied to Faster-RCNN with the same modification we propose as it does not intefere with this.

5.5.2 Modifications to Architecture

Given that Mahajan et al. (as discussed in section 4.1.1) found that user tags were helpful for classification but not localization in object detection networks, we propose modifying the way Fast-RCNN classifies the object in each *RoI*. In a similar way to the image classification network, we propose concatenating the feature vector for each region of interest with the user tag feature vector just before it is passed through the

final fully connected layer to predict the classes. The purpose of the user tag feature extractor is the same in this architecture as the image classifier, so we propose the same architecture (which was discussed in section 5.3).

We believe that the knowledge contained in the user tag feature vector of what could be in the image will help the object detector better predict which object could be in each *RoI*. For example the network might be unsure of whether the region proposed in figure 5.4 contains a dog or a wolf. But the user tags ”pet” suggest a domestic animal is in the image, and ”dog” is a strong indication that there is a dog in the image. These user tags in combination should help the network be more confident predicting the region is much more likely to contain a dog than a wolf.

5.5.3 Fine-Tuning Existing Architecture

Training object detection networks is expensive too. We believe training this network could be sped up by using the weights from a pre-trained network without our modification, and then finetuning them. The only part of the network where we wouldn't be able to re-use weights is the fully connected layer used for classification, because our modification has changed the vector input to this layer. Google provide a Faster-RCNN Inception ResNet V2 [35] pre-trained on OID which we could use for this purpose.

5.6 Challenges in implementation

In this chapter we have proposed two potential architectures, and whilst we think both are worth researching, we early on realised there was not enough time for this.

The engineering work to implement the modification proposed to Faster-RCNN in section 5.5.2 turned out to be challenging. The pre-trained network we proposed modifying is implemented in the TensorFlow Object Detection API [18]. This API was designed specifically for object detection, so the pipeline for training the network and inference is not designed for input other than images. Modifying it to take user tags as input would involve significant changes to the API. This is a large library, and given that we do not have much experience with low-level TensorFlow, this did not seem like it would be an easy task. We were concerned that we would spend a lot of time on engineering work, and potentially would not have it working by the time the project was due. Consequently we decided against this avenue of research, as it would shift the focus of the project from research to engineering, and potentially we could end up with no results to show.

We also ran into issues implementing the image classification network proposed in section 5.2. The pre-trained image classification network we planned to use is implemented in the depreciated TF-Slim library [30]. In order to use it we would have to use an outdated version of TensorFlow. By the time we realised this we had written a significant amount of code for the user tag pipeline, which was incompatible with earlier versions of TensorFlow. Additionally TF-Slim does not have good multi-GPU support, which means training across multiple GPU's would be challenging. The pre-trained model is trained on OID V2, an earlier iteration of the dataset, which had 70%

fewer human-verified labels. Consequently we would need to effectively train it from scratch to make it a fair starting point for our architecture, which we didn't have time to do given our resources.

5.7 Focus of this Project

Given the challenges we described in section 5.6, we decided to focus the project on the user tag feature extractor instead. To do this we propose a similar architecture to the one we proposed for the image classifier in figure 5.1, but without the image feature extractor. This architecture is shown in figure 5.5.

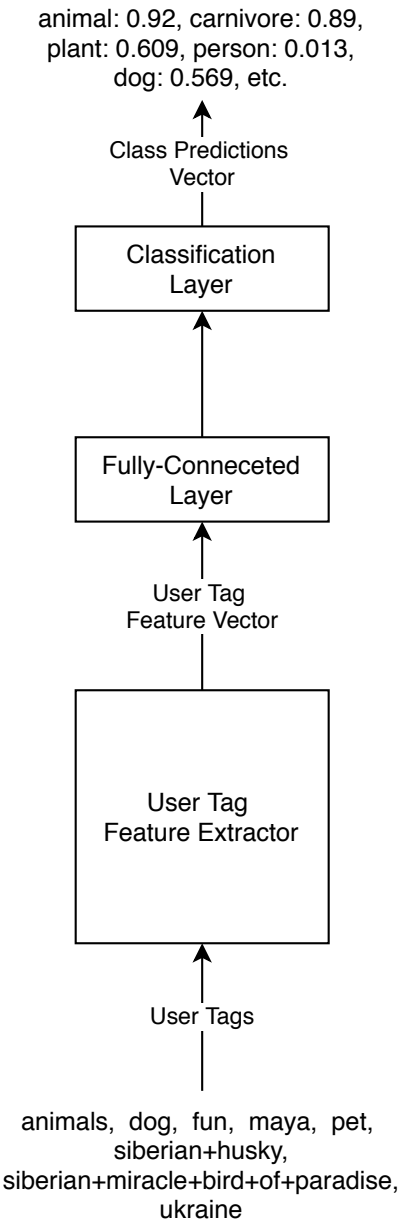

Figure 5.5: User tag image classifier architecture

5.7.1 Motivation

We do not see any practical application for this standalone architecture, but it can still help us understand whether user tags have any use as a feature for image classification. We can use the machine-generated labels provided with the dataset as a baseline (which we will discuss more in section 7.1) to determine if the user tags help us predict any classes better than just using the feature extracted from the image. We can also use a naive baseline (which we will discuss more in section 7.2) to determine whether user tags are useful as features or no better than randomly guessing.

This architecture is also a lot faster to train than the image classifier we proposed, as the user tags are not big so we can cache the entire training set in memory. This allows us to explore whether semi-supervised pre-training on YFCC100M is beneficial to performance as we discuss in chapter 9. Also the pre-trained user tag feature extractor may be useful for future work, as it might be beneficial to use the weights from a pre-trained user tag extractor in the proposed networks, given word embeddings are notoriously expensive to train.

Chapter 6

Metrics

In this chapter we discuss a metric for object detection, and how this was modified for OID. We conclude by discussing how we modified the OID metric to allow evaluation of our model.

6.1 Pascal VOC 2010

Pascal VOC 2010 [7] is a metric used to evaluate classifiers trained for object detection. It measures the mean average precision (abbreviated to *mAP*) for predicted boxes where the intersection over union of the ground truth bounding box and predicted bounding box exceeds 0.5 (abbreviated to $IoU > 0.5$).

A predicted box B_p is considered to have $IoU > 0.5$ with a ground truth box B_{gt} where:

$$\frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} > 0.5$$

Where $IoU > 0.5$ we count it either as a TP (true positive) if the predicted box's class matches the class of the ground truth box, and a FP (false positive) otherwise. If there are multiple predicted boxes with $IoU > 0.5$ with a single ground truth box, all correct predictions after the first are counted as FPs.

The average precision (denoted *AP*) for a class is the area under its precision-recall curve. The curve can be calculated from predictions on a test set by plotting the precision and recall at each threshold for the probabilities of the class being in the predicted bounding boxes. To smooth the curve, if for any recall r_1 with a corresponding precision p_1 there exists a recall r_2 with a corresponding precision p_2 such that $r_1 > r_2$ and $p_1 < p_2$, then the corresponding precision for r_1 is taken as p_2. The area under the curve is then calculated using integration.

The *mAP* is the mean of the *AP*'s for each of the classes.

6.2 Open Images Evaluation Metric

The OID evaluation metric [12] for object detection is based on the object detection metric proposed in Pascal VOC 2010.

mAP is an ideal evaluation metric for OID as it is not sensitive to imbalanced classes. This is important as the classes of OID are very imbalanced as discussed in section 3.1.1 and shown in figure 3.3.

The OID metric has some modifications to support OID.

6.2.1 Non-Exhaustive Labelling

Pascal VOC 2010 assumes exhaustive labelling, but OID labelling is not exhaustive, as discussed in section 3.1.1. This assumption is relaxed with some modifications. Where a class is annotated as present at image-level, all objects in the image of the class are annotated with bounding boxes. Predicted boxes are ignored when calculating the metric if the predicted class is not present at the image level, this is because the presence of the object in the image is unknown. For example in figure 3.1 there are people faintly visible in the car behind, but if a bounding box was predicted for ”person” around these people, it would be ignored, as the person label is absent at the image level. Likewise if a bounding box for an airplane was incorrectly predicted in the sky, it would also be ignored, as the label airplane is also absent at image level.

6.2.2 Coarse Predictions of Fine-Grain Objects

The object classes are arranged in a dendrogram, shown in figure 3.4. It is desirable that even if the network does not predict the fine-grain object class for a bounding box, it should receive a smaller reward for correctly predicting the coarse object class. To achieve this, for each predicted object class a TP or FP is counted for it and each of the classes' ascendants in the dendrogram. For example, if a ground truth box contained a trombone, but a predicted box with $IoU > 0.5$ predicted it as containing a trumpet, a FP would be counted for trombone, and a TP would be counted for musical instrument. This modification is why the new dendrogram discussed in section 3.1.1 is required, as in the original dendrogram released before the metric, there were examples that would distort it, such as building being an ascendant of computer keyboard, and also very broad classes such as clothing which encompassed a large range of objects.

6.2.3 Groups of Objects

There are ground truth boxes in OID that bound groups of objects, as discussed in section 3.1.1. To accommodate this, if a ground truth box bounding a group of objects has $IoU > 0.5$ with multiple predicted boxes for the ground truth class, only one TP is counted, with the rest ignored and no penalty applied. If none of them predict the correct class, this is counted as a single FN (false negative).

6.3 Our Metric

We propose modifying the OID object detection metric discussed in section 6.2 to evaluate image classifications networks, including the one we proposed in section 5.7.

Given that image classification networks do not localize the object, IoU is not applicable as there are no bounding boxes. Instead we treat the ground truth image labels as ground truth bounding boxes covering the entire image, and predicted labels as predicted boxes covering the entire image. This ensures that $IoU > 0.5$ is always true, relaxing the constraint.

We decided using *mAP* was a better approach than using accuracy as the classes are imbalanced which would distort it. Also the modifications to it discussed in section 6.2 make the metric better suited to the dataset than other common metrics.

Chapter 7

Baselines

In this chapter we introduce the baselines which we use to evaluate our model.

7.1 Machine-Generated Labels

For this baseline we use the machine-generated labels supplied with OID (discussed in section 3.1.1). These labels are accompanied by the confidence of the model that predicted them, which allows evaluating its performance using *mAP*. This is a good baseline as it give us an idea of how an image classification network would perform on the dataset.

There are some downsides to this baseline which we should consider.

The first is that the network that generated them was not trained on OID, instead it was trained on Google's internal JFT dataset [33]. It is unclear whether this baseline would perform worse or better than one trained on OID. JFT is 30 times bigger than OID, and has 10 times more labels, hence their model may perform better as it had more data to learn from. But JFT only contains noisy labels, which were obtained automatically and are not exhaustive. Hence their model may perform worse depending on how these labels were collected and how beneficial the human-verified labels in OID are. Consequently it is hard to say how this baseline would perform relative to one trained on OID. But nonetheless it should serve as a good general baseline for how an image classifier would perform.

Another downside is that only machine-generated labels with a confidence greater than 0.5 are included in the dataset. This means in our baseline any confidence c in the interval $0 < c < 0.5$ is floored to 0. This is a problem as this will distort the precision-recall curve, and in turn the *mAP* too. As a solution to this when comparing our model to this baseline we suggest also flooring our models predictions to 0 when they occur in this interval. This will allow a fairer comparison of the two networks.

7.2 Most Likely Prediction for Each Class

For this baseline we counted the number of positive and negative human-verified image-level labels for each class in the training set. We predict a class is present in all the images if we counted more positive labels than negative, and vice versa.

This baseline is useful as if the model proposed in section 5.7 performs very similarly to or worse than it, then this suggests that the model isn't learning anything from the user tags. This will help us understand if user tags are useful features.

Chapter 8

Choosing Hyperparameters

In this section we discuss the hyperparameters we explored for the architecture proposed in section 5.7, and also discuss the preprocessing we applied and why it is necessary. We conclude by selecting the best performing model.

8.1 Configuration for Training

We trained our model on the dataset we created, discussed in section 3.3.

To help the model generalize better we used L2 regularization and added dropout layers before each fully-connected layer.

For our optimizer we used Adam, with the default hyperparameters.

All our training was conducted on the MLP cluster. In this chapter we use the Teach Standard partition. This partition consists of 18 nodes, each with 12 CPUs, 8 GTX 1060s, and 96GB of memory. In each job we used 1 GTX 1060, 1 CPU, and 12GB of memory.

8.1.1 Selecting the Best Model

During training we evaluated our model using loss. This is because our primary evaluation metric discussed in section 6.3 is expensive to compute. We used binary-cross entropy loss as this works well for multi-class classification problems like ours.

For our loss metric we needed a ground truth for each label. But the human-verified labels are not exhaustive. To overcome this we used the machine-generated labels in the training and validation set where human-verified ones are absent. This does limit the performance of the model as the machine-generated labels are not as high quality as the human-verified ones. But otherwise we would have had to assume that labels not provided are not present in the image, which would limit performance more. As such we decided to use the machine-generated labels.

When computing our primary metric, we only used the human-verified labels, because the metric is designed to work without exhaustive labels.

We did not restrict our model to training for a fixed number of epochs, instead we trained for as many epochs as we could in 7 hours. We chose 7 hours because the maximum time limit for a job on the partition we were using was 8 hours. We left an hour spare to ensure there was enough time left over to evaluate the best found model and transfer data to/from the head node. After each epoch we evaluated the model on the validation set, and saved the model that achieved the lowest loss on the subset. This ensured that we did not penalize configurations that took longer to train.

8.1.2 Searching the Hyperparameter Space

We explored the hyperparameter space using Random Search. We decided this was the most efficient approach given that time was limited in this project, and spending a lot of time tuning hyperparameters by hand would be a poor use of our time. Also the partition we used had hundreds of GPUs, hence running hundreds of jobs each using a single GPU was an efficient use of this resource. As the partition was in use by others the jobs did have to wait in a queue. In total we explored 600 random configurations in our hyperparameter space, which took 2 weeks to complete.

We used the same random seed for randomness in the first 600 experiments. Consequently these experiments were biased towards configurations that performed well with this seed. To prevent this distorting the results, we took the top 3 best configurations for each pair of hyperparameters for user tag limit and threshold (we will describe these in section 8.2), and repeated these experiments 2 more times each with different seeds. We focused on finding the best configuration for each pair of these hyperparameters as we believed our model would be most sensitive to these, and we thought it was important to understand the effect of these parameters.

8.2 User Tags

8.2.1 Tag Limit

We had to define a limit n for the number of user tags we considered for each image. This was necessary as TensorFlow requires the maximum length of the samples in each batch to be fixed. We propose only keeping the first n user tags for each image. We hypothesised this may be beneficial as Izadinia et al. [19] found that the later a user tag occurred in the list of user tags, the less likely that the corresponding class was present (as discussed in section 4.2.2). Thus removing user tags that occurred later in the list should reduce noise.

For example, if we set a tag limit of 5, then for the sample in figure 3.7, only the user tags ”animals”, ”dog”, ”fun”, ”maya”, and ”pet” would be kept, with 3 user tags after the first 5 discarded.

Where an image has fewer tags than the tag limit, the list of tags are padded to reach the tag limit, and the embedding layer ignores the padding.

To better understand what range of values to investigate for this parameter, we plotted how increasing the limit affected the proportion of images with all tags kept, which

is shown in figure 8.1. We observe that the proportion is Zipfian distributed, with the majority of images having less than 10 user tags, and on average 4. But there are a small proportion of images that have over 100. Considering this we decided to weight our search to investigating smaller tag limits, and decided to explore the values 2, 5, 10, 20, 40, 80, and 160.

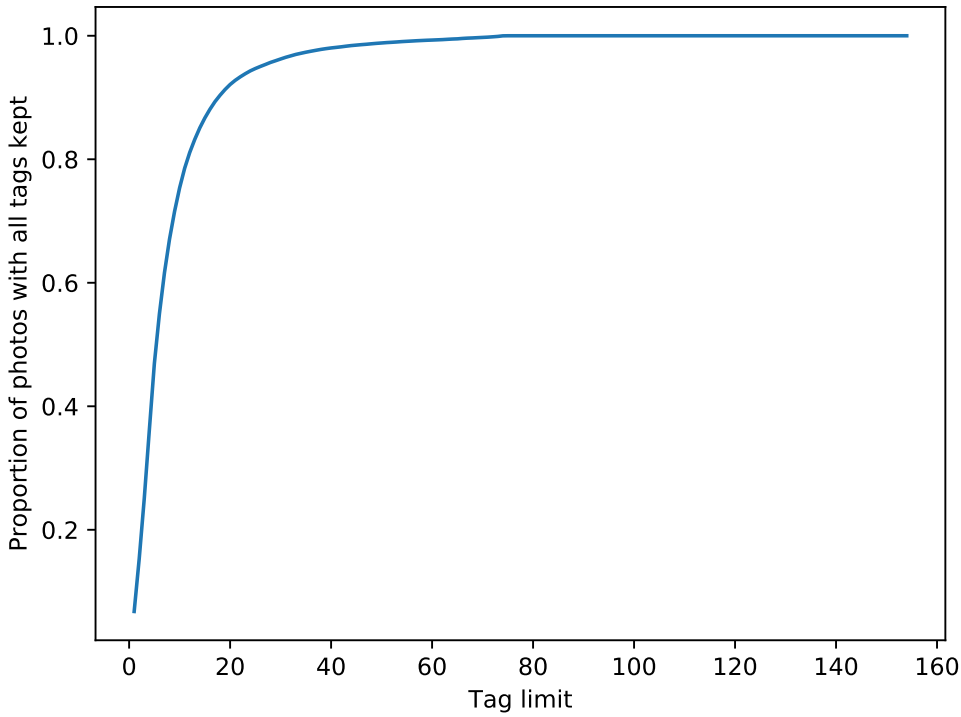

Figure 8.1: The proportion of images that keep all user tags as the tag limit is increased

8.2.2 Tag Threshold

We also needed to set a threshold for the required number of examples for a user tag to be considered part of the vocabulary (this is the same vocabulary we discussed in section 5.3.1). This is necessary as a lot of user tags only occur once in the training set, and even those that only occur a few times are challenging to learn a good embedding for. But as we increase this threshold, the vocabulary becomes smaller, which means more images have no user tags. Images with no user tags are impossible for our architecture to classify.

The user tag limit influences the number of examples for each user tag, with reducing the limit resulting in fewer examples of user tags. For this reason we determine whether a user tag exceeds the threshold after the tag limit has been applied.

For example if we have a training set containing only 2 samples, with the following user tags respectively:

$$[\text{"}dog\text{"},\text{"}cat\text{"},\text{"}pet\text{"},\text{"}cute\text{"},\text{"}woof\text{"}]$$

$$[\text{"}pet\text{"},\text{"}dog\text{"}]$$

If we set the tag limit as 2, and the tag threshold as 2, then our vocabulary would be:

$$V = \{\text{"}dog\text{"}\}$$

This is because as the tag limit is 2, we only count the user tags that occur in the first 2 user tags of each sample. This means "pet" and "cat", are each counted once, and "dog" is counted twice. Consequently only "dog" is included in the vocabulary.

To better under the range of values to investigate for the tag threshold, we plotted how increasing the threshold affects the number of user tags kept in the vocabulary, shown in figure 8.2. To understand how this parameter interacts with the tag limit, we plotted the curve for the limit 1 to 10. Tag limit only appeared to impact the offset of the the curves, with the curves having the same gradient up until the tail of the Zipfian distribution. A user tag threshold of 10^2 results in between 10^3 and 10^4 tags in the vocabulary. This suggested that thresholds higher than 10^2 would not work well, as the number of classes would be greater than the size of the vocabulary. We judged that the optimum value for this parameter was in the range 2 to 100, and decided to investigate this range. Given that there is a steep drop of tags in the vocabulary as the threshold is increased, we decided to weight our search towards the lower end of this range. We investigated the values 2, 5, 10, 20, 40, and 80.

8.2.3 Preprocessing

As the architecture only takes user tags an input, we discarded all samples in the training, validation, and test sets which had no user tags. Including them would have hindered our experiments, as these are impossible for our network to classify. If we had been able to integrate the image feature extractor, we would have included these images, as the image classification network (discussed in section 5.2) should be able to make a prediction still in the absence of user tags.

We also considered whether we should discard images where no user tags were in the vocabulary, but decided these should be kept. The problem with keeping them is that these images are impossible to classify with our architecture, as it is effectively the same as having no user tags. But the problem with removing them is that this would interact undesirably with the user tag threshold hyperparameter, because as we increase the threshold, there would be fewer images in the validation and test sets. The images removed would be the ones only with user tags at the tail of the Zipfian distribution of user tags (shown in figure 3.6). We would likely find that increasing the threshold increases performance, as we would be removing the more challenging images to classify. Consequently even though these images are impossible to classify, we decide that they should be kept in the dataset.

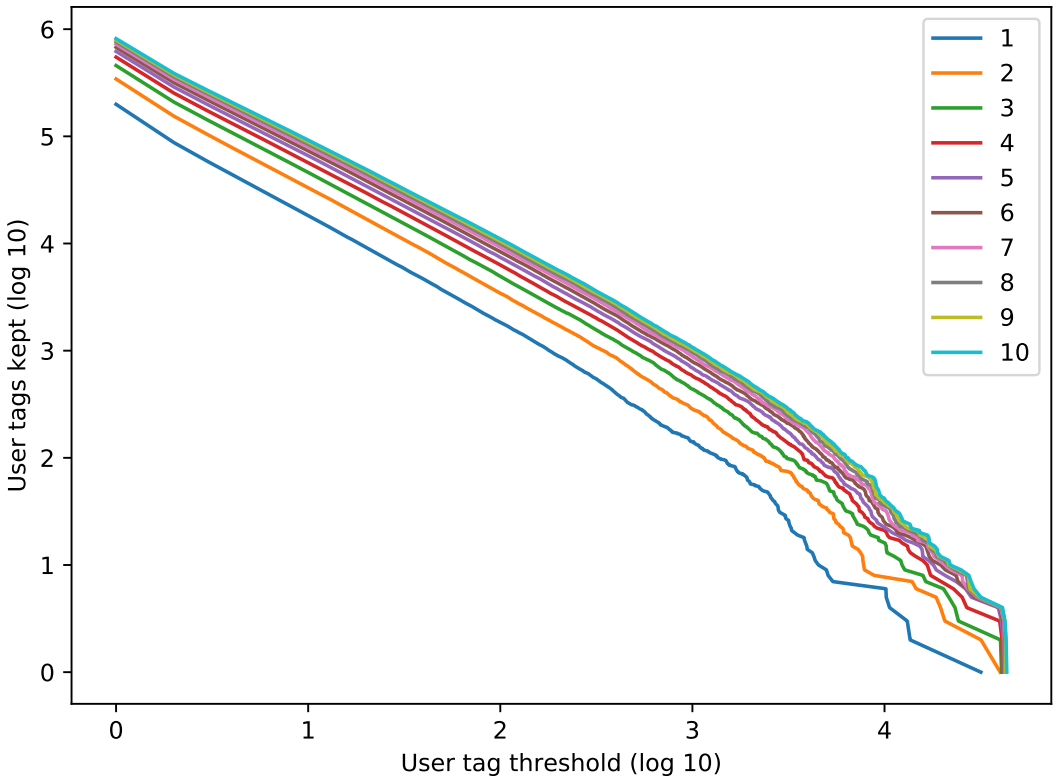

Figure 8.2: The number of user tags in the vocabulary as the tag threshold is increased. Graphs are plotted for the tag limits 1 to 10

We removed user tags that only contain numbers, as is common practice in NLP. Our main reason for this is that they usually aren't descriptive. For example prior to their removal, the distribution of the user tags was dominated by 21st century years, which tell us nothing about the contents of the image, and are likely to confuse the classifier. There are exceptions to this, for example aircraft are often referred to by their model numbers, for example 747. But in the majority of cases these numbers carry no information.

We did consider stemming the user tags, as this would solve the problem we discussed in section 4.2.2 where separate embeddings are learnt for root words and those with suffixes, such as "cat" and "cats". But this is not straight forward due to the corpus being multi-lingual. We discuss why this is a problem and possible solutions in section 11.1.

8.3 Pooling Layer

As we discussed in section 5.3.1, we use a pooling layer after our embedding layer. We experimented with both max pooling and average pooling.

8.3.1 Definitions

Average pooling computes each component of the pooled vector as the average of the component across the input vectors. For example if the input was m vectors with n components, and $v_{i,j}$ denotes j^{th} component of the i^{th} vector, then the output of the pooling layer would be:

$$AveragePooling([v_0, v_1, ..., v_{m-1}]) =$$
$$(\frac{v_{0,0} + v_{1,0} + ... + v_{m-1,0}}{m}, \frac{v_{0,1} + v_{1,1} + ... + v_{m-1,1}}{m}, ..., \frac{v_{0,n-1} + v_{1,n-1} + ... + v_{m-1,n-1}}{m})$$

Max pooling computes each component of the pooled vector as the maximum value for that component across the input vectors. For the same example above, the output vector of max pooling would be:

$$MaxPooling([v_0, v_1, ..., v_{m-1}]) =$$
$$(\max v_{0,0}, v_{1,0}, ..., v_{m-1,0}, \max v_{0,1}, v_{1,1}, ..., v_{m-1,1}, ..., \max v_{0,n-1}, v_{1,n-1}, ..., v_{m-1,n-1})$$

8.3.2 Average vs Max Pooling

We believed that max pooling could perform better as it helps filter out noise. User tags are often irrelevant to the contents of the image, as noted by Mahajan et al. [25] (discussed in section 4.1.1). Also as we reduce the tag threshold, more words are excluded from the vocabulary. As a result the embedding for unknown words (denoted e_0, introduced in section 5.3.1) will appear more frequently, introducing noise. Max pooling reduces noise, as we only take the largest component of the input vectors, meaning only one user tag contributes to each component of the vector. To reduce the impact of irrelevant user tags, the model can learn small weights for their components. This means that if there are a lot of user tags but only a small number are relevant, the irrelevant ones should have less impact on the pooled vector. Shen et al. [31] found that max pooling on word embeddings selects the most salient features of the word embeddings, which is the effect we desire.

But Mahajan et al. [25] were able to achieve good results using user tags as labels despite the noise, hence we are not certain that max pooling is necessary. If this is the case average pooling may work better. With average pooling less information is lost than with max pooling, as all embeddings contribute to the value of each component. If noise is not an issue this should summarise the user tags better than max pooling.

8.4 Other Hyperparameters

We considered how many fully-connected layers to have between our user tag feature encoder and classification layer (for the model in figure 5.5). Shen et al. [31] achieved good results classifying natural language without any hidden layers. User tags also tend to be quite precise descriptions of the image, as such we don't think adding lots of

hidden layers would be beneficial as the language in the corpus isn't complex. Given this we decided to only use 1 hidden layer.

Another hyperparameter we considered is the number of units in the embedding layer. The larger we make this the more capacity our embedding vectors have to capture the meaning of the user tags. Typically even for large corpora you wouldn't have more than 1000 units. Considering we are only classifying 500 classes, we decided to investigate using 64 and 128 units.

To simplify the hyperparameter search, we decided to use the same number of units in the hidden layer as we do in the embedding layer.

It is typical to use a dropout rate less than 0.5, hence we experiment with values in the range 0 to 0.5.

We also intended to tune the learning rate and regularization factor for L2 regularization. But in our experiments we accidentally left these as their default values. We did not have time to re-run the experiments as we realised the mistake a few days before the project deadline. Consequently these are left as their default values of 0.001 and 0.01 respectively.

8.5 Summary of Hyperparameter Search Space

As introduced in sections 8.2, 8.3, and 8.4, the hyperparameters we explore are summarised in table 8.1.

Name	**Explored Values**
Tag Limit	2, 5, 10, 20, 40, 80, 160
Tag Threshold	2, 5, 10, 20, 40, 80
Layer Capacity	64, 128
Pooling Layer	Average, Max
Batch Size	128, 256
Dropout Rate	0, 0.1, 0.2, 0.3, 0.4

Table 8.1: Hyperparameters explored

8.6 Results

8.6.1 Affect of Tag Limit and Tag Threshold

In figures 8.3 and 8.4 we show the performance of the best performing configuration found for each pair of tag threshold and tag limit. The performance across the three random seeds is used to compute mean and standard deviation for each configuration.

Our hypothesis that a low tag threshold results in too much noise is proven correct, with a tag threshold of 2 being consistently worse. We also see poor performance setting the tag threshold to 80. This suggests at the threshold is increased we start to exclude too many words to be able to predict the classes in the image, as we expected.

We also observe that setting the tag limit too low results in significantly worse performance. Tag limits of 2 and 5 consistently under-perform. This contradicts what we would first expect given the observation of Izadinia et al. [19] that tags are less likely to be relevant the further they are into the list, as shown in figure 4.7. But their analysis only matched user tags with labels where the name was the same. Hence the difference might be because our model is able to also learn the user tags that are relevant which do not share the same name as the label.

We do not see a significant fall in performance from setting very high tag limits. The latter might be explained by figure 8.1, with only a small proportion of images having more than 40 tags. As only a small number of samples are affected we would not expect a large impact on performance. Tag limits of 80 to 160 do for the most part perform slightly worse than 20 to 40 on average, but this difference is within the standard deviation. As such it is inconclusive whether too high tag limits lead to worse performance.

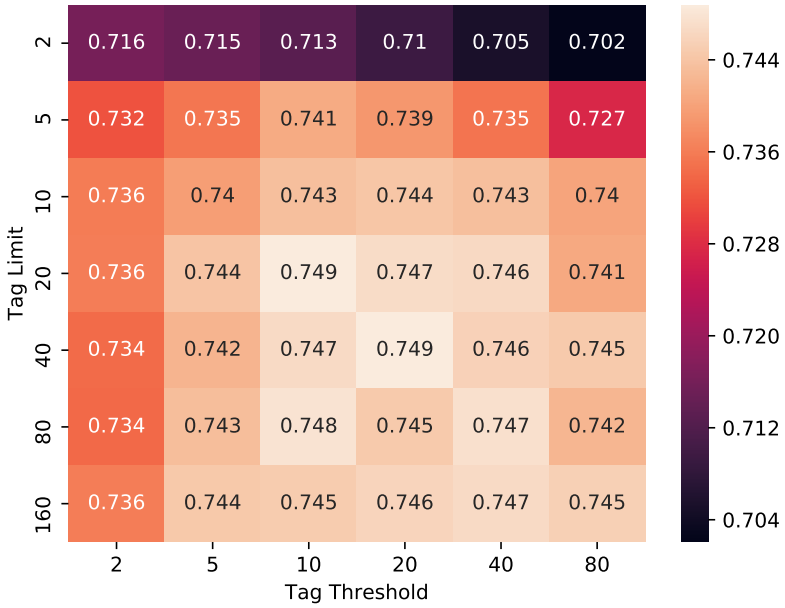

Figure 8.3: Mean of mAP for best found configuration on validation set

8.6.2 Average vs Max Pooling

To understand whether max or average pooling performs best, in figure 8.5 we show the performance for the best configuration we found for each of them. We see that max pooling clearly outperforms average pooling, with a higher average mAP and also a much smaller standard deviation. Given that the information from most words is lost using max pooling, this suggests that only a small number of user tags for the image describe the objects in it.

This might also explain why we do not see a large drop in performance when we increase the tag limit. For tag limits of 80 and 160, only 1 out of 36 of the best

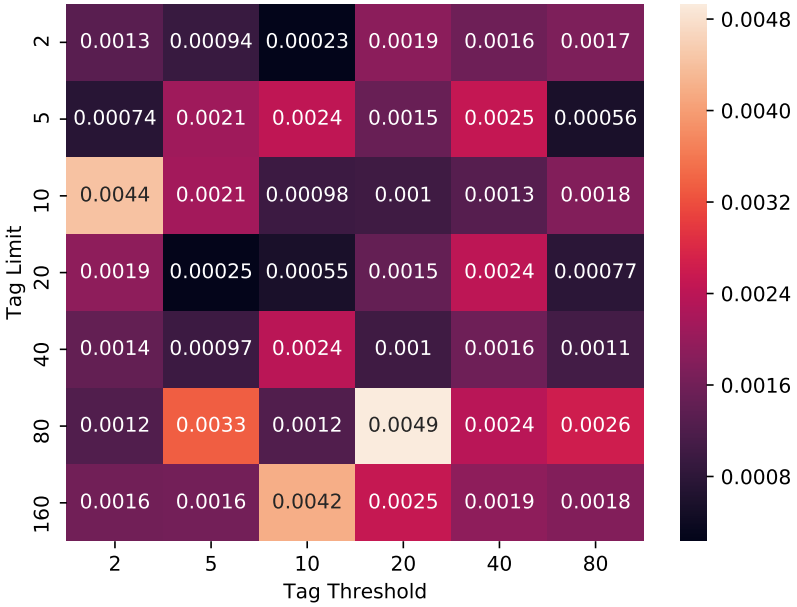

Figure 8.4: Standard deviation of mAP for best found configuration on validation set

performing configurations used average pooling, with the rest using max pooling. It might be that max pooling negates the noise introduced by increasing the tag limit, given that only a few tags contribute to the pooled vector output by the layer.

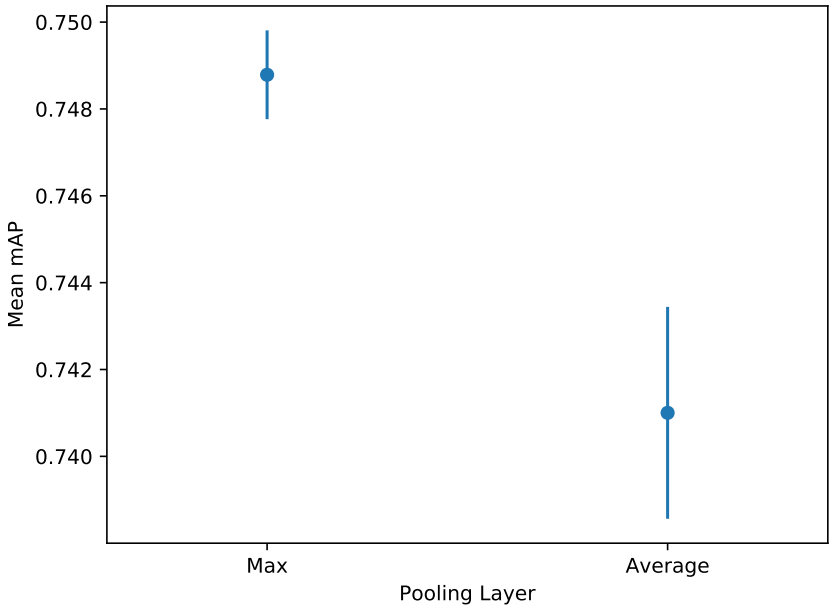

Figure 8.5: Performance of best configuration for average and maximum pooling on validation set. Error bars show standard deviation.

8.6.3 Best Model

Considering figures 8.3 and 8.4, we judge the best configuration to be the one found with a tag threshold of 10, and a tag limit of 20. This is because it achieved the highest mean mAP with one of the lowest standard deviations. The parameters of this configuration are shown in table 8.2.

Name	**Best Value**
Tag Limit	20
Tag Threshold	10
Layer Capacity	128
Pooling Layer	Max
Batch Size	128
Dropout Rate	0.1

Table 8.2: Best hyperparameters found

Chapter 9

Pre-Training on YFCC100M

In this chapter we explore whether pre-training on a larger dataset is beneficial to performance.

9.1 Purpose of Pre-Training

In chapter 8 we discovered that only including user tags in the vocabulary with at least 10 examples in the training set helped improve the performance of our model. There is a trade-off with this parameter that the higher the threshold is set, the smaller the vocabulary becomes. In this chapter we experimented with a technique to increase the number of examples we have for each user tag without increasing the threshold further. We proposed pre-training the model on the remainder of YFCC100M that does not intersect with OID. We hypothesised that having more examples whilst maintaining the same size vocabulary would allow the model to generalize better and exceed the performance of the best model we found in chapter 8. We expected this would help improve performance for images with non-English languages the most, given they are under-represented in the corpus.

9.2 Dataset for Pre-Training

As we discussed in section 3.3.3, only 2 million images in OID intersect with YFCC100M. This meant there were a further 97 million images in YFCC100M which we were not using. Of these 66 million were suitable for including in our pre-training dataset as they met our requirement of having at least one user tag (as discussed in section 8.2.3).

As discussed in section 3.2.2, machine-generated image-level labels are provided for all images in YFCC100M, which we proposed using. These are likely lower quality than the machine-generated labels for OID given they were produced by an older architecture. This meant pre-training on them could lead to worse performance. But we did not have the resources or time to classify these using a more modern architecture, hence we used using them despite this.

This allowed us to create a pre-training set that is 3200% larger than our training set. This much larger training set gave us many more examples for each user tag in our vocabulary, without reducing the size of it.

There are 1,570 labels in YFCC100M, and the most common labels are shown in figure 3.8. We observed that a lot of labels are not objects, for example ”black” and ”abstract”. We were concerned that our model may learn concepts that were irrelevant to our task at the cost of performance. Given this we decided to discard the labels that did not correspond to objects. In the remaining labels there were still a lot more labels than we were using for OID, with the labels tending to be more fine-grain than ours. For example in YFCC100M they have the label ”labrador”, but in OID they simply have ”dog”. We were concerned this could reduce performance, through the model having to learn to differentiate objects that in our label set corresponded to the same label. As a result we decided to hand-match the objects in YFCC100M to the corresponding labels in OID, and train using the OID labels. Any labels that were more fine-grain than in OID were assigned to the most fine-grain label in OID that encompassed them, and if no such label existed they were discarded.

We decided to include our training set in the pre-training dataset too. This ensured that we have an example of every label in OID for pre-training, as YFCC100M does not cover every label in OID.

9.3 Configuration for Training

For this experiment we took the best configuration found in chapter 8 which is summarised in table 8.2.

We trained our model in a similar way to in chapter 8 which is described in section 8.1.

We first trained on the pre-training dataset, and we increased the training time to 71 hours given that the dataset is much larger and hence training was slower. We chose the best model using binary-cross entropy loss as we did in section 8.1.1. We repeated this three times using different seeds.

We took each of the pre-trained models and froze the weights of the embedding layer. We fine-tuned the remaining weights by training them on the training set for 7 hours, and choose the best model again using binary-cross entropy loss. For fine-tuning we decreased the learning rate by a factor of 10, as is standard practice to prevent weights moving too far from the learnt values in pre-training.

We trained again using the MLP cluster. For pre-training we used the General Usage partition, which allows jobs that take up to 3 days. The partition consists of 7 nodes, each with 4 GTX Titan Xs, 32 CPUs, and 62GB of memory. Given the pre-training set is much larger than our training set, we increased the number of CPUs we used to speed up pre-processing, and increased memory used to ensure we could cache the entire dataset in memory. We used 1 GTX Titan X, 32 CPUs, and 62GB of memory for each job. We expect we could have achieved equivalent results in the same time with fewer CPUs and less memory, but given the partition was idle we used all the resources available to ensure there was no bottleneck. For fine-tuning we also used the General

Usage partition, but for each job we only used 1 GTX Titan X, 1 CPU, and 12GB of memory.

9.4 Results

9.4.1 Performance compared to Best Model

In figure 9.1 we show the results of these experiments. We found that fine-tuning the pre-trained network does lead to better performance. But the fine-tuned network performs worse than our best model from chapter 8. This means that the embeddings learnt by pre-training are worse than those learnt without pre-training. This suggests that there is a lot more noise in the machine-generated labels in YFCC100M, and consequently this results in learning a worse embedding despite increasing the amount of training data.

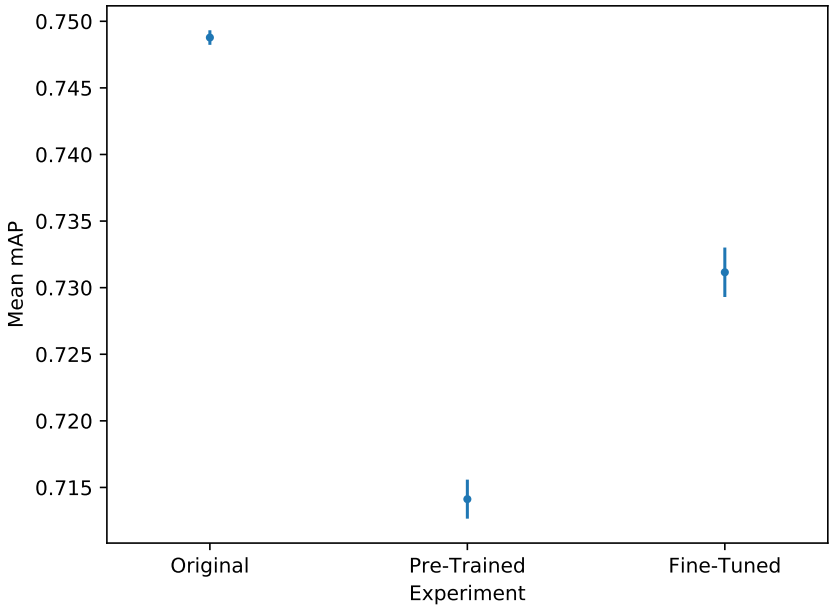

Figure 9.1: Performance on validation set of original model from chapter 8 in comparison to proposed improvements in this chapter

9.4.2 Performance for English vs Non-English

In figures 9.2 and 9.3 we show the performance of the model on English user tags and non-English user tags separately. We grouped all languages other than English as the corpus is dominated by English, and consequently there were too few test examples to analyse the model's performance on individual languages. The test set was separated into English and non-English using CLD2 [10], and any images where the classifier was not sure of the language were excluded.

For some user tags the wrong language is detected, meaning the test sets are not perfectly separated. This is for the same reasons as we discussed in section 4.2.1, with CLD2 not being designed for short text. But the majority of the samples are separated correctly. The number of images in each subset is summarised in table 9.1.

	English	**Non-English**	**No Language Detected**
Images	46,445	7,476	32,746

Table 9.1: Number of images in subsets of test set separated by language

It is not fair to compare the two figures to each other given the English and non-English test sets consist of different samples, and the English test set contains many more samples than the non-English dataset. In the non-English test set 40 out of 500 class have no examples, and consequently they are not included when computing *mAP*. Whereas in the English test set there are no examples for only 3 classes. This might explain why the mean *mAP* is higher for images without English user tags.

In both figures we see the best model we found in chapter 8 outperforms our fine-tuned model. This suggests that our hypothesis that fine-tuning helps learn better embeddings for non-English user tags is false.

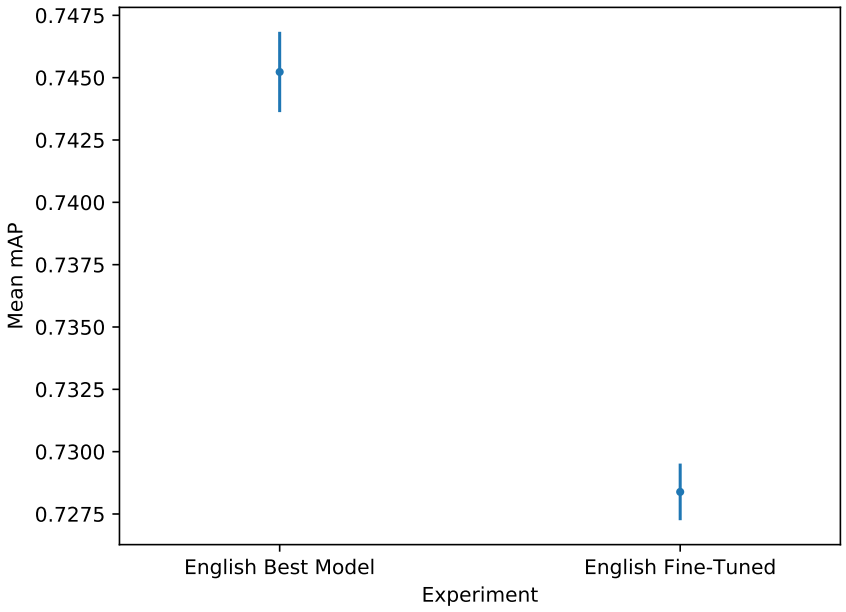

Figure 9.2: Performance for English user tags on best model from chapter 8 in comparison to fine-tuned model

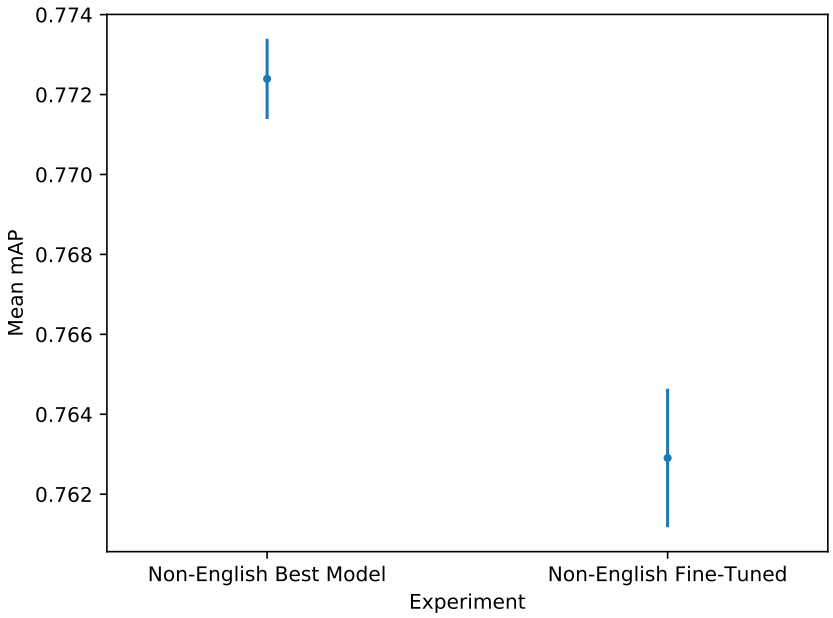

Figure 9.3: Performance for non-English user tags on best model from chapter 8 in comparison to fine-tuned model

Chapter 10

Analysis of Best Model

In this chapter we perform analysis on the test set of the best performing model we found in chapter 8. This model is described in table 8.2.

10.1 Performance Against Baselines

In this section we compare the performance of the best model against the baselines we proposed in chapter 7.

Figure 10.1 shows the performance of the best model in comparison to the baseline of predicting the most likely class which we proposed in section 7.2. Our model significantly outperforms the baseline. This indicates that the user tags do help identify the objects in the image, and our model is able to learn the objects identified by user tags despite the noise.

Figure 10.2 shows the performance of the best model in comparison to the machine-generated labels baseline we proposed in section 7.1. As we discussed in the section, we floor our model's predictions that are less than 0.5, to make it a fairer comparison to the machine-generated labels which don't include labels less than 0.5. As we would expect, the image classifier significantly outperforms our model.

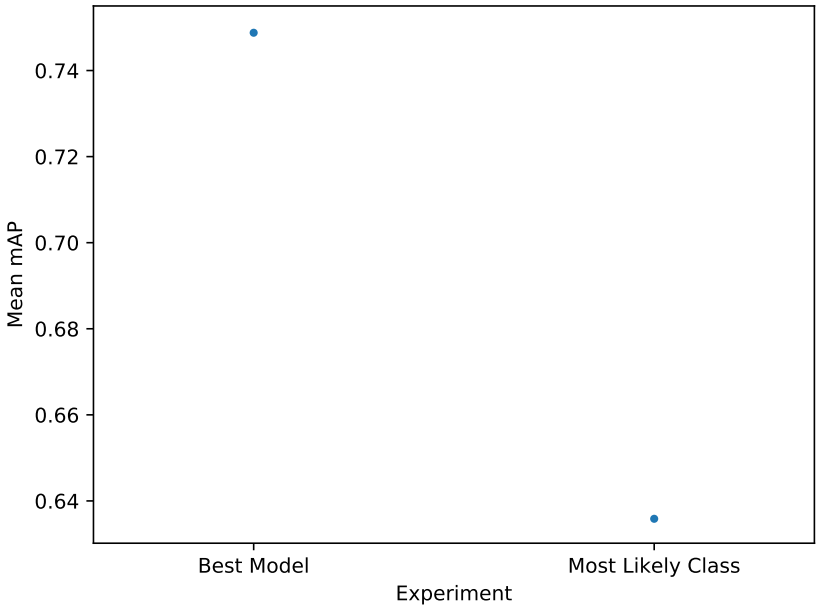

Figure 10.1: Performance on test set of best model in comparison to predicting most likely class

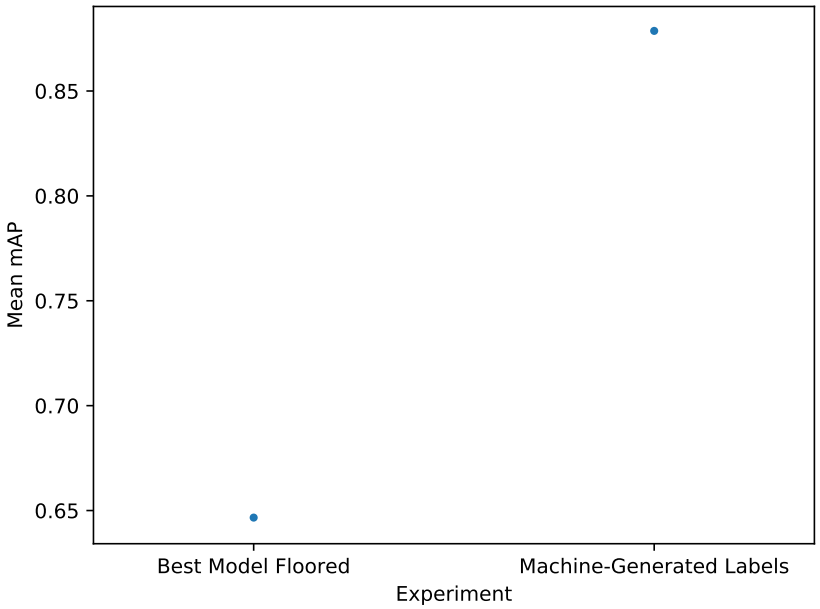

Figure 10.2: Performance on test set of best model with predictions less than 0.5 floored to 0 in comparison to the machine-generated labels

10.2 Performance for Individual Classes

10.2.1 Best vs Worst Classes

We show the 250 classes our model performs best on in figure 10.3, and the 250 classes our model performs worst on in figure 10.4.

We see our model performs best on objects that would typically be the focus of the image e.g. vehicles and animals, and it performs worst on objects that typically occur in the background e.g. bench and bathtub. This is what we would expect considering the observation of Izadinia et al. [19] that users often omit user tags of objects that are not the focus of the image (as discussed in section 4.2.2).

Figure 10.3: Top 250 best scoring classes, the size of each word is proportional to the class's mean mAP

10.2.2 Classes where Model outperformed Image Classifier

Looking at the performance of our model by class, we discover that for three classes our model outperforms the image classifier baseline. These are displayed in table 10.1. For "Monkey" and "Trousers" it only slightly outperforms the baseline, but it significantly outperforms the image classifier for "Animal", achieving a 0.09 higher mean AP. We believe this is likely because when animals are featured in images they are usually the focus, and hence users usually provide tags describing them.

10.2.3 Performance by Class's Level in Hierarchy

Given the observation of Izadina et al. that user tags corresponding to entry-level classes are least often omitted, we would expect our model to perform better for classes

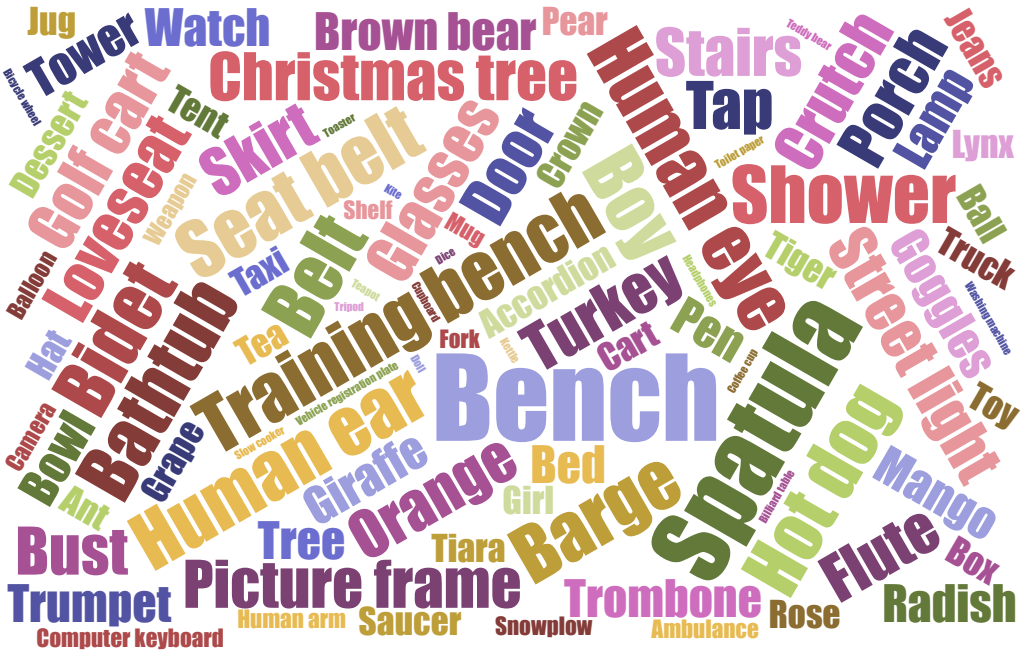

Figure 10.4: Top 250 worst scoring classes, the size of each word is inversely proportional to the class's mean mAP

	Best Model		Machine-Generated Labels
Class Name	**Mean AP**	**Standard Deviation AP**	**AP**
Animal	0.94691	0.00173	0.85967
Monkey	0.77323	0.00820	0.75860
Trousers	0.66724	0	0.65444

Table 10.1: Classes where best model outperformed machine-generated labels baseline

that are higher in the hierarchy of the dendrogram (shown in figure 3.4). To determine whether this is the case we plot the mean AP for each class by the level it occurs in the hierarchy (with 0 being the first level below the root of the dendrogram, which is "entity"). The results of this can be seen in figure 10.5. We plot the floored model predictions (as discussed in section 7.1) to allow us to compare the distribution to that of the machine-generated labels. The distribution for the machine-generated labels is shown in figure 10.6.

Looking at our model's performance alone, we do not observe better performance on labels higher in the hierarchy, and see that performance is more consistent for labels lower in the hierarchy with a tighter grouping. Looking at our model in comparison to the machine-generated labels, both have very similar distributions. In case this was being distorted by the densely packed points, we also plotted a single average mAP and standard deviation for each level, but we observed the same pattern as we see in these figures.

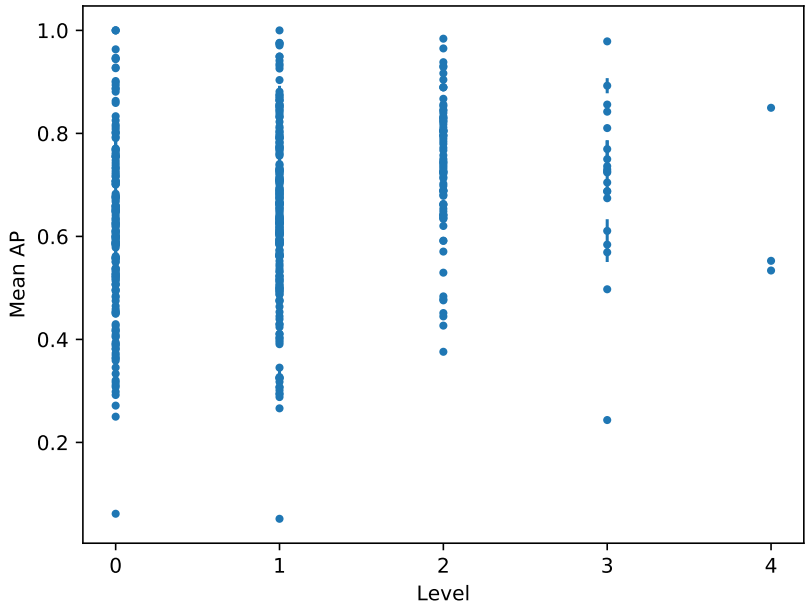

Figure 10.5: Performance of best model on individual classes by their level in dendrogram

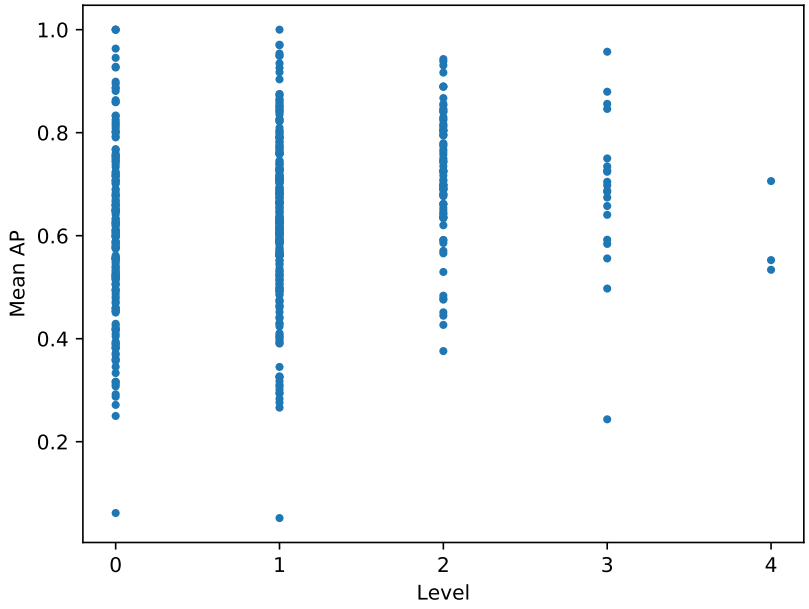

Figure 10.6: Performance of machine-generated labels baseline on individual classes by their level in dendrogram

This suggests that user tags are not better at classifying entry-level classes as we'd expect, but perform similarly to our image classifier baseline for per-level performance.

This seems another indication that our model is learning to predict classes from user tags beyond just predicting the class when the user tag with the same name appears, given that the user tags for fine-grain objects are often omitted.

Chapter 11

Experiments we did not have time to complete

In this chapter we discuss other ways to tackle the challenges of natural language (introduced in section 4.2.2), for which we ran out of time to conduct any experiments.

11.1 Stemming

As we briefly discussed in section 8.2.3, we considered applying stemming to the user tags, but there were challenges associated with this due to the multi-lingual corpus. The rules for stemming differ by language, hence we need to detect the language, which we could do using CLD2 [10]. But languages can be falsely detected, and detecting the language for short text is challenging, as we discussed in section 4.2.2.

Stemming words using a stemmer for a different language than they belong to will result in the words being stemmed incorrectly, introducing noise in the dataset. Also for a third of images we are unable to detect the language from the user tags, meaning we will not be able to stem them. Considering all of this stemming words in a traditional manner is likely to result in a larger vocabulary, with fewer examples for each user tag.

We thought of two approaches that may work better. Our first idea was to take a dictionary-based approach, querying the user tags in monolingual dictionaries to determine which language each belongs to, and then stemming using their determined language. Another idea we had was to count the number of times each user tag was detected as belonging to each language, and then assume across the whole dataset that each word belonged to the language that it was detected as most frequently. The second approach is advantageous over the first as it is more flexible, with it being able to determine the language for words when they are spelt incorrectly or absent from any dictionaries. But a downside to it is that the language detector may detect languages incorrectly. A downside to both approaches is that they do not handle the issue that languages often share the same words. In such cases we would have to assume the word belongs to the most frequent language in the corpus, which would create bias against under-represented languages.

11.2 Translation

We also thought about translating non-English words to English. We hypothesised this could improve performance given there are many more examples for English user tags, hence we are likely to learn better embeddings for words in English. If words were translated correctly this should help improve performance for under-represented languages. But in order to translate words we need to know their language, hence we run into the same language detection problem we discussed in section 11.1. This approach also has a bigger risk of introducing noise, as often words can have different meanings depending on their context, hence translation could result in translating the word to one with an entirely different meaning.

Translation seems more likely to introduce noise than stemming, hence there is a greater risk of translation making performance worse. We think it would be worth researching this further, but with a focus on how to reduce the noise introduced. Potentially a different approach could achieve the same effect, for example attempting to determine which user tags are related using the similarity of their embeddings, and using this to cluster them.

Chapter 12

Conclusion

In the introduction we outlined five aims that would allow us to test our hypothesis. Below we discuss for each aim whether or not we met it, and why.

1. Construct a dataset which contains user tags and human-verified labels for images.

This goal was achieved. We constructed a training set with 1.8 million images with user tags, along with 9.8 million human-verified labels.

2. Develop a feature extractor for user tags.

This goal was achieved. We proposed an architecture using an embedding layer followed by a pooling layer, which reduced the input user tags into a single feature vector. Our classifier made predictions about the contents of images using only this feature vector. This model outperformed our naive baseline of just predicting the most frequent observation in the training set of whether or not the class was present. This suggests that our proposed architecture is successful at extracting features from user tags.

3. Develop a network that combined the features extracted from user tags with those extracted from images to classify the images contents.

This goal was not achieved. The engineering work was more challenging than we had expected, and would not have been achievable in the time frame given the computation resources and our experience with TensorFlow. As a result of this failure early on, we refocused our project on predicting the classes using only user tags, which we achieved.

4. Reduce the impact of user tag noise on classification.

This goal was achieved. We applied max pooling to the embeddings of our user tags. This resulted in an increase in performance over average pooling. Given that more information is lost through max pooling than average pooling, this suggests that max pooling was useful for reducing noise. We also discarded all user tags which occurred fewer than 10 times in training set. This reduced the size of the vocabulary by a factor

of 10, and as a result we only included user tags in the vocabulary where we had enough examples to learn which class they indicated.

5. Improve classification for under-represented languages.

This goal was not achieved. Our first approach to this of pre-training on a larger dataset did not improve performance on under-represented languages. We started work on two other approaches, but ran out of time to conduct experiments for them. We attributed running out of time for this aim down to over-ambition. We under-estimated how long the engineering work would take for this project, which meant we did not have time to conduct all the experiments we would have liked to.

We hypothesised that:

- An image classifier with the joint knowledge of the image and user tags would outperform an image classifier that could only make predictions using images.

Our failure to meet our aim of integrating user tags in an image classifier meant we could not answer our hypothesis. However our results did give us an indication of an answer. Using user tags we were able to exceed the performance of an image classifier for predicting ”animal”, hence a classifier that uses the image and user tags to predict the class should perform better than one that only uses the image. This would suggest at the very least our hypothesis is correct for some classes, but we would need to conduct further experiments to confirm this and learn how many classes it improves.

The main focus of future work should be to experiment with one of the architectures we proposed in order to definitively answer our hypothesis. As we discussed in chapter 11, we also believe a good direction for future research would be in further tackling the challenges introduced by the corpus being multi-lingual.

Bibliography

[1] astrangelyisolatedplace. Rrrrrooooaaaaadtripppppp! https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=detection&c=%2Fm%2F0bt9lr&id=5d76f6c05fa69465, 2015. [Online; accessed March 23, 2020].

[2] Hessam Bagherinezhad, Maxwell Horton, Mohammad Rastegari, and Ali Farhadi. Label refinery: Improving imagenet classification through label progression, 2018.

[3] Devansh Bisla and Anna Choromanska. Visualbackprop for learning using privileged information with cnns, 2018.

[4] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, pages 1–9, 2009.

[5] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data, 2018.

[6] dnuffer. open_images_downloader. https://www.flickr.com/services/api/misc.urls.html, 2017. [Online; accessed December 3, 2019].

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[8] Flickr. Photo source urls. https://www.flickr.com/services/api/misc.urls.html, 2019. [Online; accessed November 29, 2019].

[9] Ross Girshick. Fast r-cnn, 2015.

[10] Google. Compact language detector 2. https://github.com/CLD2Owners/cld2, 2013. [Online; accessed March 29, 2020].

[11] Google. Open images challenge 2019. https://storage.googleapis.com/openimages/web/challenge2019.html#object_detection, 2019. [Online; accessed March 25, 2020].

[12] Google. Open images challenge object detection evaluation. https://storage.googleapis.com/openimages/web/evaluation.html, 2019. [Online; accessed March 26, 2020].

[13] Google. Open images trained models. https://storage.googleapis.com/openimages/web/extras.html, 2019. [Online; accessed December 27, 2019].

[14] Google. Overview of open images v5. https://storage.googleapis.com/openimages/web/factsfigures_v5.html, 2019. [Online; accessed March 23, 2020].

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[18] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors, 2016.

[19] Hamid Izadinia, Bryan C. Russell, Ali Farhadi, Matthew D. Hoffman, and Aaron Hertzmann. Deep classifiers from image tags in the wild. In *Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions*, MMCommons '15, page 13–18, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Alireza Koochali, Sebastian Kalkowski, Andreas Dengel, Damian Borth, and Christian Schulze. Which languages do people speak on flickr? a language and geo-location study of the yfcc100m dataset. In *Proceedings of the 2016 ACM Workshop on Multimedia COMMONS*, pages 35–42, 2016.

[21] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[23] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018.

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[25] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining, 2018.

[26] Irene Mei. Maya and rudy. https://www.flickr.com/photos/41069398@N08/8222923933/, 2012. [Online; accessed March 27, 2020].

[27] V. Ordonez, J. Deng, Y. Choi, A. C. Berg, and T. L. Berg. From large scale image categorization to entry-level categories. In *2013 IEEE International Conference on Computer Vision*, pages 2768–2775, 2013.

[28] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[30] Sergio Guadarrama, Nathan Silberman. TensorFlow-Slim: A lightweight library for defining, training and evaluating complex models in tensorflow. https://github.com/google-research/tf-slim, 2016.

[31] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[32] Noah A. Smith. Contextual word representations: A contextual introduction, 2019.

[33] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era, 2017.

[34] James Sutton. Deep dive into object detection with open images, using tensorflow. https://algorithmia.com/blog/deep-dive-into-object-detection-with-open-images-using-tensorflow, 2017. [Online; accessed March 28, 2020].

[35] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[36] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m expansion packs. https://multimediacommons.wordpress.com/yfcc100m-expansion-packs/, 2016. [Online; accessed March 28, 2020].

[37] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Commun. ACM*, 59(2):64–73, January 2016.

[38] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5):544 – 557, 2009. Advances in Neural Networks Research: IJCNN2009.

[39] Angelo Vittorio. Toolkit to download and visualize single or multiple classes from the huge open images v4 dataset. https://github.com/EscVM/OIDv4_ToolKit, 2018.

[40] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2016.

[41] I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification, 2019.

[42] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017.