COCO Multilabel Classification

Nick Rhodes **Roger Hong** **Alex Mirrington**

1 Introduction

Recent advances in deep learning have seen an immense growth in the capability for image classification, and have resulted in a plethora of image classification models pretrained on datasets such as ImageNet [2], which can be tailored to domain-specific tasks. However, the deep learning community has primarily focused on multi-class classification problems, which seek to assign a single class to each image. Many useful tasks such as object detection and image tagging require assigning multiple labels to an image, which is a more complicated problem. Not only does the multi-label classification problem involve determining which labels are the most relevant to an image, but the model must also decipher the appropriate number of labels to output. Investigating the multi-label classification problem is important. With the recent saturation of image classification models on popular datasets, focusing research efforts on improving multi-label image classification may help unlock new deep learning techniques which push the field forward.

In this study, we aim to investigate the most appropriate method to perform multi-label classification on an image dataset, with associated captions. More specifically, given the prevalence of powerful pretrained models, we consider the best pretrained model for the given task and determine the most effective way to augment its predictions for our own task. We also aim to investigate the potential of multi-modal models to combine image and caption data for improved prediction.

Our investigation firstly comprises of a thorough analysis of the provided dataset to determine useful characteristics for selecting a pretrained model to augment. Since there is correspondence between labels in the dataset and objects in the images, we particularly examine the use of pretrained object detection models, and demonstrate superior performance for our task over CNN based approaches. Given the output scores of the pretrained model, we examine the use of a linear layer, which maps to a vector of correct dimensionality, to assist in predicting labels for our task. Our approach also entails natural language processing techniques on the captions to assist the object detection model in a multi-modal model. Several approaches for combining information from the different modalities are explored, with a demonstrated improvement from incorporating this additional information. Hyperparameter searches for the appropriate threshold to exceed before outputting a score as a predicted label are also throroughly conducted. Ultimately, using these techniques we achieved a weighted F1 score of 0.9120 on our validation set.

2 Related Works and Methods

In this section we introduce the key related work in image classification, object detection, and natural language processing techniques that form the bases of our approach, alongside an overview of the techniques we utilised and evaluated to determine the most appropriate model for this multi-label classification task. Finally, we describe our multi-modal model, which combines our object detection and natural language processing approaches in a novel way to achieve superior performance.

2.1 Dataset

As with most classification tasks, the first step we took towards building a model was to investigate the properties of the dataset. In total, the dataset is split into 30k training and 10k test samples, with each sample containing an image and a brief caption summarising the contents of the image as input. Samples in the training set are additionally paired with up to 20 labels, each of which represent an object in the image. Further data analysis as seen in Figure 1 showed that all samples in the training dataset were associated with between 1 and 7 labels, with a heavy skew towards fewer labels over more labels.

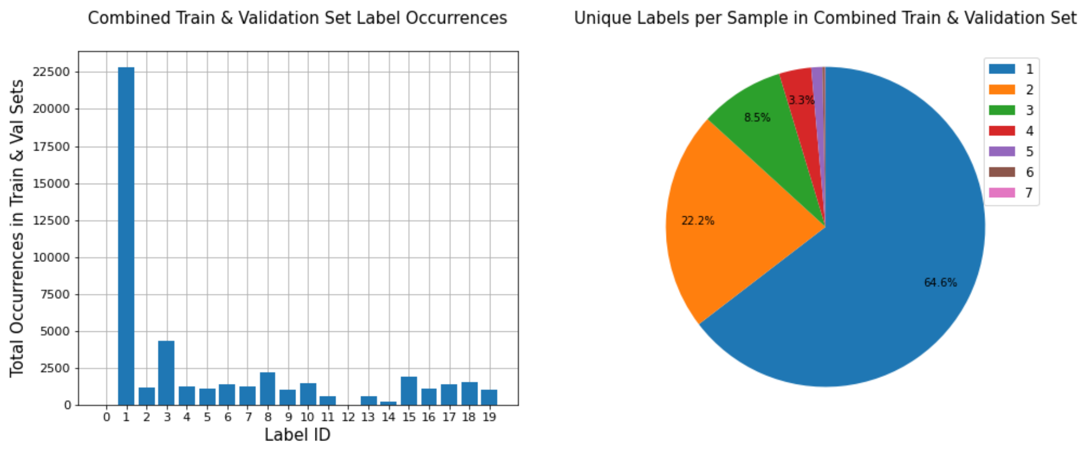


Figure 1: *Left:* Label count across all samples in the combined training and validation sets. *Right:* Number of labels per image for all samples in the combined training and validation sets.

Given the dataset comprised images and captions, we suspected it was a subset of the COCO dataset [16]. We determined it was indeed a 40k sample subset of COCO without object, stuff and panoptic segmentation data or person keypoints. Knowing this, we felt much more comfortable searching for pre-trained models that we could adapt, and settled on the models in the `torchvision` package. The CNN models in this package were trained on the ImageNet dataset [2], so were able to be refined for use on our dataset, and the Faster R-CNN model was already trained on the `train2017` set of COCO so needed little refining to achieve reasonable results. More details of how we adapted these models will be presented in Sections 2 and 3.

One other major aspect of the dataset is its heavily imbalanced label distribution, as illustrated in Figure 1. Label 1 accounts for just shy of 49% of all labels in the dataset, and some labels (0, 12) are not present in the training set whatsoever. This leaves us with two problems to solve when developing and evaluating our model:

1. The training and validation sets need to have similar distributions in order for training performance to transfer well to validation set performance. The distributions of labels in the training and validation sets are shown in Figure 2. Whilst we have no control over the label distribution in the test set, out testing results are comparable to our validation set results, indicating a similar label distribution between the train, val and test tiers.
2. Macro evaluation metrics are not useful on datasets with uneven label distributions, as they average per-class metrics with equal weight. We opted for weighted and micro F1 measures to evaluate our models, as they account for uneven label distributions where macro F1 does not. This is evidenced by the much lower macro F1 scores of all models presented in section 3 when compared to the micro and weighted F1 for the

same models. Details of these evaluation metrics are presented in Section 2.2.

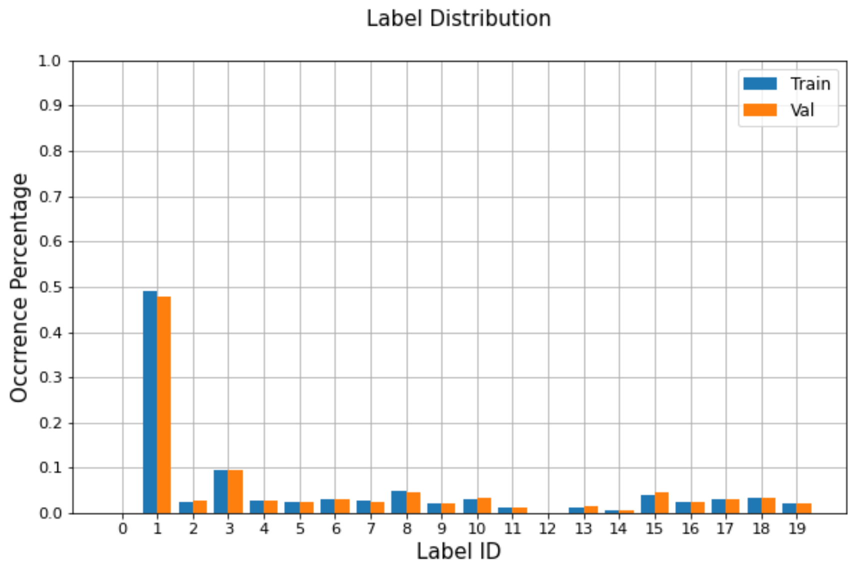


Figure 2: Distribution of labels for the training and validation sets. The training set contains a total of 27000 samples and 41768 labels, whist the validation set contains 3000 samples and 4770 labels.

2.2 Evaluation Metrics

2.2.1 F1 Measure

As mentioned previously, macro-type evaluation metrics to not give a good indication of model performance for datasets with imbalanced label distributions. This is because they average per-class metrics without any consideration for label distribution. We can define the per-class F1, precision and recall in terms of the number of true-positive, false-positive and false-negative samples for a class c as follows:

$$F1_c = \frac{2P_cR_c}{P_c + R_c} \quad \text{where} \quad P_c = \frac{tp_c}{tp_c + fp_c} \quad \text{and} \quad R_c = \frac{tp_c}{tp_c + fn_c}$$

The macro F1 measure is then defined for the set of all classes C as:

$$F1_{\text{macro}} = \frac{1}{|C|}\sum_{c \in C} F1_c$$

In the case of our dataset, if we perfectly identified all class 1 samples but misidentified all other samples, $F1_1 = 1$ but $F1_i = 0 \; \forall \; i \in \{C - 1\}$, leading to a low F1 score despite having identified close to 49% of all labels. The weighted F1 measure takes the distribution of labels into account, and the micro F1 measure uses the total number of true-positive, false-positive and false-negative samples samples directly, as shown below:

$$F1_{\text{weighted}} = \frac{1}{\sum_{c \in C} n_c}\sum_{c \in C} n_c F1_c, \quad \text{where } n_c \text{ is the number of true instances of class } c$$

$$F1_{\text{micro}} = \frac{2P_{\text{micro}}R_{\text{micro}}}{P_{\text{micro}} + R_{\text{micro}}}, \quad \text{where } P_{\text{micro}} = \frac{\sum_{c\in C} tp_c}{\sum_{c\in C} tp_c + fp_c} \quad \text{and } R_{\text{micro}} = \frac{\sum_{c\in C} tp_c}{\sum_{c\in C} tp_c + fn_c}$$

It should be noted that these formulas apply to both multi-class and multi-label classification tasks, with the key difference being that in multi-class tasks, classes are mutually exclusive so n_c is equal to the number of samples with class c, whereas in multi-label tasks a sample can have multiple labels/classes, so n_c is the total number of occurrences of the label across all samples and thus $\sum_{c\in C} n_c$ is not necessarily equal to the number of samples in the dataset. We take both the weighted and micro F1 scores into account when evaluating out models, and also present macro F1 values to illustrate that macro F1 performance is not sufficient for proper model evaluation.

2.3 Multi-label Loss Functions

Classifying multi-label data is more complex than single-label classification; for a single-label image classification task, it is common practice to output a softmax probability distribution across all possible classes in the dataset, and compute the categorical cross-entropy loss between the softmax distribution and a one-hot encoding of the true class for each sample in a batch. For multi-label classification tasks like the object existence task we have here, there are two key factors to consider when designing a multi-label classification model:

1. Computation and back-propagation of loss for multiple labels. A common approach is to encode target labels as a multi-hot vector.
2. Retrieving predicted labels from the output activations of the network. Multi-label loss functions may not output a probability distribution across labels, and it is often the case that the correct number of labels to output is unknown.

We used an unweighted binary cross entropy (BCE) loss function for our experiments, due to its simplicity and ability to handle multi-label data. Each of our models outputs a 20-dimensional vector for each sample in the batch, and the BCE loss is computed between the output vector x and a multi-hot encoding y of the true labels as follows:

$$L = -\frac{1}{D}\sum_{i=1}^{D} y_i \log x_i + (1-y_i)\cdot\log(1-x_i), \quad \text{where } D \text{ is the dimension of } x \text{ and } y \textit{ i.e. } 20$$

This loss function is especially efficient to compute, since it is used in combination with a sigmoid activation function in the last layer; in doing so, we simplify derivative calculations by avoiding computations of exponentials and logarithms as per the implementation of the PyTorch `BCEWithLogitsLoss` module [24].

2.4 An Initial Baseline: Convolutional Neural Network (CNN) Architectures

With an increasing number of scientists participating in solving the challenge in image recognition, research on different CNN architectures has been significantly promoted. Since the success of some applicable CNNs such as LeNet [15], several architectures have become popular and commonly used to develop deeper CNNs.

In our experiments, we attempted some baseline experiments using AlexNet, DenseNet, VGGNet, ResNet and ResNext. We trained a linear layer mapping from the output features of the CNN to scores for the 20 labels, with the aim of verifying whether a CNN-based approach was feasible. Overview figures of each of them which demonstrate their network configuration will be provided in this section.

2.4.1 LeNet

LeNet, as known as lenet-5, is one of the simplest and earliest convolutional network proposed by Yann LeCun et al. [15] in 1998. This architecture contains two sets of convolutional layers combined with max pooling layers. It is then followed by two fully-connected layers and a Softmax layer (See Figure 3). Although we did not experiment on LeNet since it can only deal with simple digit images, it is worth to mention LeNet since it is one of the most classic architecture we can refer to.

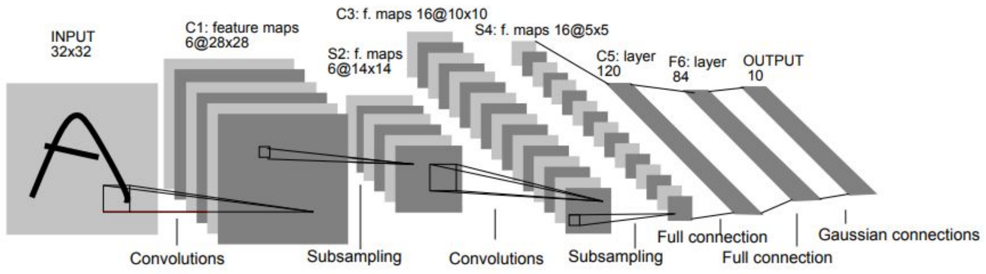


Figure 3: Overview of LeNet-5 architecture [15]

2.4.2 AlexNet

Considered as the architecture which made convolutional network popular in computer vision, AlexNet has a larger and deeper network. It contains five convolutional layers and three fully-connected layers [13], as indicated in Figure 4. Furthermore, AlexNet replaced tanh function in LeNet with Rectified Linear Units (ReLU), which adds non-linearity. It also enables overlapping pooling and supports trainings on multiple GPUs.

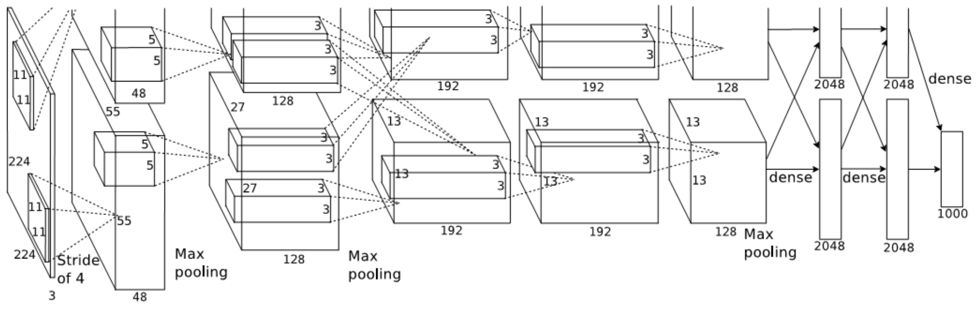
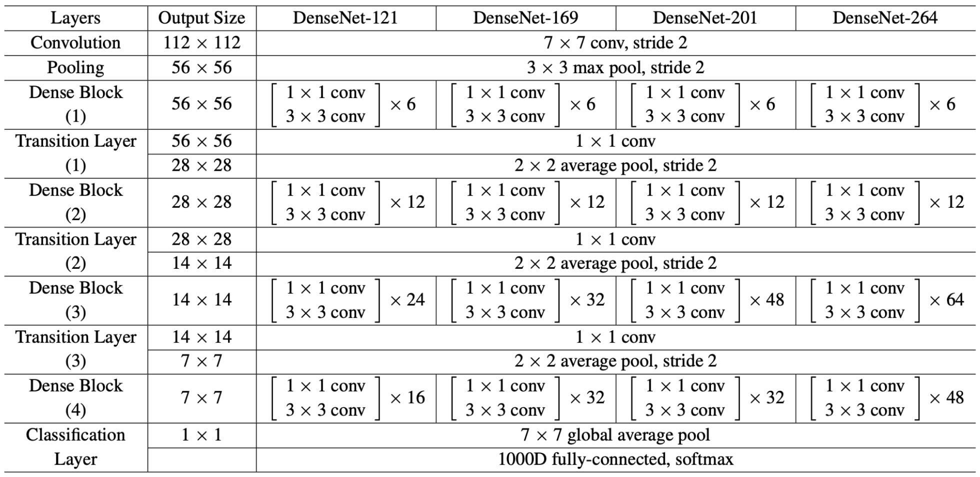


Figure 4: Overview of AlexNet architecture [13]

2.4.3 DenseNet

DenseNet was born with several advantages in enhancing feature propagation, reducing the overall size the parameters and mitigate the effect of vanishing-gradient problem. Instead of applying summation, it uses the technique as concatenating outputs from previous layers. It is easier to train since the the flow of gradients across the network is improved [10]. Figure 5. Each Dense block is composed of a batch normalisation, a ReLU activation and 3×3 convolution.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer	56×56	1×1 conv			
(1)	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer	28×28	1×1 conv			
(2)	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer	14×14	1×1 conv			
(3)	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification	1×1	7×7 global average pool			
Layer		1000D fully-connected, softmax			

Figure 5: Configuration of DenseNet architecture [10]

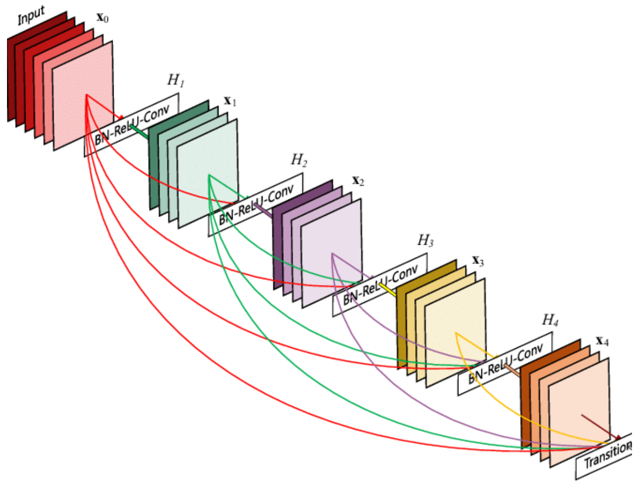
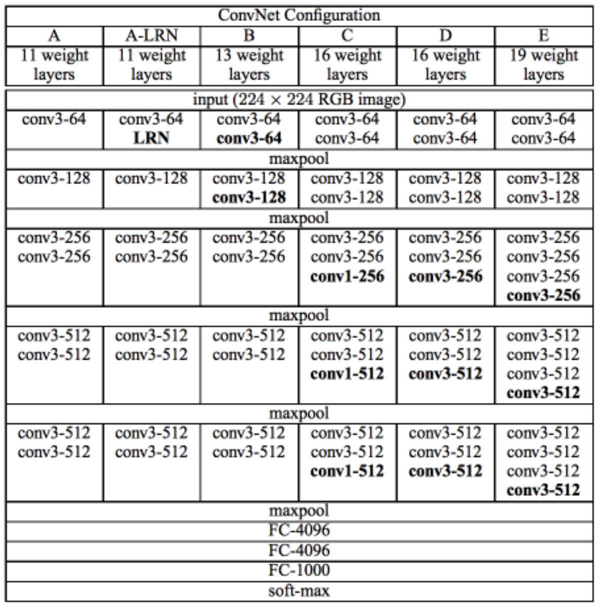


Figure 6: A dense block with 5 layers which commonly applied in DenseNet [10]

2.4.4 VGGNet

Following how AlexNet is constructed, 13 convolutional layers and 3 fully-connected layers are included in VGGNet, as demostrated in Figure 7. This architecture showed that how deep a CNN is can have critical impacts on the model's performance. Its good performance is mainly due to that fact that it uses large amount of memory and lots of parameters, as well as, smaller size of filters [22].

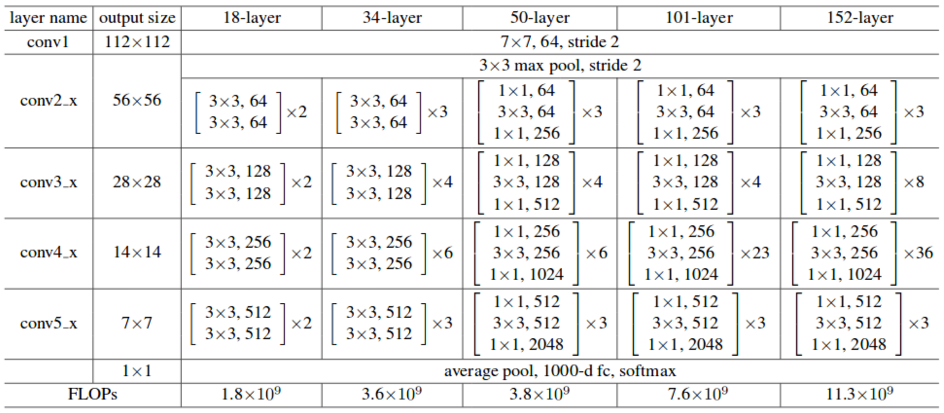


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	LRN	**conv3-64**	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
			conv1-256	**conv3-256**	conv3-256
					conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 7: Configuration of VggNet architecture [22]

2.4.5 ResNet

Residual network (ResNet) was developed by Kaiming He et al. in 2015 [8]. In ResNet, some connections are skipped, and large amount of batch normalisation are prevented. Important contribution it has made is that it allows deeper CNN without compromising the generalisation power of the model. Figure 8 shows the configuration of such network. A new critical component was introduced in this model, called residual block 9. With those residual connections, the training of some layers can be skipped.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 8: Configuration of ResNet architecture [8]

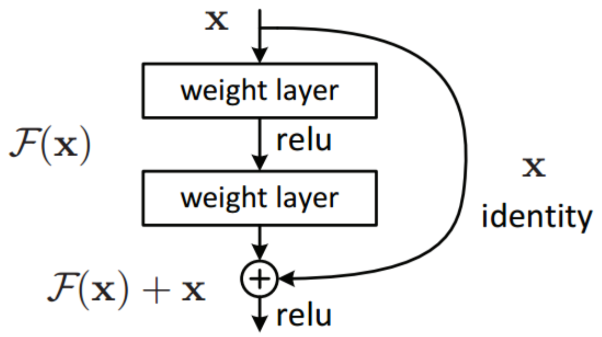
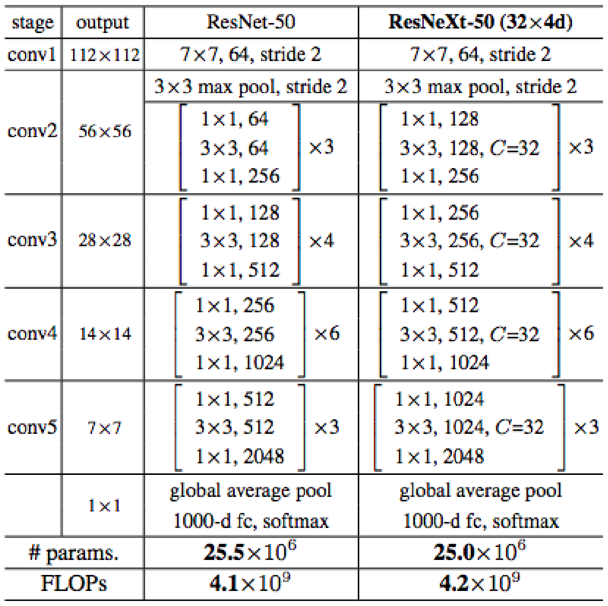


Figure 9: Residual learning: a building block [8]

2.4.6 ResNeXt

ResNeXt is similar to ResNet, but is also inherited from ResNet, VGGNet, and Inception. The key of ResNeXt is cardinality. Figure 10 shows the differences between ResNet and ReNeXt. In the table, $C = 32$ refers to the setting of "cardinality", which refers to the number of paths inside the ResNeXt block. By increasing cardinality, the performance of the model can be reinforced. Within a single module, it includes 32 same topology blocks, which leads to less usage of paramaters [28].



stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**$\times10^6$	**25.0**$\times10^6$
FLOPs		**4.1**$\times10^9$	**4.2**$\times10^9$

Figure 10: Comparison of ResNet and ResNeXt regarding to configuration [28]

2.5 Object Detection and Segmentation Models

With the recent saturation of regular CNN architecture performance on popular datasets such as CIFAR [12] and ImageNet [2], the deep learning community has turned to more complex computer vision tasks like object detection and segmentation. These tasks move beyond classification tasks, requiring models to isolate and label common objects from images. In the following section, we will describe the famous Region CNN (R-CNN) model [4], which achieved an improvement of over 30% over existing state-of-the-art models on the VOC 2012 dataset [3] in 2014. We will then discuss further improvements made to the R-CNN model

[4, 21, 7] for both object detection and segmentation tasks.

2.5.1 R-CNN

The Region CNN (R-CNN) model leverages the vision capabilities of CNN architectures for object detection tasks, using a selective search methods [26] to generate multiple region proposals to inspect for objects, motivated by the need to overcome the large variance in the scale and position of individual objects within an image. By resizing each region proposal to a fixed size (227×227) using a simple resize operation or other transformation pipeline, a pre-trained CNN architecture is used to yield image features, which are in turn passed to a one-vs-rest linear SVM ensemble. The flexibility of the R-CNN object detection process is highlighted in Figure 11; the region proposal extraction, feature extraction and region classification steps are not limited to the methods shown in the original paper, and can be adapted depending on the dataset and application of the object detection model.

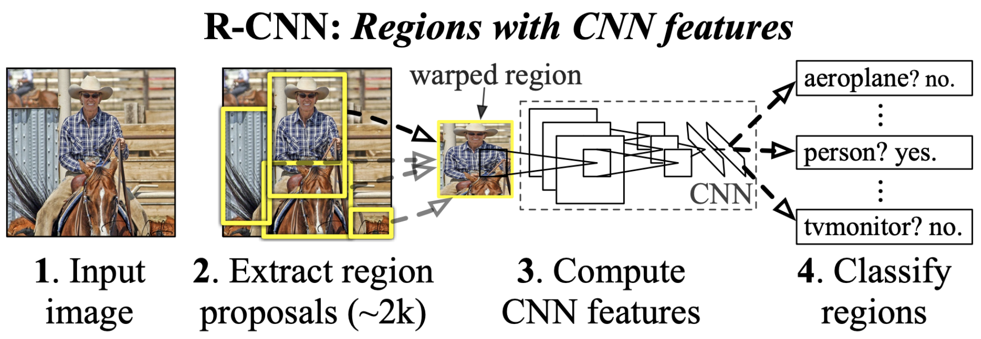


Figure 11: Overview of the key steps of the R-CNN model [5]

In addition to its flexibility, the R-CNN model takes advantage of shared CNN weights across all region proposals to reduce training time and model space complexity, only branching the training process in order to handle multi-label outputs with its SVM ensemble. Nonetheless, the model has its drawbacks; since region proposals are made in the input image pixel space, forward and backward passes have to be computed for all layers of the CNN for each one of the region proposals. Whilst somewhat mitigated by the pre-training of the CNN, the training time of the model is significantly larger when compared to a regular CNN of comparable size.

2.5.2 Fast R-CNN and Faster R-CNN

Girshick addresses the shortcomings of the R-CNN model in his Fast R-CNN model [4], passing each image through several convolutional layers *once*, and then computing region proposal features from the convolved features and using a region of interest (RoI) max pooling layer to ensure each convolved RoI is of the same dimension ready for input to a fully-connected layer. This process is summarised in Figure 12.

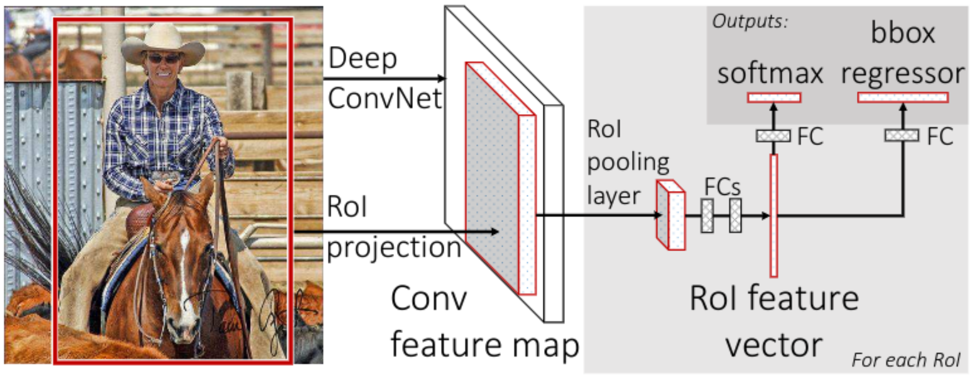


Figure 12: Overview of the key steps of the Fast R-CNN model [4]

Whilst novel, this approach still suffers from the amount of time taken to compute which regions in the image to inspect for objects, as over 2000 region proposal bounding boxes still need to be generated via some method such as the selective search described earlier. The Faster R-CNN [21] model aims to solve this issue by introducing a region proposal network (RPN), which slides different sized windows across the convolved feature maps of the CNN, and predicts a score and co-ordinate offsets for different regions. This process is illustrated in Figure 13.

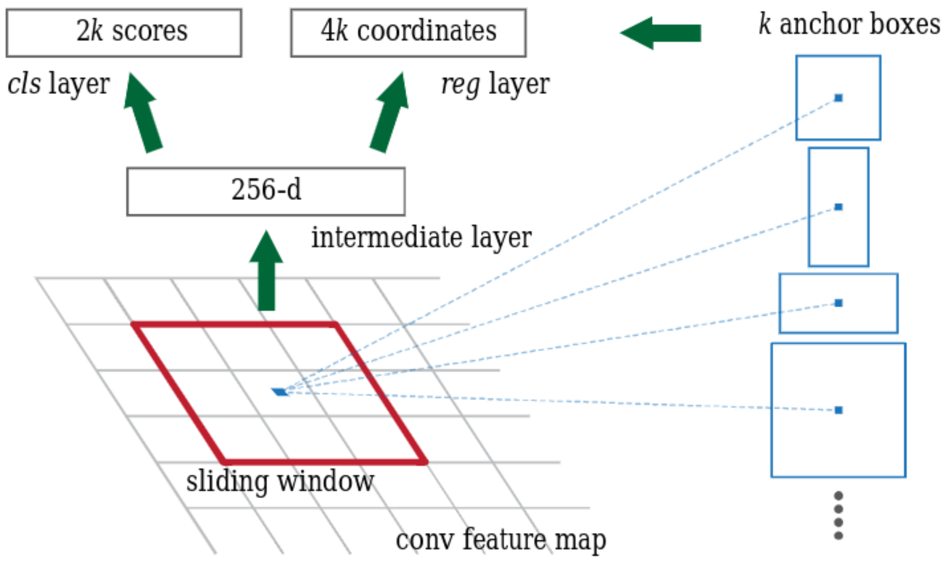


Figure 13: Region proposal network used in the Faster R-CNN model. [21]

In addition to the speedup that the Faster R-CNN model gains over its predecessors, it also achieves a higher mean average precision on the PASCAL VOC 2012 [3] dataset, and remains one of the most versatile and efficient object detection models to this date.

2.5.3 Application to Our Classification Task

We applied a pretrained Faster R-CNN model to our classification task, the motivation being that the classes in our dataset seemed to correspond to objects in the image *e.g.* class 1 occurs whenever a human is present, and class 19 occurs whenever a horse is present. Although we could not train the R-CNN further since we had no bounding box information in our dataset, it was already pre-trained on the COCO dataset, and manual inspection of the pre-trained model's output labels indicated a one-to-one correspondence between its output labels and the labels in our dataset. This is illustrated in Section 6. Details on the implementation of the model are discussed in Section 3.

As a side note, we decided not to use Mask R-CNN [7] in our model, as its object segmentation capabilities are beyond what is required for our task; we don't leverage the bounding box information returned by Faster R-CNN at all in our models, as we are only concerned about the existence of objects in the image, which can be expressed using the distribution of scores across classes.

2.6 Language Models

The text captions corresponding to each image provide another source of information not exploited by the previously discussed CNN or object detection approaches. Natural language processing using deep neural models has demonstrated potential to recognise entities referred to in text [14, 1] and perform multi-label text classification tasks [17, 11]. Thus, for our investigation we decided to employ a natural language processing model on the captions. Both a simple TFIDF [20] approach and an LSTM [9] approach were tested, and are discussed in the following sections.

2.6.1 TFIDF with Linear Layer

Term frequency-inverse document frequency (TFIDF) [20] is a statistic that can be used to represent text based on the words it contains. It considers not only the occurrence count of each word in the text, but is offset by the number of documents in the corpus that contain the word. The calculated TFIDF statistics can be used to build a vector, where each component describes the relevance of a particular word from the vocabulary to the current document. A typical mathematical formulation for the TFIDF statistic is:

$$TFIDF(t,d) = tf(t,d)\log\left(\frac{n}{df(t)+1}\right)$$

where $tf(t,d)$ is the number of times term t occurs in 'document' d, n is the number of documents, and $df(t)$ is the number of documents in the corpus containing t.

Representing a caption by a TFIDF vector is similar to using a multi-hot vector of word occurrence, but may better capture the importance of words that occur rarely.

For our most simple language model, multi-label scores were produced for a caption by first computing its TFIDF vector, and then passing this output into a linear layer of size $(vocab\ size, num\ classses)$. The activations of this layer are passed through a sigmoid and used in the calculation of Binary Cross Entropy loss (Section 2.3). For extracting predicted labels from the model, a threshold is set on these sigmoid outputs, and any components with activations higher than the threshold are predicted. The results are demonstrated in Section 3. Parameter tuning for the appropriate vocabulary size is also considered.

2.6.2 LSTM

A bidirectional long short-term memory (LSTM) network was also considered. Recurrent models, which apply the same set of weights to multiple elements in a sequence input, are suitable to process text because they can handle variable length input, and are capable of recognising specific information in the text regardless of where it appears in the sequence [6]. At each stage of processing the sequence input the hidden state from the previous timestep is taken as input and adjusted based on the new input, such that after processing the sequence a state representing the entire sequence is obtained. For our task, we are concerned with training the sequence model such that this final hidden representation can be used to predict the labels associated with the caption.

LSTM models improve on the general sequence model framework by reducing vanishing gradient problems resultant from applying the same weights repeatedly, and explicitly allowing better propagation of information over longer sequences [6]. The candidate cell state is calculated as:

$$\widetilde{c}_t = \tanh\left(w_{hh}h_{t-1} + w_{hx}x_t + b_h\right)$$

A sigmoidal input gate controls whether the new input should be accumulated into the hidden state or not:

$$i_t = \sigma\left(w_{ih}h_{t-1} + w_{ix}x_t + b_i\right)$$

A forget gate determines what information is relevant to retain from the previous hidden state:

$$f_t = \sigma\left(w_{fh}h_{t-1} + w_{fx}x_t + b_f\right)$$

Using these two calculated gates, the cell state is updated accordingly:

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c}_t$$

Finally, the output of the LSTM cell can be shut off using an output gate:

$$o_t = \sigma\left(w_{oh}h_{t-1} + w_{ox}x_t + b_o\right)$$

The final hidden state which is passed to the next timestep is given by:

$$h_t = o_t * \tanh\left(c_t\right)$$

To prepare the text data for our LSTM, we first preprocess the text and convert it to a form suitable for input to the model. Punctuation is removed from captions, and they are then tokenized into discrete words. Any words which are considered stop words (the most commonly occurring English words) are removed from the tokens. 100 dimensional GloVe [19] embeddings are used to map words to vectors which capture the word. Using GloVe embeddings is favourable over a one-hot encoding, as it reduces the dimensionality of the input, and helps the model learn since similar words (as determined by their usage in the Wikipedia corpus used to train GloVe) as mapped to spatially close vectors. The caption is passed to the model as the resultant sequence of embeddings.

The final forward and backward hidden states (a bidirectional LSTM was used) are concatenated and passed into a Linear layer of the appropriate size to reduce dimensionality to the number of classes. As for the TFIDF model, these outputs are passed through a sigmoid function and used with Binary Cross Entropy loss to train the model. Predicted labels can be extracted using a threshold value.

Results are demonstrated in Section 3, including experiments on the appropriate hidden state size.

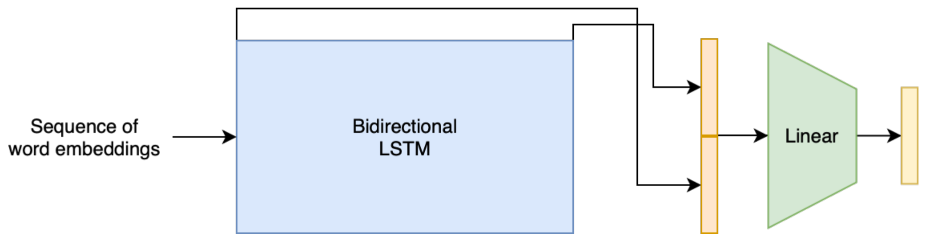


Figure 14: Overview of our BiLSTM language model. The output is passed through a sigmoid function and used as described above.

2.7 Multi-modal Models: Combining Object Detection with Language

Our final model utilises information from both the image and text modalities by combining the predictions of the best object detection and language model (as determined by our results on the individual models in Section 3). We generate scores for each label using the Faster RCNN model described in Section 2.5.2, and similarly for the LSTM language model described in Section 2.6.2. These scores are combined to produce a vector of final output scores for each class, which are passed through a sigmoid and used to train the model with Binary Cross Entropy loss. As for the models discussed above, predictions can be obtained by selecting labels with outputs above a threshold value. We experimented with two methods of combining the scores from the different modality models.

2.7.1 Concatenation approach

Since we want to retain information about which model each score came from, in order to capture the strengths of both models, we experimented with concatenating the scores from each model into a 40-dimensional vector (rather than adding, which would lose this contextual information).

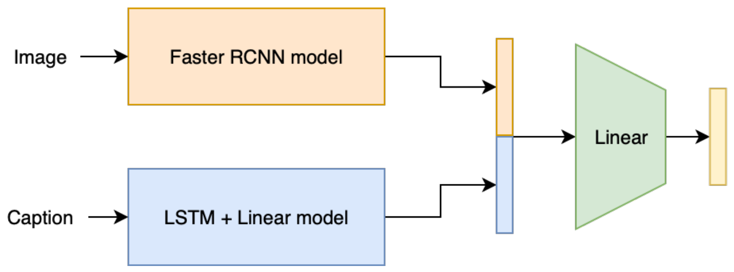


Figure 15: Overview of the approach to combine the outputs of the models with concatenation. The output is passed through a sigmoid function and used as described above.

Let the scores from the object detection model be $O(i)$ and the scores from the language model be $L(t)$. Then the concatenated scores are passed through a linear layer with dimensions to reduce this vector to the same dimensionality as the number of classes. A sigmoid layer is used to produce the output.

$$out_{multi-modal,\ concat} = \sigma(W^{40\times 20}[O(i); L(t)])$$

2.7.2 Bilinear approach

The second approach for combination was to use a bilinear layer, which is a layer of three dimensional weights which aggregrates the information from each input vector into the appropriate output dimensions, providing a strong mechanism for learning the feature correspondence between the two vectors.

$$out_{multi-modal,\ bilinear} = \sigma(O(i)W^{20\times 20\times 20}L(t))$$

3 Experiments

3.1 Environment

These experiments were predominantly executed in a Google Colab environment, with a Tesla P100 GPU instance. For the exact software environment, please see the Appendix section on 'How to run our code.' Our final model, the combination of a Faster R-CNN and LSTM language model, has an average runtime of 38 minutes per epoch in this environment.

3.2 CNN Baselines

For initial experimentation with CNN image classification models, we trained a linear layer mapping from the output features to scores for the 20 labels, with the aim of verifying whether a CNN-based approach was the most feasible. These baseline results are listed in Table 1 below. Amongst all five experimented CNN models, ResNext has the highest weighted F1 score. The training on DenseNet outputs the highest Micro F1 score and Macro F1 score. Clearly, the results of the pretrained Faster R-CNN model with a similar approach were more promising, so further experimentation was focused on improving that model instead by merging it with language models.

Model	**Weighted F1**	**Micro F1**	**Macro F1**
AlexNet	0.6946	0.6653	0.3828
VggNet	0.7440	0.7200	0.4794
ResNet	0.7107	0.7021	0.4914
DenseNet	0.7827	**0.7446**	**0.5112**
ResNeXt	**0.8013**	0.7367	0.4952

Table 1: Initial baseline results using a linear layer to map from CNN outputs to scores.

3.3 Results and Ablation Study on Faster R-CNN with Language Models

As described in Section 2.5.2, the scores from a Faster R-CNN model pre-trained for object extraction were converted to scores for our labels using a Linear layer. This approach demonstrated significantly better results than the CNN baselines.

While not reaching the same levels of performance as the Faster R-CNN model, both the language models we tried (LSTM and TFIDF+Linear) demonstrated significant predictive power, with the LSTM model producing slightly better F1 metrics.

The best models we obtained were multi-modal models combining the scores from the Faster R-CNN model and the LSTM language model as described in Section 2.7. The concatenation method of combining the scores was slightly better than the bilinear approach, reaching a weighted F1 score of 0.9120 on the validation set.

An ablation study on the results of each of these components and their combination is presented in Table 2.

Model	Weighted F1	Micro F1	Macro F1
BiLSTM	0.8426	0.8166	0.6504
TFIDF	0.8339	0.7973	0.6096
CNN (best)	0.8013	0.7367	0.4952
Faster R-CNN	0.9017	0.8981	0.7478
Faster R-CNN + BiLSTM Concat	**0.9120**	**0.9080**	**0.7625**
Faster R-CNN + BiLSTM Bilinear	0.9104	0.9071	0.7619

Table 2: Validation set performance of various models, separated into language-only, vision-only and multi-modal groups.

3.4 Parameter Optimisation

Parameter analysis was performed on models to achieve the results above. Notably, since the R-CNN approach clearly outperformed the CNN approaches, the CNNs were excluded from the parameter optimisation.

The best parameter values obtained for the object detection and language models were used to inform the choice of parameters for these models as components in the combined multi-modal model.

3.4.1 Faster R-CNN

The Faster R-CNN model we used [25] was already pre-trained on the `train2017` subset of the COCO dataset, and used a linear layer to map scores from the R-CNN model to the 20 classes of our dataset. This linear layer serves two purposes:

1. It accounts for co-occurrence of labels in the dataset, as each output neuron is expressed as a linear combination of the R-CNN output scores. As a practical example of why this is important consider images containing a traffic light. Given they contain traffic lights, it would be reasonable to expect them to contain cars as well.
2. It allows each class score to be scaled by its weights, which is important as some labels are associated with less common objects which often have a lower R-CNN output scores. This also allows us to leverage an unweighted BCE loss rather than a weighted one, as discussed in Section 2.3.

Since we use a Linear layer to map from the scores of the model for its large label set to our 20 classes, the crucial parameter to optimise for the performance of the R-CNN model in practice is the threshold, for which classes with higher sigmoid outputs are output as label predictions. With a threshold too high, the model may be too conservative, and typically output less labels than the ground-truth. Conversely, if the threshold value is too low, the model may output many incorrect labels.

The performance of using different threshold values for the same model was evaluated on the validation set, and the results are presented in Table 3. We found that 0.5 and 0.625 were the most appropriate thresholds.

Threshold	Weighted F1	Micro F1	Macro F1
0.25	0.8854	0.8859	0.7397
0.375	0.8967	0.8952	**0.7479**
0.5	0.9017	**0.8981**	0.7478
0.625	**0.9044**	0.8973	0.7423
0.75	0.9005	0.8900	0.7271

Table 3: Validation set performance of our RCNN model trained for 13 epochs, using different thresholds to produce the labels from the sigmoid outputs.

3.4.2 TFIDF with Linear layer

A crucial parameter to optimise for the TFIDF model is the vocabulary size. A large vocabulary increases the dimensionality of the TFIDF vectors and hence the weights of the Linear layer, and may make training more difficult. However, with a vocabulary size too small the necessary words from the caption to properly extract the labels may be missing. A vocabulary of size n is determined by taking the most frequent n words among all the captions, excluding stopwords and with plurals stemmed. We trained TFIDF+Linear models with vocabulary sizes of 100, 500, and 1000, with results presented in Table 4.

Vocabulary size	**Weighted F1**	**Micro F1**	**Macro F1**
100	0.8139	0.7556	0.5209
500	0.8340	0.7989	0.6145
1000	**0.8348**	**0.8009**	**0.6209**

Table 4: Validation set performance of our TFIDF model trained for 15 epochs with different vocabulary sizes.

3.4.3 LSTM

For the LSTM language model, we experimented with the size of the hidden cell state representation in each direction to ensure the BiLSTM model most effectively captures the necessary context to predict the labels. As presented in Table 5, the F1 scores did not increase with the size of the hidden representation, with 128 being the optimal value across the metrics.

Hidden size	**Weighted F1**	**Micro F1**	**Macro F1**
64	**0.8446**	0.8128	0.6467
128	0.8426	**0.8166**	**0.6504**
256	0.8429	0.8117	0.6481

Table 5: Validation set performance of our LSTM language model trained for 5 epochs with different hidden cell state sizes.

3.4.4 Our Best Model: Faster R-CNN with LSTM

Number of Epochs The trend of the weighted F1 score on the validation set was observed during training to determine the appropriate stage to stop training the model. As can be seen in Figure 16, the weighted F1 score was close to its optimal value after the first epoch of training. After the second epoch, the weighted F1 score began to lower, suggesting that 2 epochs is the optimal stopping time. Thus, for our final model to be used on the entire train set, we trained up to 2 epochs.

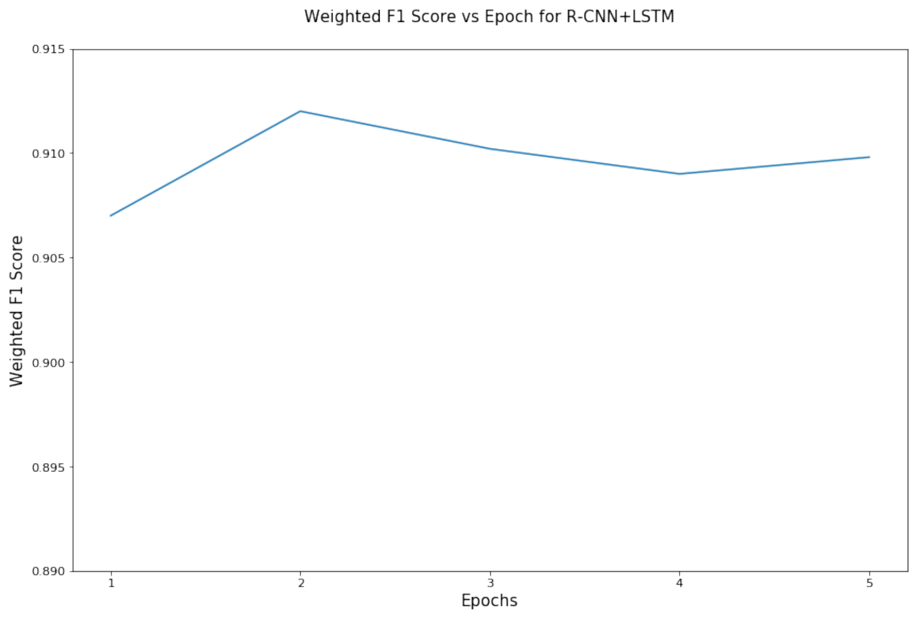


Figure 16: Weighted F1 score during training for the R-CNN+LSTM model, as measured in the validation data.

Threshold values The threshold parameter is also highly important for predictions using the multi-modal model. The performance of using different threshold values for the same model was evaluated on the validation set, and the results are presented in Table 6

Threshold	**Weighted F1**	**Micro F1**	**Macro F1**
0.25	0.8979	0.8982	0.7593
0.375	0.9080	**0.9060**	0.7624
0.5	**0.9120**	0.8981	**0.7625**
0.625	0.9111	0.9048	0.7560
0.75	0.9005	0.8971	0.7416

Table 6: Validation set performance of our RCNN+LSTM model trained for 2 epochs, using different thresholds to produce the labels from the sigmoid outputs.

4 Discussion

This investigation demonstrated the significant increase in performance that can be achieved through careful investigation of the appropriate pretrained model to incorporate. Our analysis of the data illustrated the strong correspondence between labels and objects present in images, which motivated our use of an object detection model as the base of our approach. Using this model, as opposed to a CNN trained for a single label classification task saw us achieve F1 scores well beyond what could otherwise be expected. We also demonstrated that tuning of the underlying pretrained model may not be necessary if it is trained on a highly relevant task. Instead, the parameters for additional linear layers can be learned to map the model outputs for its pretrained task to relevant outputs for the task at hand, without the computational expense of retraining large models. It was also demonstrated that information from multiple modalities can be useful together if appropriately combined. Our investigation demonstrated some highly successful techniques for combining modalities, however multi-modal learning is notoriously challenging, so further investigation into reaching the full potential of both information sources would be worthwhile.

5 Conclusion

Through the experiments we have conducted using several commonly used pre-trained models, we concluded that using R-CNN model and LSTM model, this multi-label classification task on COCO dataset performed well, with a best weighted F1 score of 0.9120. It also illustrates that model training using object detection and segmentation neural network models can perform better when combined with a language model in a multi-modal approach.

Future works could focus on incorporating the use of attention into the natural language processing component to better extract the most relevant objects from captions [27]. Some other recent work [18] involving the use of transformers in object detection could also be interesting to apply to this problem.

References

[1] J. P. Chiu and E. Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[3] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[4] R. Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, December 2015.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, June 2014.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, Oct 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[11] M. Jabreel and A. Moreno. A deep learning-based approach for multi-label emotion classification in tweets. *Applied Sciences*, 9(6):1123, 2019.

[12] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[14] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[17] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, 2017.

[18] G. S. N. U. A. K. Nicolas Carion, Francisco Massa and S. Zagoruyko. End-to-end object detection with transformers. Facebook AI, 2020.

[19] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[20] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[21] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[24] Torch Contributors. torch.nn, 2019.

[25] Torch Contributors. torchvision.models, 2019.

[26] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[28] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

6 Appendix

6.1 How to run our code

The main entry point for our code is the `main.py` file.

Before starting please ensure you have the correct environment set up by installing the requirements file. This software has been tested on Python 3.8. You may wish to create a python virtual environment `https://docs.python.org/3/library/venv.html` to avoid altering your primary python environment. You can install the requirements file from the root directory of the project by running `pip install -r requirements.txt`.

Next, please download the dataset, unzip it and place it in the 'input' folder as follows:

```
|_ code
    |_ algorithm
    |   |_ main.py
    |   |_ ...
    |_ input
    |   |_ data
    |   |   |_ 0.jpg
    |   |   |_ ...
    |   |   |_ 39999.jpg
    |   |_ test.csv
    |   |_ train.csv
    |_ output
    |   |_ predicted_labels.txt
    |_ results
        |_ ...
```

With the appropriate requirements and data installed, please change into the `code/algorithm` directory.

To train our best multimodal model for the optimal number of epochs, you can simply run

```
python main.py
```

Other type of models can also be run from this file by passing in the argument `--model-type` with one of the following options:

```
'rcnn_lstm',
'rcnn',
'lstm',
'tfidf',
'rcnn_lstm_bilinear',
'resnet',
'alexnet',
'vgg',
'densenet',
'inception',
'googlenet',
'resnext'
```

If you wish to test the model and produce an output file with the predictions, you can pass the `--test` flag. The `--epochs` argument can also be specified to alter the number of epochs to train the model for. For example, to train an LSTM model for 5 epochs and produce test outputs at the end of training, you can run:

```
python main.py --model-type lstm --test --epochs 5
```

A variety of other options can be passed to the program entry point in this fashion. All of the arguments have default values. To get help running the program using the arguments, you can call `python main.py -h`. A summary of the available arguments is given below:

General arguments/flags
`--test`: produce test outputs for the model after training.
`--load`: (string argument) Load model state from the specified .pt file, relative to `--checkpoint-dir`.
`--nosave`: don't save the model state during or after training.
`--noval`: train with the whole train set and don't perform validation.
`--name`: (string argument) A unique identifier for the current run, used for saving logs and model state checkpoints.

Config arguments/flags
`--data-dir`: (string argument) The directory to load the dataset from.
`--output-dir`: (string argument) The directory to save test set predictions to.
`--log-dir`: (string argument) The directory to save logs to.
`--checkpoint-dir`: (string argument) The directory to save model checkpoints to.

Model arguments/flags
`--model-type`: (string argument) Which model type to run (takes values listed earlier).
`--batch-size`: (int argument) The batch size to use when training.
`--epochs`: (int argument) The number of epochs to train the model for.
`--threshold`: (float argument) Threshold for sigmoid output to be predicted as a label.

6.2 Pretrained R-CNN Label Names

Class ID to label names of the first 20 classes returned by the Faster R-CNN model pretrained on COCO `train2017` in the `torchvision.models` package [25]. Note that the `'__background__'` and `'N/A'` labels in the mapping below most likely correspond to labels 0 and 12 in the training dataset, both of which had zero count in the training set as described in Section 2.1.

```
 0: __background__,
 1: person,
 2: bicycle,
 3: car,
 4: motorcycle,
 5: airplane,
 6: bus,
 7: train,
 8: truck,
 9: boat,
10: traffic light,
11: fire hydrant,
12: N/A,
13: stop sign,
14: parking meter,
15: bench,
16: bird,
17: cat,
18: dog,
19: horse,
20: sheep
```