COMP5329 - Assignment 2

ANDREA BOSIA: 520600843
FLORIAN GERON: 520354511
YASHAR TAVASOLI: 510570756

1 INTRODUCTION

In this report, the development of a neural network in Python using the PyTorch package for a multi-label classification task is discussed. The data that needs to be classified has a bi-modal input: An image and a short text that describes the image. Each image-text pair has a (set of) label(s) attached to it, which is the output of the neural network and thus what we are trying to predict.

The data set is given to us by the teaching team. This data set contains 30,000 training examples and 10,000 examples for testing. Each example consists of an RGB image as well as a caption. There are 18 target labels, the meaning of which has not been disclosed to us. Each image-text pair has at least one label attached to it.

The aim of the study is to develop a neural network that is able to predict the relevant labels for an image-text pair. In order to achieve this goal, we split the task in two separate challenges: Developing a multi-label image classification model and developing a multi-label text classification model. The results of these two models are combined using an ensemble method. The performance of these models has been improved by means of hyperparameter analyses and ablation studies.

We leverage a CNN network to realize the image classification. Our model is built on top of the ResNet-34 model. This base model is a pre-trained network containing several convolutional layers. In this report we describe two different approaches that we compare side by side. Both leverage the advantages offered by transfer learning; however with the first approach we freeze most of the network and queue some additional fully connected layers. The alternative strategy consisted instead in retraining the whole network originally initialised with pre-learned weights.

To realize the text classification, an LSTM network is developed. The problem of predicting a label for a sentence can be considered a many-to-one problem, where there are many words in the sentence that lead to one label being predicted. LSTM networks are widely used for these types of problems, as they are able to realize long-term dependencies, meaning that the first word of a sentence can be remembered and put to use in the prediction of the label at the end of the sentence.

This study is important because of several reasons. Firstly, it gives the students hands-on experience in implementing neural network for image and text classification using PyTorch. PyTorch is a leading framework for implementing neural networks and mastering this package is essential in developing practical skills for deep learning engineers. Secondly, the problem is unique because of the fact that it combines two distinct challenges: On the one hand, the input is bi-modal, consisting of both an image and a caption. On the other hand, the output can consist of multiple labels, rather than just one class. These challenges make this task significantly different from other classification tasks. This means the students have the opportunity to engage in research and engross themselves in deep learning literature. Thirdly, the students are able to assess the influences of certain hyperparameters and modules on the performance of image and text classification networks, as hyperparameter analyses and ablation studies are an essential part of this assignment.

This report consists of this introduction, followed by a literature study in chapter 2. Chapter 3 covers the exploratory data analysis of the data set and chapter 4 explains the preprocessing steps that were implemented on the data. In chapter 5, the techniques that were used to develop our model are presented and their implementation is discussed. In chapter 6, the hyperparameter analyses and ablation studies are discussed. The results of these experiments are discussed in chapter 7. Chapter 8 contains a reflection on this project. Finally, an appendix is provided, which contains a link to the code as well as instruction for running the code.

2 LITERATURE STUDY

The development of a bi-model multi-label learning model is not a new challenge. Adding labels to image-text pairs has a variety of applications, such as when adding tags to captioned images on social media. Our challenge boils down to two separate challenges: Firstly, we are dealing with multi-label classification, rather than a single-label, multi-class classification problem. Secondly, our data is bi-modal, meaning it is described both by an image as well as a caption. This will require us to make some adaptions to the algorithms that we have seen in this course so far.

2.1 Modelling label dependencies

Deep convolutional neural networks (CNNs) are regularly employed for single-class image classification problems. These models can be extended to perform a multi-label classification, by simply creating multiple single-label classification models, one for each potential label. This approach does not capture the dependencies between the labels. The labels of most multi-label classification problems show strong dependencies between the labels [1]. This means that a lot of information is not being put to use if the label dependencies are not properly modelled.

Several approaches have been put forward to model the label dependencies, such as training a chain of binary classifiers. This method boils down to designing binary classifiers for all the labels, where each classifier takes as its input the input features as well as the already predicted labels, as discussed in [2]. This approach has a high computational cost for large numbers of labels.

Several other methods for modelling the dependencies between labels in multi-label classification problems have been proposed. Some work on the basis of graphical models, such as [1]. Another approach which combines multiple binary classifiers is the Rank-SVM method proposed in [3]. Wang J. et al, noticing that these approaches become prohibitively computationally expensive for large numbers of labels, proposed another approach for dealing with multi-label

image data with the CNN-RNN framework [4]. This framework leverages the memory of recurrent neural networks (RNNs) to employ the dependencies of the labels in an image. These RNNs are combined with deep CNNs to create the CNN-RNN framework which learns a joint image-label embedding to capture the semantic label dependency. This framework integrates attention models into its structure; This framework is built upon the premise of detecting the most important label first, and then shifting the attention to progressively smaller ones. In the paper, the example is given of recognizing an airplane first, followed by the runway, then a person, then a hat.

An improved channel-wise attention mechanism has further been proposed to improve the performance of this framework [5]. This method creates regional attention maps and connects these to labels. Secondly, this approach leverages the semantic meaning of the labels by extracting semantic information out of them using large pre-trained language model such as BERT [6] and ELMO [7].

2.2 Dealing with Bi-Model input

The challenge of categorizing data that consists of both an image and a caption can be approached by decomposing the problem: An image classification model is developed based the images that are given and a text classification model is trained on the captions. The challenge lies in devising a strategy to combine the predictions of these two models. A fusion process is defined in [8]. This paper focuses on the case of a multi-class classification problem with eight possible classes, of which only one can be chosen. The image model returns a probability vector, as does the text classifier. The final result is obtained by adding the two probability vectors together, each having a certain weight. Importantly, these vectors are the outputs of the sigmoid layers, rather than the actual predictions.

This process is shown in formula 1 in the case of two models: A image classification model and a text classification model. P_{img} refers to the output of the sigmoid layer of the image model and P_{text} to the output of the sigmoid layer of the text model. They are combined into a final probability prediction P_{final} using λ as the weight for the image model and $(1 - \lambda)$ as the weight of the text model. In the paper, a multi-class classification is discussed, so the maximum position is taken.

$$P_{final} = max_c[\lambda * P_{img}(v) + (1 - \lambda) * P_{text}(t)] \tag{1}$$

This process can be expanded for any number of models, as shown in formula 2, where there are i models defined and with $\sum_{n=1}^{i} \lambda_n = 1$.

$$P_{final} = max_c[\sum_{n=1}^{i} \lambda_n * P_n(data)] \tag{2}$$

This approach is an alternative approach to the technique of creating an ensemble model by majority voting. In majority voting, an uneven number of models are created and trained. Each of these models makes a prediction on the testing data. In the end, a final prediction for each testing example is obtained by looking at the predictions of the separate models and choosing the prediction which has the largest number of 'votes'. In the case of this assignment, each label would be inspected for each example; If the majority of models agree that this label should be present, then the label is included in the final prediction. The approach presented in [8] seems superior to the method of majority voting due to two reasons: Firstly, it works also when there is an uneven number of models developed. Secondly, the methods in [8] takes into account how certain a model is about its prediction. For example, if a model predict that an example has label 1, but the sigmoid layer returned a value of 0.55, then we can say that the model is not very confident in this prediction. If the value was 0.99, then we can say that the model was quite certain. By using the output of the sigmoid layer, instead of only using the final prediction, this confidence level is integrated into the ensemble method.

Alternative strategies for processing bi-modal image-text input for classification have been proposed as well. In [9], the researchers set out to improve the performance of image classification networks by adding natural language supervision to the process. In vanilla image classification models, they explain, a network to extract image features is jointly trained with a linear classifier to predict a label. In their proposed CLIP model, an image encoder is jointly trained with a text encoder. During training, the caption is processed by the text encoder and the model is trained to match the caption with the image. During testing, the descriptions of the target classes are processed by the text encoder and the model once again tries to match the image to a label.

3 EXPLORATORY DATA ANALYSIS

3.1 Labels

The data consists of 29,996 training examples and 10,000 testing examples. In the training set, each example consists of an image and a caption, as well as a number of labels; In the testing set, these labels are omitted and are what we are trying to predict.

There are 18 possible labels. These labels are numbered (1 through 19, with no examples for label 12 existing) and their meaning is not given to us. Each example can have any combination of labels. The distribution of the labels of the training examples is given in figure 1. As you can see, the data is very unbalanced: Label 1 is present in nearly 76% of the training examples, while the next most common label, label 3, is only present in about 15% of the examples. Each of the other 16 labels is present in less than 10% of the examples.

The distribution of the number of labels per example is given in figure 2. The majority of examples only has one label, at almost 65% of cases. Next, 22% of the examples has 2 labels, followed by 8.5% of examples with three labels. The maximum number of labels that are present within one example in the training data is seven. Only 11 examples have seven labels, which makes up 0.04% of the training examples.

In order to get a better feeling with the data, we have tried to discern the meaning of each label. In order to do this, we have plotted 15 examples of each label and 15 examples in which the label is not present in order to be able to compare and contrast. For some labels, this exercise gave us clear results. For label 1, for example, it seems like it is present if there is a (part of a) human present in the picture. This is visible in figure 3. For label 2, the label

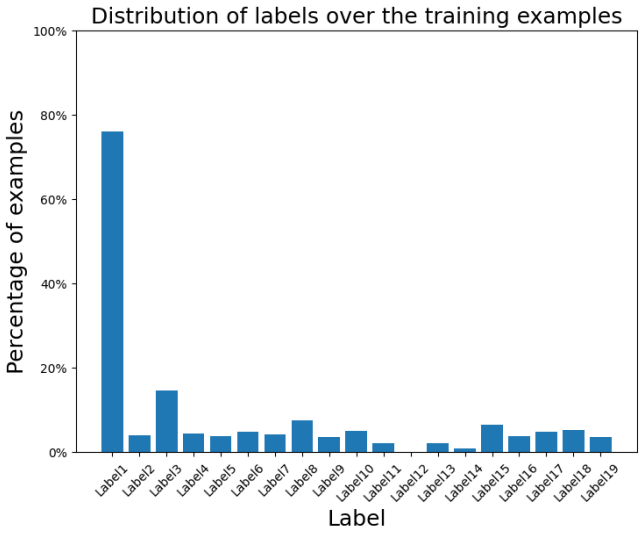

Fig. 1. Distribution of target labels over training examples

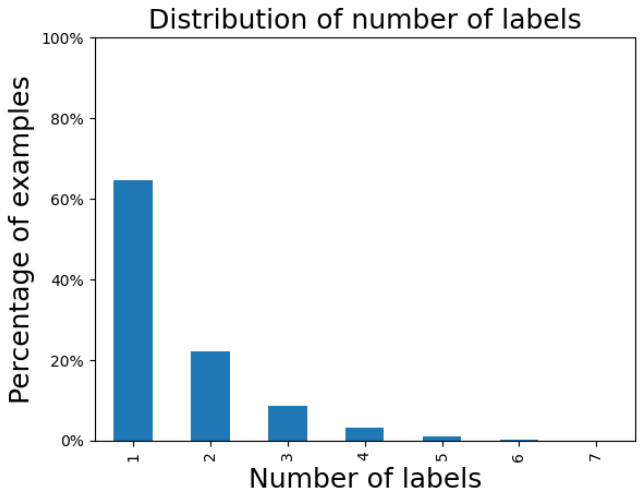

Fig. 2. Distribution of number of labels over training examples

seems to be attached if there is a bicycle in the picture (figure 4). Our best guesses for each of the 18 labels are given in table 1.

From this table, it is clear that label 2 through label 14 describe traffic-related objects, such as cars and traffic signs. The other labels describe objects that we can expect to find on or near the road, such as humans and horses. Therefore, it seems plausible to assume that this data set has been chosen specifically for training a traffic-related machine learning model, such as the object detection model of an autonomous car.

3.2 Caption

Each example is accompanied by a piece of text, i.e. the caption of the image. The distribution of the number of words for each caption (when the caption is split on the white spaces) is given in figure 5. It is clear that most captions have about 9 to 20 words. Each caption has at least 7 words and captions with more than 30 words are exceedingly rare, as there are no more than 2 examples for each number over 30. No caption exists that has more than 49 words.

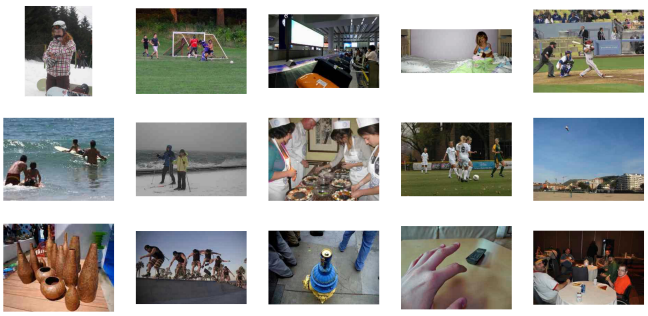

Fig. 3. Images with label 1: There is a human present in every picture

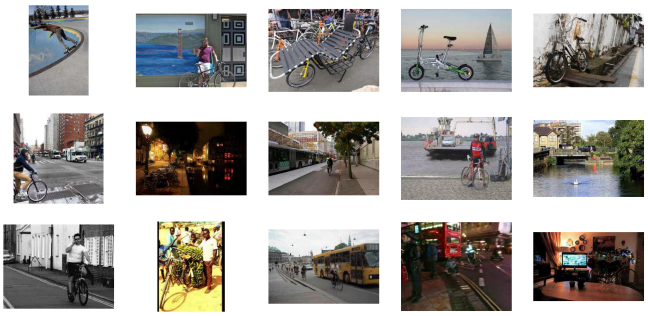

Fig. 4. Images with label 2: Bicycle

Label	Corresponding item
Label 1	Human
Label 2	Bicycle
Label 3	Car
Label 4	Motorcycle
Label 5	Airplane
Label 6	Bus
Label 7	Train
Label 8	Road
Label 9	Boat
Label 10	Traffic sign or traffic light
Label 11	Fire hydrant
Label 13	Red traffic sign (like "stop" or "no entry")
Label 14	Parking meter
Label 15	Bench
Label 16	Bird
Label 17	Cat
Label 18	Dog
Label 19	Horse

Table 1. Item that is present in the picture for each label

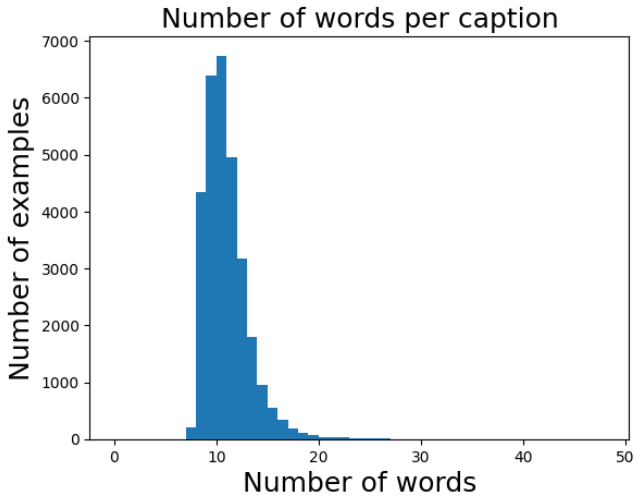

Fig. 5. Distribution of number of words over training examples

4 DATA PREPROCESSING

4.1 Images

The dataset directory is structured with a main folder containing "train.csv", "test.csv" and a subfolder, "data". The first two files are loaded as a data frame and consists of the column "Label", form which dummies are obtained, and "ImageID" column that contains the url to the image relative to the specific sample. Every image is stored in the "data" subfolder in a .jpg format. As a first step two lists, datatrain and datatest are created each containing a label vector and the image path for each sample.

The DataLoader object from torch.utils library is used to turn the train and test lists into tensors, on such tensors some transformations are performed using the 'transform.Compose' method. More specifically, three steps of preprocessing are implemented: First, all the images are resized to 224x224 pixels. In this step, there is no cropping or padding, rather the images are stretched or shrank along certain dimensions to obtain a square shape. The image is then reduced to have 224 pixels along each dimension. Secondly, this array is transformed into a tensor. Thirdly, the images are normalized to have a mean and standard deviation of 0.5 for each RGB channel.

We have done some experiments with augmenting the image data set. In a first experiment, we duplicated each example that has multiple labels into several examples that only have one label. This did not result in an increase in performance but did increase the training time significantly, so we have decided not to include it. Other methods of image augmentation were considered, such as duplicating images that have few examples but having them rotated, flipped, or otherwise slightly amended. We ended up not implementing these image augmentation methods because of the fact that they would lead to a significant increase in training time. Given the time available for this project, we decided this trade-off was not worth it.

4.2 Captions

In order to improve the performance for the text classifier, some steps of preprocessing have been implemented on the captions. Three different steps have been implemented: General cleaning, removing stopwords, and stemming the words.

Firstly, the text is cleaned. Concretely, this means that all upper case letters are converted into lower case letters and that all punctuation is removed.

Secondly, the stop words are removed from the text. For this purpose, we have made use of the NLTK module [10]. It has defined a number of words in the English language that are considered stop words. As these words usually do not add any meaningful content to a sentence, they are removed from the captions. Due to this reduction in words, the distribution of words per caption now looks like figure 6. There are now no captions with more than 28 words, and captions with ore than 20 words are exceedingly rare, as there are only 5 examples of this.

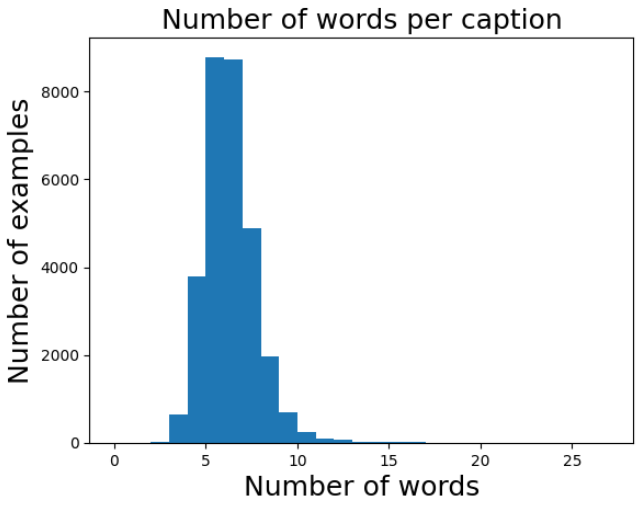

Fig. 6. Distribution of number of words after removing stop words

Thirdly, the words are stemmed. Here too, we make use of the NLTK module. Its SnowballStemmer module takes as its input an English word and outputs the stem of that word. This way, we ensure that words like "enjoy" and "enjoys" are considered the same word.

The first four examples of this process are given in table 2. Some of the captions are shortened considerably, like in the case of the third caption, while some of the captions were already written exclusively in lower case letters and without punctuation, like the fourth example. This shows that this step of preprocessing has a uniforming impact on the captions.

Using these cleaned, reduced, and stemmed captions, a dictionary is created in which each unique word obtains a number. There are 4791 unique words. Adding a number for '–OOV–' (out of vocabulary) and '–PAD–' (padding), we obtain a dictionary with 4793 entries. Using this dictionary, each caption is translated into a numerical equivalent. Thus, for each example, a list of numbers that

Original text	Processed text
Woman in swim suit holding parasol on sunny day.	woman swim suit hold parasol sunni day
A couple of men riding horses on top of a green field.	coupl men ride hors top green field
They are brave for riding in the jungle on those elephants.	brave ride jungl eleph
a black and silver clock tower at an intersection near a tree	black silver clock tower intersect near tree

Table 2. Examples of text preprocessing

represent the caption is produced. This list is extended with padding until a length of 20 is obtained for examples that have less than 20 words. If the example has more than 20 words, than the list will be reduced to its 20 first words. As stated previously, this only happens in 5 of the training examples. These lists of length 20 will be the input of the model for text classification.

5 TECHNIQUES

5.1 CNN for image labelling

5.1.1 Overview of the approach.

To tackle the multi label image classification task we made use of transfer learning in order to build an effective CNN. The following decision is justified by the different advantages that come along with such a method. First and foremost, this allow us to exploit the features extraction capabilities of a network trained on a rather big dataset, namely ImageNet. Moreover using a knowingly effective architecture for the feature extraction part of the neural network saved us from having to experiment many different architectures. Once outlined the main idea regarding how to structure the CNN a big effort was then devoted to ablation studies aimed at identifying the most promising techniques and models to be applied in this specific setting in order to maximise the performances. Indeed many slightly different approaches are adopted along the whole pipeline from the pre-processing to the training procedures implemented. As for the implementation, the PyTorch framework was used.

5.1.2 CNN architecture.

Given that the approach followed is that of transfer learning an architecture had to be chosen. We focused on the ResNet models family, in particular we adopted the architecture proposed with ResNet34 in the paper [11]. The decision was led by the memory constraint imposed by the guidelines of the assignment as well as our effort in developing a CNN with a contained size and a reasonable training time. Moreover, ResNet34 still yields impressive performances, even if it does no longer represent the latest model developed in the ResNet models family. The original idea that drove the development of this model was to address the issue of the vanishing gradient in deep neural networks. The work on RestNet indeed intended to allow to have neural networks as deep as desired without causing the gradient to vanish. As clear from figure 7, ResNet34 building blocks are 4 groups of residual layers which are anticipated by a convolutional layer and a max pooling layer.

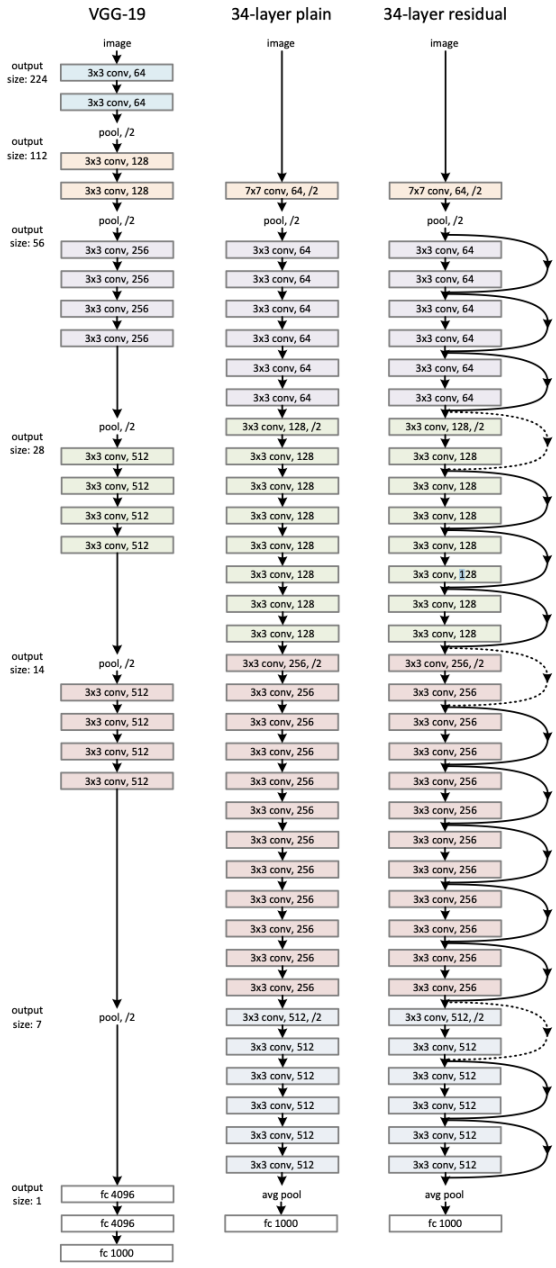

Fig. 7. Comparison of of three different architectures, all embracing the general idea of "layers stacking". Figure taken from [11].

The deepest layers, implemented after the 4 residual blocks are an average pooling layer, that reduces the spatial dimension of the feature maps produced, and a fully connected layer used for the classification on the original dataset on which the model was trained. It can be noticed how this architecture somehow mimics that presented by VGG since layers are simply stacked on each other.

However the innovation introduce with ResNet models lays in the "identity shortcut connections", also known as residual connections. That is, every 2 convolution layer, we bypass one with a skipped connection.

In our implementation we loaded the ResNet34 architecture together with the weights derived from the training on ImageNet and used them as initialisation. From here on two alternative architectures were explored and compared in the attempt of identifying the most promising of the two for this specific scenario.

- **Architecture A**
  On one hand, we only modified the last fully connected layer of ResNet34 so that its output matches the number of possible labels present in our data set. In this case, the parameters of the ResNet34 were kept trainable.
- **Architecture B**
  Alternatively we queued several fully connected layers (with ReLu activation function) to the ResNet34 base model. In this case, the base parameters were left untouched, i.e. they were frozen.

These two slightly different architectures are the direct consequence of two different training procedures that are introduced in section 5.1.3 . However for both implementation the final output is fed to a sigmoid function in order to obtain the final predictions. Following such construction we obtained a predictive model where every value of the output vector represents the probability that the label corresponding to a given index is present in the image. The threshold used to determine whether a label is identified or not in an image is a parameter the required tuning.

5.1.3 CNN training.

In the training of the CNN described we explored two main alternative approaches adopted on the two slightly different architectures presented in the previous section.

For both cases we downloaded ResNet34 together with the weights derived from the training on ImageNet. In architecture A, we used the weights learned from the training on ImageNet just as an initialisation. Hence, the whole neural network was retrained on our dataset. This technique is a lot more computationally expensive but allows us to retrain the entire model and fine tune it on our specific data, and our specific task. After all, ImageNet is a dataset for single-label multi-class image classification, rather than multi-label classification; It is not unreasonable to assume that this difference in objective might impact the data itself.

In architecture B, we froze all the layers of the ResNet34 base network. This approach seems promising as the base network is able to extract the high level features of the image. It is hard to say for sure, but these features could correspond to objects such as "wheel" or "hand". Combining these features into label predictions using several fully connected layers thus seems like a reasonable approach. The big advantage is that only a minor part of the architecture needs to be trained, namely the last fully connected layers added on top of the ResNet34 network. This drastically decreases the number of trainable parameters and the total training time with it. Because of the decrease in computational complexity, we opted to use architecture B. Using the ResNet34 base model with three fully connected layers on top of it resulted in a model of 90.6 MB in size and a performance of just under 80%.

5.2 LSTM for text labelling

An LSTM (Long-Short Term Memory) model is developed for the multi-label classification of the captions. It is a kind of recurrent neural network that has been developed to be able to deal with long-term memory better than the simple RNN framework. While a simple RNN maintains a hidden state to model memory, it is not able to hold this hidden state over long stretches. Furthermore, it suffers from vanishing and exploding gradients. The LSTM improves on the RNN model by making use of three gates: A forget gate, an input gate, and an output gate. Using these gates, the LSTM model is able to learn long-term dependencies. Furthermore, it is less susceptible to the issue of vanishing and exploding gradients.

The LSTM model that has been implemented for this assignment consists of three distinct layers:

- **An embedding layer**
  This layer converts the words of the captions into word embeddings. The in-built embedding methods of PyTorch are used for this purpose. An embedding dimension of size 128 has been chosen.
- **The LSTM layer**
  The LSTM layer consists of 20 LSTM units; The first word embedding is fed into the first unit. The second word embedding is fed into the second unit. This unit also receives the cell state that was generated in the previous unit. This goes on until the 20th word embedding is fed into the final LSTM unit, which also receives the cell state of the 19th unit.
- **18 fully connected layers** (one for each label)
  The last LSTM unit produces an output. This output is fed into 18 separate linear layers. A dropout layer with a dropout rate of 0.4 is included between the LSTM unit and each fully connected layer. Each of these 18 layers then produces a single output. This output is fed through a sigmoid function, which produces a probability of the caption belonging to one of the 18 labels. If this probability value is larger than 0.5, then the relevant label is given to the example in question

The resulting model has a size of 4.1 MB and a micro F1-score of over 80%. Considering its size, this is an impressive result. One drawback of the LSTM approach is the amount of time it takes to train the model; It takes about 200 minutes to train the module for 200 epochs, which was the number of epochs needed to obtain the aforementioned performance.

Because of the limited size of the model and the fact that we had about 5 MB to spare before reaching the 100 MB threshold, we have decided to train a second LSTM model and use an ensemble strategy to combine the prediction of the two LSTM models. This doubles the model size from 4.1 MB to 8.2 MB and increased the F1-score from 81% to 82%.

5.3 Ensemble

Our case is different from the one discussed in [8]. First off, we are dealing with a multi-label classification problem, rather than a multi-class problem. Secondly, we have 18 possible labels instead

of just 8. We will therefore adapt formula 1 into formula 3. Each P vector is of size 18 and contains 18 probability values of an example belonging to the 18 labels. If the value for a certain label is smaller than 0.5, then it is not considered to have that label. If the value is larger than 0.5, than the label is predicted for that example.

$$P_{final} = [\lambda * P_{img}(v) + (1 - \lambda) * P_{text}(t)] \quad (3)$$

Different values for the weight variable λ have been tried. The results of these experiments are included in chapter 6.

After combining the outputs of the sigmoid layer in this fashion, probabilities for each of the 18 labels are obtained. Next, a cut-off value is defined and applied to decide whether a probability leads to a positive or a negative classification. A cut-off value of 0.5 makes a lot of intuitive sense (as a probability of less than 50% intuitively means that you should not consider it and a probability of more than 50% intuitively means that it should be considered that class), but some other values are tried as well in chapter 6.

6 EXPERIMENTS AND RESULTS

6.1 LSTM model

Several experiments were undertaken to tune the architecture and hyperparameters of the LSTM model that labels the caption of the image. To have robust results for these experiments, three separate models were trained for each option. The evaluation metrics for these three models are shown in the box plots of each experiment. To keep the code runtime within reasonable bounds, these models were only trained for 20 epochs. These experiments and their results are presented here.

6.1.1 Impact of model architecture.

Three separate models were created: The standard model described in section 5.2, a model two LSTM layers, and a model with bidirectional LSTM units. Using either of the two non-standard options will lead to an increased number of calculations and number of variables. This will have an effect, both on the training time and on the size of the model, as shown in table 3.

Model Type	Model Size
Standard	4.1 MB
Bi-LSTM	5.7 MB
Two layers	6.2 MB

Table 3. Size of the three variations of LSTM models

Furthermore, we can actually expect that using a bi-LSTM will have a negligible impact on the prediction performance of the model. After all, we are only interested in the final prediction that is the output of the final LSTM unit; Adding bidirectionality to the units will not result in any additional predictive power. Therefore, we expect that the bi-LSTM model will have longer training times but no increase in predictive performance. (Bi-LSTM might lead to an improved performance if we combine the output of the final LSTM unit with the output of the first LSTM unit. In order to keep the complexity of the model low, we have not implemented this option.)

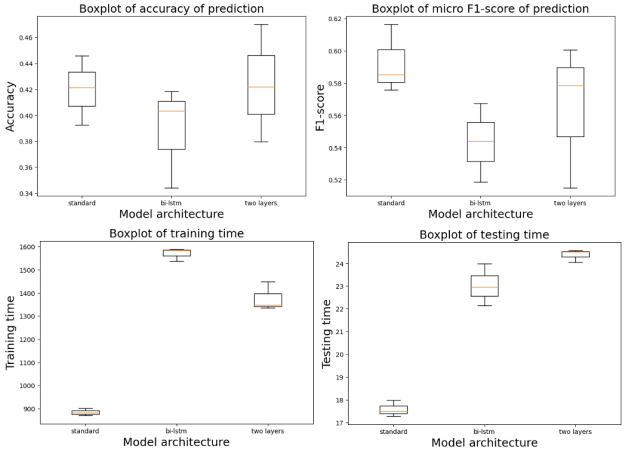

Fig. 8. Impact of model architecture on model performance

In figure 8, we see that the training time and testing time both increase significantly for the non-standard models. This is entirely to be expected. Furthermore, the performance in terms of accuracy and F1-score actually decreases in the bi-LSTM compared to the standard model. Lastly, adding a second layer to the LSTM model seems to have a negligible effect on the predictive performance; The mean accuracy and F1-score is more or less unchanged. As this model has a much larger training time and model size, it is clear we should not use this model version.

6.1.2 Impact of cut-off value for prediction.

After the fully connected layer, a value is provided for each possible label. This value is fed into a sigmoid layer which returns a value between 0 and 1. Next, a cutoff value is used to divide the outcomes in positive and negative predictions. If a cut-off value of 0.5 is taken, then if a value that is larger than 0.5 is returned for a certain label, we conclude that the model predicts this piece of text to have this label. However, we can set this cut-off value at other values.

The impact of this is shown in figure 9 for three different values: 0.25, 0.5, and 0.75. From this figure, it is clear that a value of 0.25 decreases the performance; The average accuracy and F1-score decreases. The testing time decreases a bit as well, but this does not make up for the loss of accuracy. A value of 0.75 returns a slightly higher accuracy and F1-score, while also reducing the testing time. It also reduces the variance in the training time of the model.

6.1.3 Impact of learning rate.

The learning rate determines the size of the step that is taken after the gradient is determined in the backward pass. A large learning rate generally results into fast convergence, but has the risk of resulting in an unstable learning process. Such an unstable process might result in overshooting the optimum or converging to a suboptimal solution. On the other hand, a small learning rate might result in a slow learning process where the network does not converge in the allocated number of epochs.

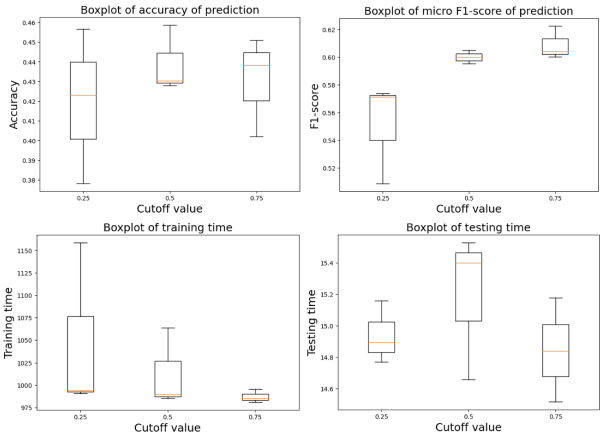

Fig. 9. Impact of cutoff value on model performance

We have tested the model performance for three learning rate values, namely 0.1, 0.01, and 0.001. The results are visible in figure 10. It is clear that the learning rate of 0.1 does not result in a satisfying model: The accuracy and F1-score are far below the other two models. This could be the result of an unstable optimization process. The learning rate of 0.01 and 0.001 result in similar accuracy and F1-scores. However, the training time for the model with learning rate of 0.001 is significantly shorter, making it the preferred learning rate value.

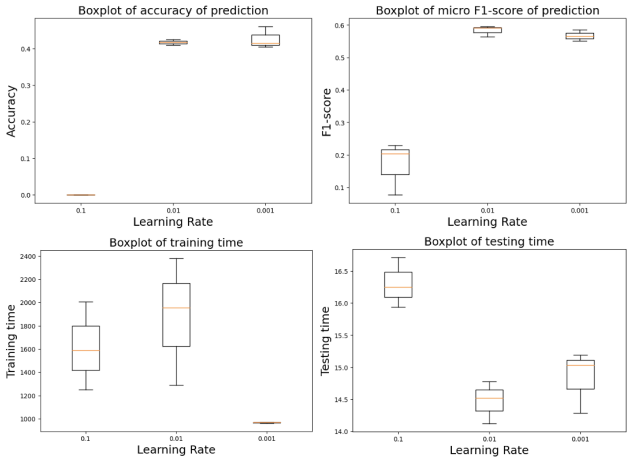

Fig. 10. Impact of the learning rate on model performance

6.1.4 *Impact of the dimension of the word embedding.*

As explained in section 5.2, the first step of the text classification is to create word embeddings for the words in the caption. This word embedding has a dimension that determines how many numbers are used to describe the word's features. We expect that a word embedding of a high dimension can capture more of the meaning of the word, but results in higher computational cost. The reverse holds for a word embedding with a small dimension.

Tests were carried out to assess the model performance for three word embedding dimensions: 32, 64, and 128. The results are given in figure 11. The results are exactly what we would expect: The performance is terms of accuracy and F1-score increases as the dimension increases. However, this also results in higher training times and testing times.

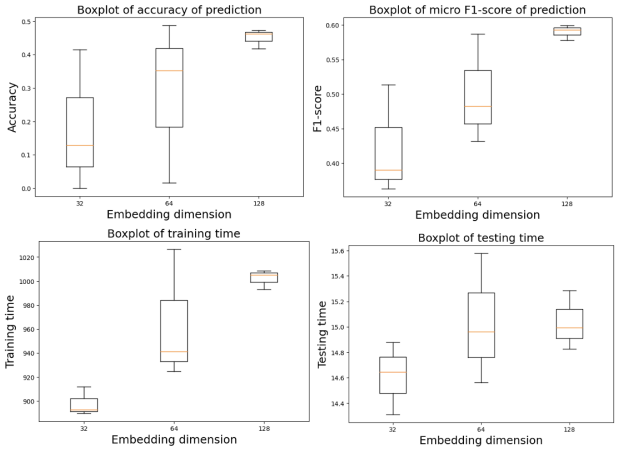

Fig. 11. Impact of the dimension of word embedding on model performance

6.1.5 *Impact of the dimension of the hidden state of the LSTM unit.*

An LSTM unit contains a hidden state, also known as the cell state, which is responsible for remembering and passing on information between the LSTM units. This state is a vector of a certain dimension. The larger the state, the more information can be captured, but the higher the computational cost for training and using the model. This trade-off is very similar to the trade-off for the word embedding dimension.

Models were trained for three cell state dimensions: 64, 128, and 256. Their performance is given in figure 12. These results are entirely in line with our expectation; The larger the cell state, the better the performance, but the longer it takes to train and test the model.

6.1.6 *Impact of text preprocessing.*

In section 4.2, the preprocessing of the captions was discussed. Three important steps were undertaken: The text is cleaned, i.e. the punctuation is removed and all letters are converted to lower-case characters, the stop words are removed, and the words are stemmed. Experiments have been carried out to assess the impact of these latter two steps on the model performance.

The first model was trained using the standard manner of text preprocessing that is discussed in section 4.2. The second model was trained on text whose stop words were not removed. This had as its major impact that the average text length increased by quite a bit. This impact is visible in figure 13. Comparing this figure to figure 6, it is clear that some sentences are quite a bit longer than

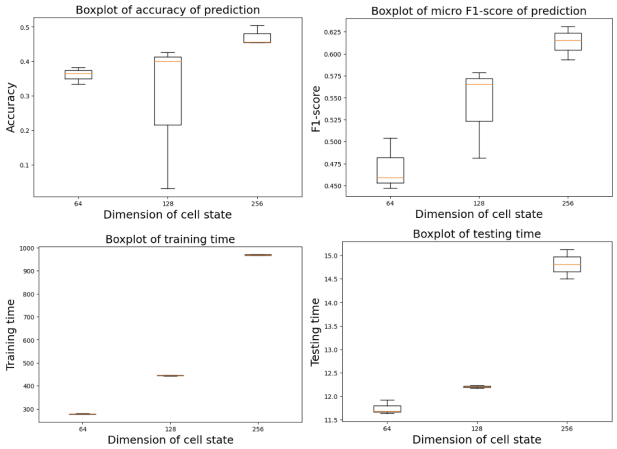

Fig. 12. Impact of the dimension of hidden state on model performance

20 words. However, the percentage of words that are longer than 25 words is still very rare, so the captions were cut off after 25 words (or padded to 25 words if they were shorter than this). The size of the dictionary increased only modestly, from 4793 to 4903. From this, we can conclude that 110 stopwords were removed by the NLTK module.

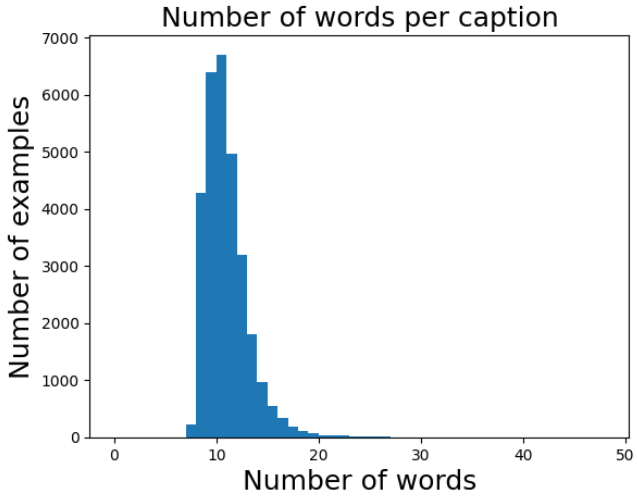

Fig. 13. Distribution of number of words without removing stop words

The third model was trained on text whose words were not stemmed. The major impact that this had was not on the number of words per sentence, but on the number of words in the dictionary; Its size increased from 4793 unique words to 6907 unique words, as shown in table 4. This is an increase of 2114 words, or an increase of about 45%!

The performance of these three models is visualized in figure 14. There seems to be very little impact on the performance of the model in terms of accuracy and F1-score. In terms of training time, it is

Method of preprocessing	Dictionary size
Standard	4793
With stop words	4903
Without stemming	6907

Table 4. Dictionary size for each method of text preprocessing

clear that the standard preprocessing is the most efficient. Including stop words increases the training time a bit, and not stemming the words increases it even more. Interestingly, the testing time for the data set including the stop words is actually the shortest.

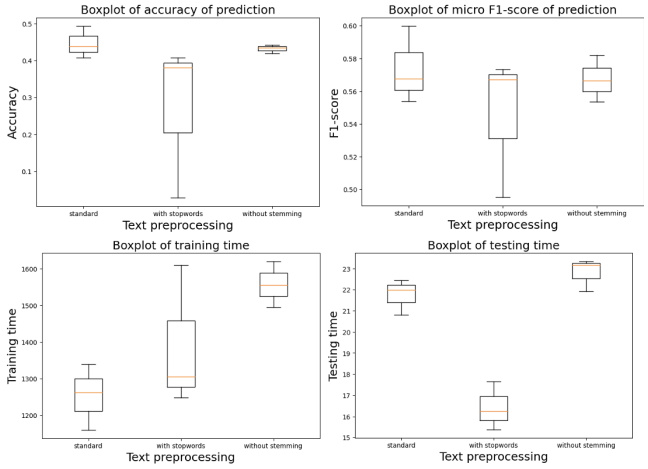

Fig. 14. Impact of text preprocessing on model performance

6.2 CNN model

We have performed a number of different experiments in order to optimise our CNN model for this specific image classification task. To achieve this, we trained the different variants of the model over 20 epochs (to keep the training time manageable). According to the results observed during training and testing we then gradually figured which parameters would outperform the others and converged to our best model.

6.2.1 *Impact of Learning Rate.*

Tuning learning rate in all ML and NN tasks is the essential part of experiments. In this work we compared the results of the model with learning rate of 0.001 and 0.01 as they are most common learning rate used in the literature. The results of F1-score on the test data for these learning rate van be observed in figure 15.

As we can see in the above box-plot, learning rate 0.001 results in an average F1-score of 0.76 in three independent runs whereas learning rate 0.01 only reaches an average F1-score of 0.58 after 20 epochs.

6.2.2 *Impact of cut-off value for prediction.*

The concept of why we need a cut-off value for prediction explained in the experiment section of LSTM experiments. For the

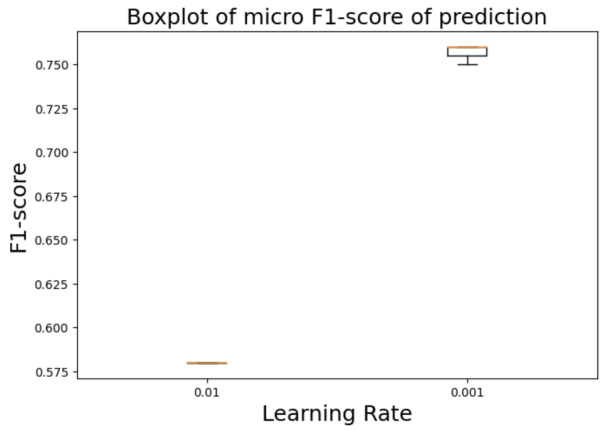

Fig. 15. Impact of learning rate tuning on F1-score

same reason we need to examine different cut-off values for the CNN model to see at what threshold we should catch the highest probabilities as the labels of a certain image. To do this we ran our model three times for different cut-off values. The most important ones are presented in figure 16.

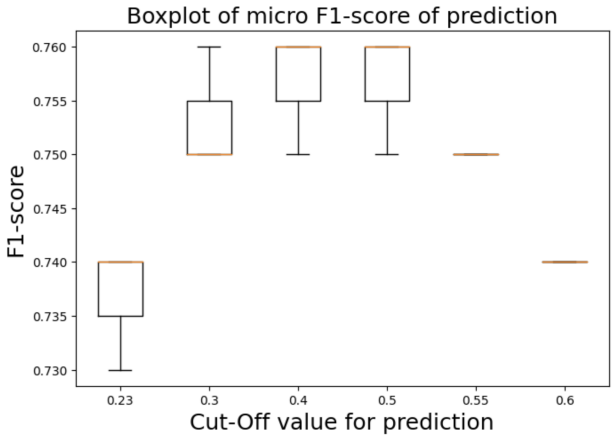

Fig. 16. Impact of different cut-off values on F1-score

As we can see cut-off values below 0.4 and above 0.5 result in lower F1-score on the test dataset. In case of comparing 0.4 and 0.5 as we can see in the above box-plot they show similar performance on the test dataset. Therefore we compared the accuracy of predictions with both scenarios and figured out cut-off value of 0.5 outperform 0.4 with the accuracy on the test dataset. So we chose 0.5 as the cut-off value of our best model.

6.2.3 Impact of optimisation method .

Another parameter that we thought might impact on the results is the optimisation method. For this experiment we compared the results of the model using two of the most commonly used methods Adam and SGD. The F1-score on the test data are reported in figure 17.

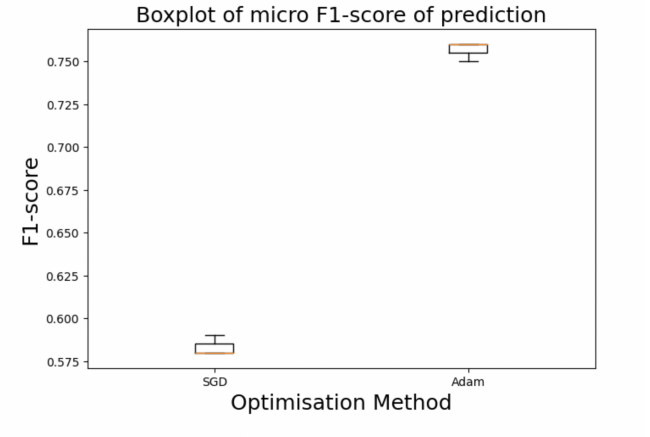

Fig. 17. Impact of optimiser choice on F1-score

Adam optimisation method proved to outperform SGD for our dataset.

6.2.4 Impact of number of training epochs.

Since we are using architecture A, we are only training the three fully connected layers that are added on top of the ResNet34 base model; All the parameters of this model are frozen. Therefore, it is reasonable to assume not a lot of training is necessary: The ResNet34 base model is already capable of extracting the high-level features of the images. The last layers only combine these into predictions. We have done some experiments to assess the performance of the model for different numbers of epochs, the results of which are visible in figure 18. Note, this is the F1-score obtained after testing on a separate testing set.

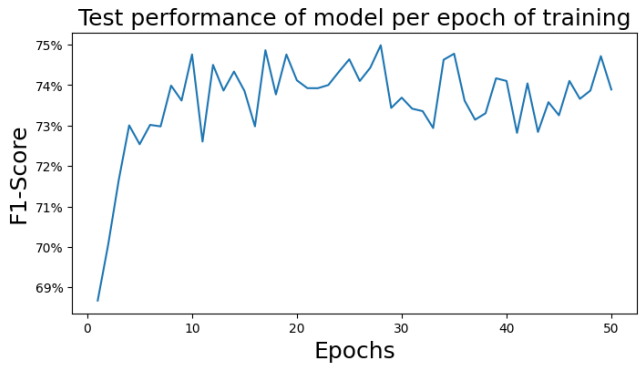

Fig. 18. Test performance of CNN network in function of number of training epochs

From this figure, it is clear that the model does not improve much after 20 epochs of training. In order to err on the safe side, we have done further experimentations with models trained for 80, 100, and 120 epochs. The submission with 120 epochs of training ended being the most high-performing.

6.3 Ensemble model

As for the final ensemble model, only one parameter is left to be tuned: The λ parameter that decides on how much weight to give on each of the two models. A weight of λ is given to the image classification model probabilities and a weight of $(1 - \lambda)$ is given to the text classification model as per formule 3. The results of the tests are visible in table 5. We have therefore chosen a λ value of 0.5.

Lambda value	F1-score
0.4	86.30%
0.5	86.39%
0.6	82.96%

Table 5. Predictive performance of ensemble model for different lambda values

6.4 Final model

Our final result is the output of two distinct models. The CNN model is built on top of ResNet34 and attaches labels based on the images. The structure of the final CNN model is visible in table 6. Its performance is visible in table 7.

Module / Hyperparameter	Value
Base model	ResNet34
Frozen base model parameters	Yes
Number of additional FCs	3
Dropout	0.4
Optimizer	Adam
Cut-off value	0.5
Learning rate	0.001
Epochs	120

Table 6. Design of the best performing CNN image labelling model

Performance metric	Value
F1-score	78.99%
Training time	152min 11sec
Testing time	41sec
Model size	90.6 MB

Table 7. Performance of the best performing CNN image labelling model

The second model is a custom LSTM model consisting of an embedding layer, 20 LSTM units, and a fully connected layer for each label. The final structure and hyperparameters chosen for this model are shown in table 8. Its performance is shown in table 9.

The predictions of these two models are combined using the process of formula 3. The performance and final model characteristics are shown in table 10

Module / Hyperparameter	Value
Stop words	Removed
Stemming	Implemented
Number of LSTM layers	1
Bi-LSTM	No
Cut-off value	0.5
Learning rate	0.001
Word embedding dimension	128
Cell state dimension	256

Table 8. Design of the best performing LSTM text labelling model

Performance metric	Value
F1-score	82.85%
Training time	190min 26sec
Testing time	56sec
Model size	8.2 MB

Table 9. Performance of the best performing LSTM text labelling model (ensemble of two separate LSTM models)

Performance metric	Value
F1-score	86.10%
Training time	342min 3sec
Testing time	97sec
Model size	98.7 MB
Lambda	0.5

Table 10. Performance of the ensemble model (one CNN and two LSTM models)

7 DISCUSSION OF RESULTS

Looking at the two models separately, it is interesting to note that the LSTM network consistently outperforms the CNN network. A single LSTM network produces an F1-score of about 81% and two LSTM networks combined result in a 82% F1-score, which is significantly more than the F1-score of about 79% of the CNN network. This is especially impressive considering the fact that a single LSTM network is only 4.1 MB in size, compared to the 90.6 MB size of the CNN network.

Another factor that makes the result of the LSTM network so impressive, is the fact that the captions do not always contain a direct reference to all of the labels that are attached to the image-text pair. Figure 19 shows ten examples of captions and the meaning of their respective labels. Sometimes, it is fairly straightforward to predict the label of a caption; In the case of caption with index 6, the target label "airplane" is literally featured in the caption, so the LSTM has to remember that specific word. A similar logic holds for "motor bike" in caption 7 or "train" in caption 4. However, there are also cases where this is not that straight forward at all. For example, in caption 9, a scenery is described of a park. It is reasonable to assume that there will be humans, dogs, and benches there, but it is not at all explicitly stated. Even more difficult would be to predict the label "bus" for caption 18 or "car" for label 19.

index	Caption	Meanings
0	Woman in swim suit holding parasol on sunny day.	human
1	A couple of men riding horses on top of a green field.	human,horse
2	They are brave for riding in the jungle on those elephants.	human
3	a black and silver clock tower at an intersection near a tree	road,car,red sign
4	A train coming to a stop on the tracks out side.	road,car,train
5	A young man riding a skateboard into the air.	human
6	A big airplane flying in the big blue sky	airplane
7	A man riding a motor bike across a forest.	human,motorcycle
8	There is a street lined with packed buildings	road,car
9	A skate park next to a body of water and green park.	human,dog,bench
10	Woman cutting pizza with fork and knife sitting next to young girl	human
11	a man and woman cut into a big cake	human
12	A piece of cake and coffee are on an outdoor table.	human,bench
13	People gathered outside in a big field on a cloudy day	human,car
14	A stop sign directs pedestrians as a train travels by.	human,red sign,train
15	a female in military uniform cutting a businessman's neck tie	human
16	Six snowboards are propped in the snow on a rail.	human
17	a small airplane that is on a runway	car,airplane
18	A female traveler leaning on a luggage cart.	human,car,bus
19	The dog is playing with his toy in the grass	dog,car

Fig. 19. Ten captions and their associated labels

This outperformance by the LSTM network also explains why a heavier weighting of the LSTM network by the λ parameter in formula 3 does not significantly decrease the ensemble prediction performance, but a heavier weighting of the CNN probailities does. This is the conclusion of the data in table 5. A λ value of 0.6 significantly decreases the performance of the ensemble result, while a value of 0.4 barely decreases it. In fact, up until the last submission, a λ value of 0.4 actually resulted in the best result.

The major takeaway therefore is that, given the constraints imposed on the model size, the LSTM model does a better job at predicting the labels than the CNN network, even though the image data always contains a direct reference to the labels while the text data does not. The lackluster performance of the CNN network can be explained in several different ways. Firstly, although ResNet34 is considered a very high performing model, it is not quite the state-of-the-art model for image classification anymore. We chose this network because of the size constraints on the overall model. If such constraints were not present, a larger base model would have been chosen. Secondly, due to the constraints imposed on the training time of the model, we decided to only add a limited number of fully connected layers on top of the base model. Increasing the number of layers or the number of neurons per layer would potentially allow the network to combine features more effectively into label predictions. Thirdly, ResNet34 is trained on the ImageNet image data set. This data set contains 14 million images with about 20,000 categories. These categories include categories like "strawberry", which are not especially relevant to our use case [12]. Therefore, ResNet34 captures features that are not particularly useful to us. Therefore, it is reasonable to assume that our performance would have increased if the base model was trained on a data set that is more directly similar to ours, like a data set on traffic-related images and categories, for example. Lastly, augmenting the data set would likely yield a higher performance as well. The data set can be augmented by rotating, mirroring, or otherwise slightly amending the existing images. This would train the model to be able to handle more variation in the images. Doing these augmentations on labels that do not have a lot of examples would help balance the data set as well.

In terms of improving the LSTM performance, we could expect further improvements in several ways: One way would consist of employing a bi-LSTM model where the output of the first and the last LSTM units are combined should result is a model that is better able to utilize information that is mentioned early in the sentence. This would increase the size of the model though, as shown in table 3. It would also be worthwile to analyze the model performance when lemmatization is used as a method of preprocessing rather than stemming. After all, lemmatization has been shown to lead to a better performance for certain NLP tasks [13].

Next to improving the CNN model and improving the LSTM model, additional steps can still be undertaken to improve the process of combining these two models. In this report, the two models are trained separately and their predictive probabilities are averaged afterwards. It would likely be better to integrate both models into a single framework so that they can be trained together. One such framework is proposed in the CLIP paper [9].

Furthermore, the act of combining the probabilities from two separate models can be further optimized as well. Right now, all the probabilities of all the labels are averaged using the same weight. Instead, both prediction vectors could be used as an input for a linear neural network. The output would correspond to the final predictive probabilities. This neural network could then be trained; This would enable the network to decide whether the CNN might be more reliable for certain labels while the LSTM might be more reliable at predicting others.

8 CONCLUSION

In this project, we have created two separate neural networks for a bi-modal multi-label classification problem. The goal is to predict the labels for an image and its caption. There are 18 possible, non-mutually exclusive labels. A CNN model based on ResNet34 was developed for the image classification and a LSTM model was developed for the text classification; The results of these two models are combined into an ensemble prediction.

The LSTM model, with an average F1-score of 81%, consistently outperformed the CNN model, whose performance did not exceed 80%. This is especially impressive, given the difference in model size (4.1 MB for the LSTM model and 90.6 MB for the CNN model) and the fact that the caption often does not include a direct reference to the attached labels. The LSTM prediction was further improved by developing a second LSTM model and combining their results into an ensemble prediction, whose performance reached 83%. Finally, by combining this LSTM result with the CNN result, a F1-score of over 86% was reached.

For each of the two models, hyperparameter analyses and ablation studies were conducted to assess which parameters and which modules would lead to the best performance. Four key metrics were taken into account in these analyses: Accuracy, F1-score, training time, and testing time. The best parameters and modules were included in the final models that are described in chapter 6.4.

9 APPENDIX

9.1 Link to code

For this report, we are submitting three different Google Colab sheets. The first sheet contains the LSTM model and some experiments that were done on it. The second sheet contains the ResNet34 based CNN model. The third sheet contains the calculation of the ensemble prediction.

- First sheet (CNN): Link.
- Second sheet (LSTM): Link.
- Third sheet (ensemble): Link.

The three Colab sheets can also behind in the shared folder which can be accessed here. In this shared folder, you can also find our final submission on Kaggle (called "Predicted_labels.csv") as well as the three final models that were used to make this prediction.

9.2 Instruction for running the code

The code is provided as three separate Google Colab files. To run the code, please run each individual cell sequentially. To do this, you can manually run each cell, or you can choose to run all cells sequentially automatically by pressing CMD+F9 (MAC) or CTRL+F9 (PC).

The second or third cell in the sheets downloads the relevant data from Google Drive. When this cell is run, you will need to give Google Colab permission to access Google Drive. After this, all cells will run without any further input needed from your side.

9.3 Specifications

Google Colab environment: 12GB RAM, 15GB Disk Space, CPU Intel(R) Xeon(R) CPU 2.20GHz, 13GB GPU, Python Version: 3.9.

REFERENCES

[1] X. Xue, W. Zhang, J. Zhang, B. Wu, J. Fan, and Y. Lu. Correlative multi-label multi-instance image annotation. *IEEE International Conference on Computer Vision*, pages 651–65, 2011.

[2] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifer chains for multi-label classifcation. *Mach. Learn.*, 85(3):333–3359, 2011.

[3] A. Elisseeff and J. Weston. A kernel method for multi-labelled classication. *Proc. Adv. Neural Inf. Process. Syst (NIPS)*, pages 681–687, 2001.

[4] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 2285–2294, 2016.

[5] Peng Li, Peng Chen, Yonghong Xie, and Dezheng Zhang. Bi-modal learning with channel-wise attention for multi-label image classification. *IEEE Access*, 8: 9965–9977, 2020.

[6] J. Devlin, M. W. Chang, K. Lee, , and K. Toutanova. Bert: Pretraining of deep bidirectional transformers for language understanding. *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol. (NAACL-HLT)*, 1:4171–4186, 2019.

[7] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol. (NAACL-HLT)*, 1:2227–2237, 2018.

[8] Fangyi Zhu, Zhanyu Ma, Xiaoxu Li, Guang Chen, Jen-Tzung Chien, Jing-Hao Xue, and Jun Guo. Image-text dual neural network with decision strategy for small sample image classification. *Neurocomputing*, 2018.

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[10] Nltk :: Natural language toolkit, 2023. URL https://www.nltk.org/. Accessed: 2023-05-04.

[11] K. He, X. Zhang, and S Ren. Deep residual learning for image recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 70–778, 2016.

[12] Imagenet, 2023. URL https://www.image-net.org/. Accessed: 2023-05-19.

[13] Divya Khyani, Siddhartha B S, Niveditha N M, and Divya B M. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 22:350–357, 2020.